Visual Tracking of Self-Occluding Articulated Objects

James M. Rehg Takeo Kanade

31 December 1994

CMU-CS-94-224

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

A portion of this report will appear in the *Proceedings of the International Conference on Computer Vision*, June 1995, Boston, MA.

This research was partially supported by the NASA George Marshall Space Flight Center (GMSFC), Huntsville, Alabama 35812 through the Graduate Student Researchers Program (GSRP), Grant No. NGT-50559. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NASA or the U.S. government.

Keywords: Model-Based Visual Tracking, Articulated and Nonrigid Object Motion, Occlusion, Human Motion Sensing, Human-Computer Interaction, Gesture Recognition

Abstract

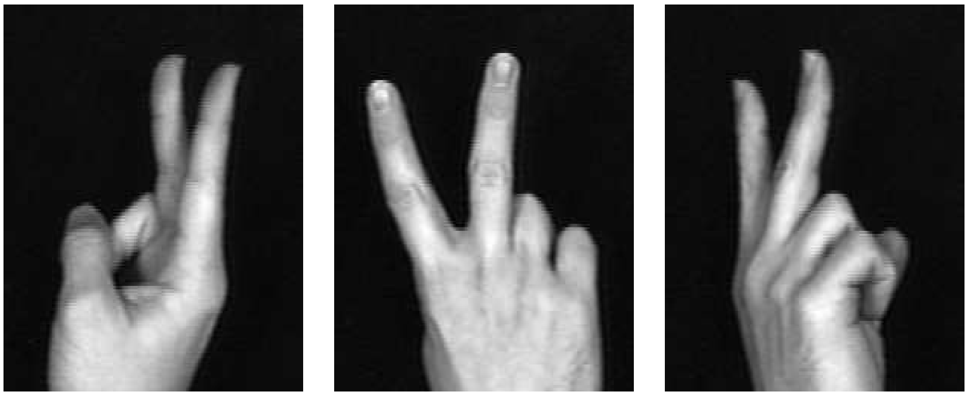
Computer sensing of hand and limb motion is an important problem for applications in human-computer interaction, virtual reality, and athletic performance measurement. We describe a framework for local tracking of *self-occluding* motion, in which parts of the mechanism obstruct each others visibility to the camera. Our approach uses a kinematic model to predict occlusion and windowed templates to track partially occluded objects. We analyze our model of self-occlusion, discuss the implementation of our algorithm, and give experimental results for 3D hand tracking under significant amounts of self-occlusion. These results extend the *DigitEyes* system for articulated tracking described in [22, 21] to handle self-occluding motions.

Contents

1 Introduction

A "human sensor" that tracks a person's spatial motion using techniques from computer vision would be a powerful tool for user-interface, athletic performance analysis, and virtual reality applications. Human hands and limbs can be modeled as systems of rigid bodies connected together by joints with one or more degrees of freedom (DOFs). Thus human sensing can be formulated as the real-time visual tracking of articulated kinematic chains. At a high image sampling rate (10 Hz or more), the tracking problem is simplified to recovering the small motion between successive frames. Given a kinematic model in a known starting configuration, our local tracking algorithm processes a sequence of intensity images of an unmarked hand to obtain an estimated state trajectory.

First Occludes Second | **First and Second Disjoint** | **Second Occludes First**



(a) | **(b)** | **(c)**

Figure 1: Three snapshots from a motion sequence, illustrating the different occlusion relations between the first and second fingers of the hand.

Self-occlusion is an ubiquitous property of articulated object motion, which complicates tracking. For example, consider the occlusion between the first and second fingers of the hand, depicted in Figs. 1 (a), (b), and (c). These examples illustrate the three possible scenarios for template-based finger tracking. In (b), the two finger templates can be registered to the image independently, as they are both fully visible. In (a) and (c) the templates interact as a result of self-occlusion, and one of them is visible while the other is partially obscured. Self-occlusion adds a combinatorial aspect to tracking— the visibility of different parts of the model must be estimated in addition to the registration of the model with the image. For many interesting mechanisms like the hand, however, the role each body plays in occlusion is fixed over a large region of the state space. In (a), for example, the forefinger is in front of the middle finger under all articulated motions, which is a six DOF space. We present a framework for local tracking that exploits this property in two stages. First, a *visibility ordering* of templates is determined from the kinematic model and updated over time. Second, partially occluded templates are registered using *window functions* determined by the ordering.

Using this framework, we obtain a direct, energy-based formulation of articulated tracking. The coupling of image segmentation and state estimation problems in our approach leads to increased robustness through the direct application of the kinematic constraints. We analyze the existence of our representation for general systems of self-occluding objects, and specialize it to the case of the hand. We also present experimental results for a real image sequence. These results constitute the first demonstration of 3D hand tracking in the presence of nontrivial occlusions using natural images. This report extends the *DigitEyes* system for hand tracking described in [22, 21] to the case of self-occluding motions.

2 A Framework for Tracking Self-Occluding Objects

Self-occlusion occurs when an *occluding* link blocks the camera's view of an *occluded* link. Figure 1 illustrates the three types of occlusion relations between two bodies. In this example, each finger is treated as a single rigid link and the hand rotates around the axis of the middle finger. Note that the occlusion relationship in each of the sample frames is valid for large rotations (nearly ± 90 degrees) around the given pose. The image acquisition rate and maximum hand velocity determine the maximum possible hand rotation between frames. This will be much less than 90 degrees, implying that occlusion relations determined from the estimated configuration in one frame can be applied to interpret successive frames in the sequence. Moreover, the two most important configurations for tracking, where one finger occludes the other, are separated (in the state space) by a large region in which they don't interact at all (the disjoint case.) This illustrates the main idea behind our tracking approach: the occlusion relations for an object between two images in the sequence can be inferred from the model, removing the determination of occlusion from the estimation problem.

In the two cases where fingers occlude each other, the effect on the image can be modeled by assigning each finger to a layer, and ordering the layered templates based on their visibility to the camera. Layered representations based on distance from the camera have been employed in image coding [29]. In the context of articulated object tracking, distance from the camera is not by itself a satisfactory ordering criteria, as two occluding links may occupy the same depth range. In this section, we describe a layered template model for arbitrary systems of moving objects, and its application to local tracking.

2.1 Representing Self-Occlusion with Layered Templates

The major components of our representation are illustrated in Fig. 2, for the *forward* problem of synthesizing an image of a self-occluded hand in a known configuration with respect to a calibrated camera. The hand configuration is represented by a state vector, $\mathbf{q}$, which contains the pose of the palm and joint angles of the fingers and thumb. A kinematic model gives the spatial position of each finger link (phalange) as a function of $\mathbf{q}$. Beginning with the 3D kinematic model and working in left to right, there are four components involved in synthesizing a hand image:

1. A set of *templates* that characterize the appearance of each link from different viewing directions.

2. A *deformation function* parameterized by the kinematic DOFs (state) that maps each template into the image, based on the hand configuration and camera model.

3. A *visibility order* for the transformed templates that specifies their visibility to the camera.

4. A set of *window functions*, determined by the visibility order, that select only the visible portions of each template in composing the output image.

The templates and their deformations model the appearance of the hand under different viewing angles, while the visibility order and window functions comprise a layered representation of self-occlusion. Hand tracking consists of inverting this forward model by searching locally in the state space for the configuration of the hand that best explains each image in the motion sequence.

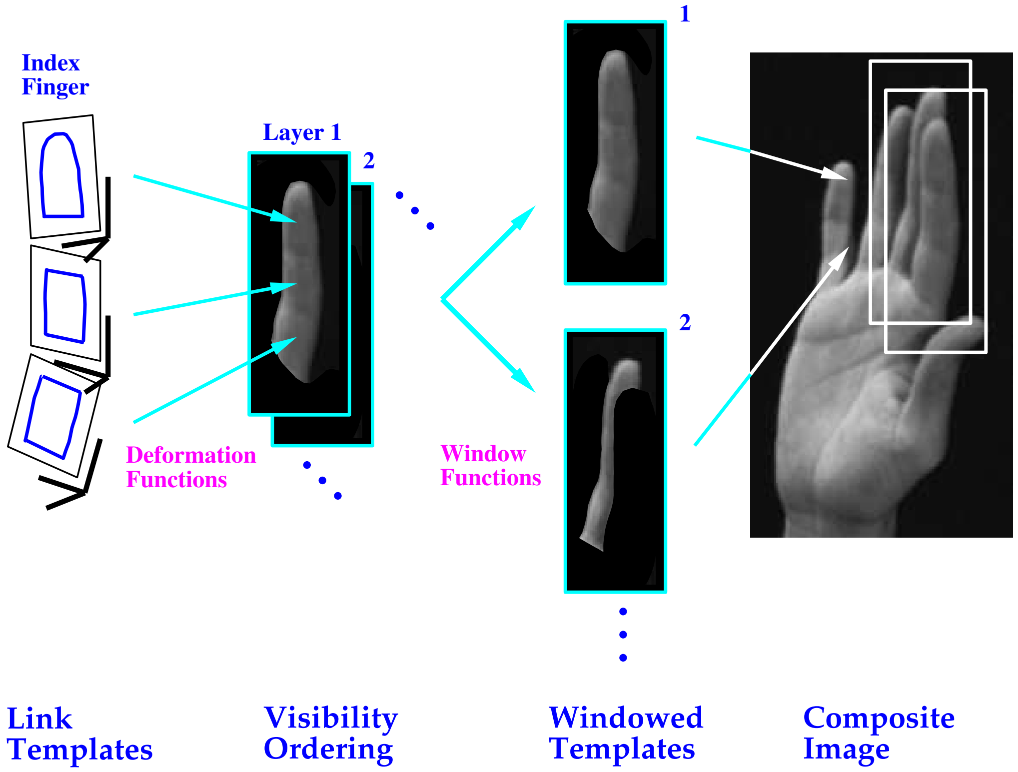


Figure 2: Synthesis of an image of a self-occluding hand by layered templates. In the four stage process, link templates are projected into the image plane, ordered by their visibility to the camera, clipped by window functions, and overlaid to produce the final image.

2.2 Local Visibility Orders for Templates

A key step in the image synthesis problem in Fig. 2 is the ordering of link templates by their visibility. This *visibility order* has the property that each body in the list will not be occluded by any of the bodies that follow it. Moreover, the ordering is preserved under bounded motion of the bodies, and can be therefore be used to estimate the motion between two frames of an image sequence. The simplest type of visibility order is a binary occlusion relation between two bodies.

When the image plane projections of two objects overlap, and the visibility of one of them (object A) is completely unaffected by the other (object B), we call it a *binary occlusion* and say that A occludes B. If two solid objects have convex shapes, then any occlusion between them will be binary.[1]

Consider a pair of convex objects undergoing bounded motion, such as would occur between two frames in an image sequence. If the image plane projections of these bodies do not overlap under the allowed motion, no occlusion is possible. In this case we say that the bodies A and B are *disjoint*, which we write $A \equiv B$. We define the binary occlusion relation, *will-occlude(A,B)*, to be true if A and B are not disjoint and A occludes B whenever their image plane projections overlap. We write this as $A \gg B$. For example, if two occluding objects are located at distances Z_A and Z_B with respect to the camera, such that $Z_A < Z_B$ over some range of motion, then $A \gg B$. The *will-occlude* relation describes a property of all possible occlusions of the two bodies under limited motion. For most articulated objects (see Sec. 4,) one of $A \equiv B$, $A \gg B$, and $B \gg A$ will be true for each pair of bodies in all configurations. Because these relations are fixed over a motion interval, they define a local *occlusion invariant*. Since the occlusion relations are defined for objects with arbitrary degrees of freedom, they generalize the concept of depth sorting in constructing a layered representation.

For a specific object of interest, like the hand, binary occlusion relations can be defined in terms of the object kinematics. This leads in Sec. 3 to a visibility ordering algorithm derived from kinematic analysis. More generally, visibility orders can be built up from the set of pairwise occlusion relations for all bodies making up a given object. Existence conditions for visibility orders and their determination for arbitrary collections of rigid bodies is taken up in Sec. 4.

2.3 Window Functions for Template Registration

Given a visibility ordered set of templates, the effect of self-occlusion on the image can be modeled by *window functions*, whose position in the image is a continuous function of the

[1] Any two convex bodies can be separated by a plane which divides the viewing sphere in half, and for all view points in each half, the object it contains is completely visible.

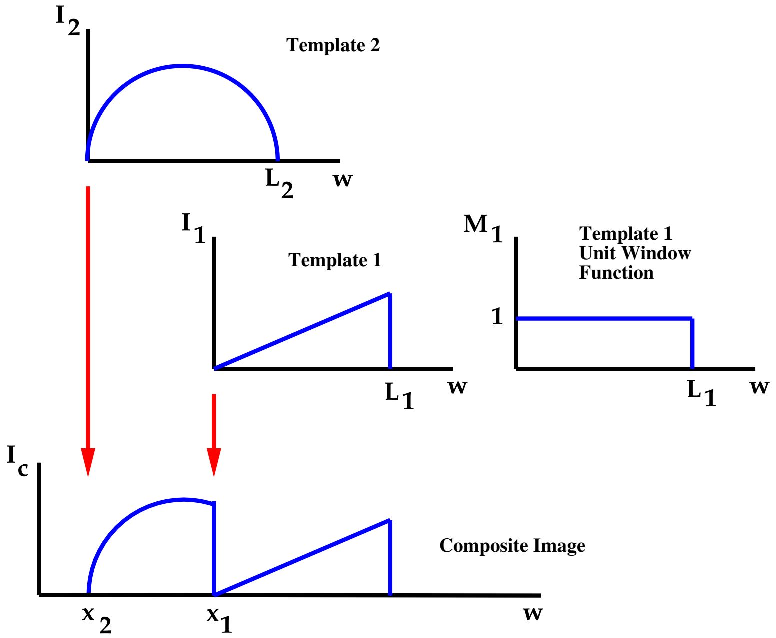


Figure 3: Image composition example for two 1D templates. Occlusion is modeled by the unit window function shown on the right.

state. Each template has an associated window, which masks out contributions to the image from templates below it in the visibility order. An example of a simple unit window function is given in Fig. 3 for the case of tracking two 1D occluding patterns with the visibility order, *Template 1* $\gg$ *Template 2*. In the forward model for this example, the two templates are combined to give a composite image:

$$I_c(x) = M_1(x - x_1)I_1(x - x_1) + [1 - M_1(x - x_1)]I_2(x - x_2) \tag{1}$$

where $I_{1,2}(\cdot)$ are the templates and M_1 is the window function for template 1. Given $m(\cdot, L)$, a unit window of length L for 1D images, we have $M_1(\cdot) = m(\cdot, L_1)$.

I_c is a forward model of the image as a function of the state (in this case, x_1 and x_2.) To formulate a tracking problem, the forward model is compared to the measured image, I, resulting in a an error function

$$E(x_1, x_2) = \frac{1}{2}\int_0^L [I_c(x_1, x_2, u) - I(u)]^2\, du \ , \tag{2}$$

which is minimized over time.

A set of templates in visibility order results in a recursive hierarchy of window functions, called the *window tree*. This tree is illustrated in Fig. 4 (also see [1], Fig. 2.) The path from

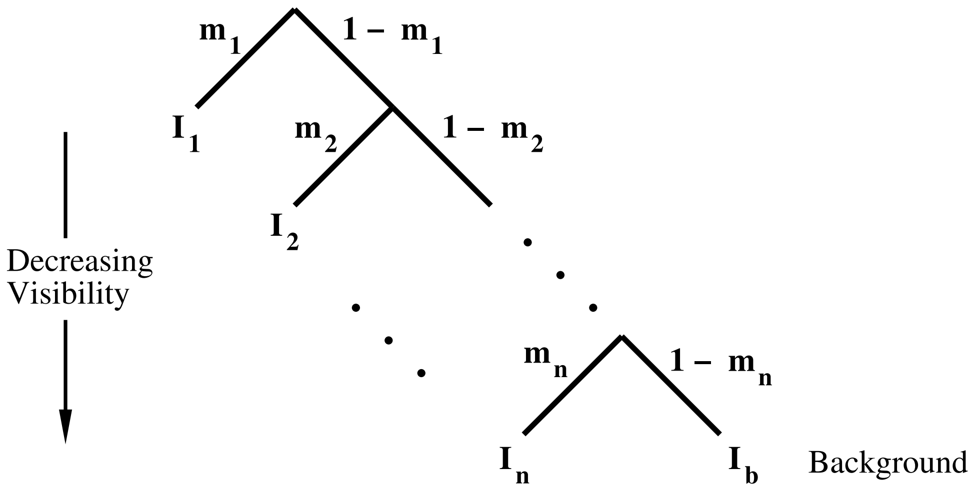


Figure 4: Tree of window functions generated by a set of templates, $I_1, I_2, \ldots, I_n$, in visibility order. I_b is the background template.

the root of the tree to a leaf template captures the composition of masks that determine the template's visibility. Incorporation of the window tree into the tracking algorithm is discussed in Sec. 5. Adding a background template, I_b, to Eqn. 1 and substituting into Eqn. 2 yields an error function with two windows,

$$\begin{aligned} E(x_1, x_2) \;=\; & \frac{1}{2}\int_0^L [M_1(x-x_1)I_1(x-x_1) - [1-M_1(x-x_1)]\{M_2(x-x_2)I_2(x-x_2) + \\ & [1-M_2(x-x_2)]I_b(u)\} - I(u)]^2\, du \end{aligned} \tag{3}$$

A simple gradient descent tracking algorithm consists of differentiating Eqn. 3 by the state and taking steps in the negative gradient direction. Without the window functions determined from template ordering, the minimization problem would include a combinatorial search to assign image pixels to templates, increasing the complexity of the estimation problem. Minimizing Eqn. 3 generates a segmentation of the image, which assigns each pixel to one of three templates.

3 Modeling Articulated Objects with Layers

The previous section outlined an approach to tracking self-occluding objects based on visibility orders and window functions. The on-line determination of visibility orders is an important part of the tracking algorithm, which can be greatly simplified for an object like

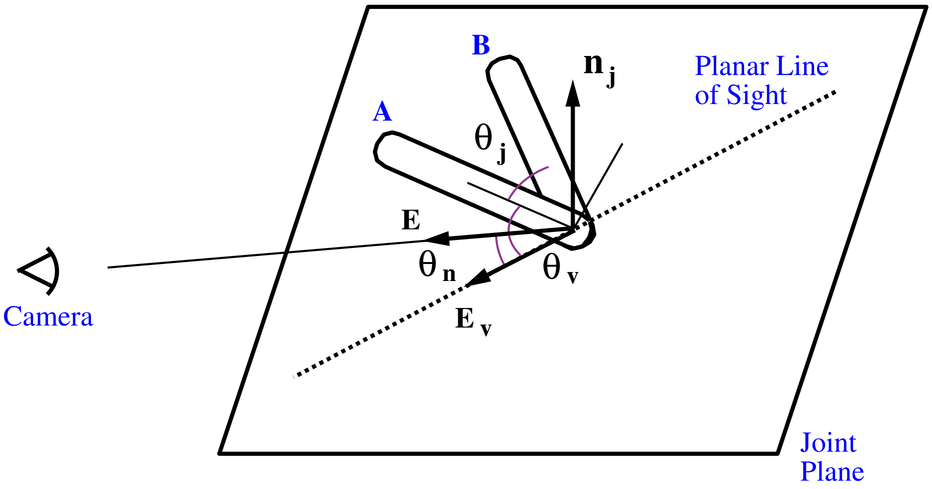


Figure 5: Occlusion properties of two links connected by a revolute joint.

the hand through kinematic analysis. The first step is to define the will-occlude relation for two links connected by a revolute joint, a basic component of articulated kinematic chains. Using this definition and the kinematics of the hand, we derive an algorithm for visibility ordering hand templates. Given the visibility order, determining the window functions is straightforward, and can be done in conjunction with the residual computation, as described in Sec. 5.

3.1 Occlusion Relations for Revolute Joints

A revolute joint constrains two links to a single degree of freedom of relative motion. The first step in defining occlusion relations for the revolute case is to identify the fixed and moving bodies. There is a natural ordering between any two links in a kinematic chain. The link closest to the base of the chain is chosen as the fixed link, A. The joint angle, θ_j, positions link B relative to A's coordinate frame at the joint center, illustrated in Fig. 5. Both links lie in a *joint plane*, whose normal is defined by the joint axis, $\mathbf{n}_j$.

Since connected links overlap at their joints, there is always some degree of occlusion between adjacent links. However, for links like finger phalanges that are convex and roughly symmetric, we are only interested in the visible surface along the axis of symmetry. Occlusion in this region is determined by the camera position relative to the joint center. In general, the camera viewpoint will lie outside the joint plane, as illustrated in Fig. 5. Consider, for the moment, a camera viewpoint located *in* the plane, along the planar viewing vector E_v from the joint center. Occlusion occurs whenever both links lie on the same side of the planar line of sight containing E_v. The following table summarizes the conditions for occlusion as a function of θ_j and the viewing angle θ_v, between E_v and A:

$$\begin{array}{lll} \theta_v > 0: & \theta_j \in [0, \theta_v] & \Rightarrow B \text{ occludes } A \\ & \theta_j \in [\theta_v - \pi, 0] & \Rightarrow A \text{ occludes } B \\ \theta_v < 0: & \theta_j \in [\theta_v, 0] & \Rightarrow B \text{ occludes } A \\ & \theta_j \in [0, \theta_v + \pi] & \Rightarrow A \text{ occludes } B \end{array} \quad (4)$$

In Fig. 5, $\theta_v > 0$ and $\theta_v - \pi < \theta_j < 0$, so that A is occluding B. Occlusion properties change at the boundaries of the intervals. Note that θ_j is bounded away from zero on both sides by noninterpenetration.

As the viewpoint moves out of the joint plane, the amount of occluded surface area decreases. When the general viewing vector, E, is parallel to $\mathbf{n}_j$ there is essentially no occlusion for all joint angles. E makes an angle θ_n with the joint plane, in which it has the projection E_v. It follows that any viewing direction can be represented in the joint coordinate frame by two angles: θ_v and θ_n. The occlusion conditions from Eqn. 4 apply only to viewpoints for which $|\theta_n| < \Delta_n$, for some fixed threshold Δ_n. For viewpoints above this threshold, the links are disjoint.

Given the state of an articulated object, Eqn. 4 can be applied to determine the occlusion at a revolute joint. To use this model for tracking, it must be extended to include bounded motions of the two links. Bounded change in the DOFs before link A in the kinematic chain will displace the joint coordinate frame, causing θ_n and θ_v to vary. The exact change in these angles will be a complex function of the state, but it can be approximated by restricting them to intervals, I_n and I_v, of a fixed size, centered around their current value. Bounded motion between B and A is modeled by an interval $I_j = [\theta_j^0 - \Delta_j, \theta_j^0 + \Delta_j]$, of width Δ_j containing θ_j. The intervals I_n and I_v are defined similarly. These intervals can be incorporated into Eqn. 4 by replacing inequalities with intersection tests. The normal, viewing, and joint angles at the current state are θ_n^0, θ_v^0, and θ_j^0, respectively. The revolute occlusion relations are

$$\begin{array}{llll} I_n \cap [-\Delta_n, \Delta_n] \neq \emptyset : & \theta_v^0 > 0 : & I_j \cap [0, \theta_v^0 + \Delta_v] \neq \emptyset & \Rightarrow\ B \gg A \\ & & I_j \cap [\theta_v^0 - \Delta_v - \pi, 0] \neq \emptyset & \Rightarrow\ A \gg B \\ & \theta_v^0 < 0 : & I_j \cap [\theta_v^0 - \Delta_v, 0] \neq \emptyset & \Rightarrow\ B \gg A \\ & & I_j \cap [0, \theta_v^0 + \Delta_v + \pi] \neq \emptyset & \Rightarrow\ A \gg B \\ \text{Otherwise} & & & \Rightarrow\ A \equiv B \end{array} \quad (5)$$

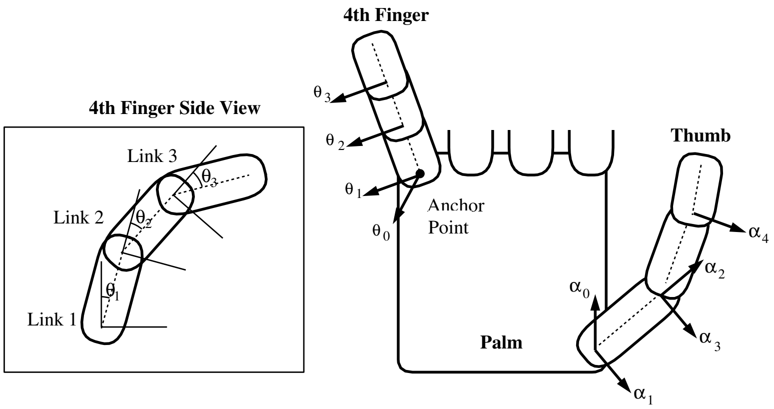


Figure 6: Kinematic models, illustrated for fourth finger and thumb. The arrows illustrate the joint axes for each link in the chain.

3.2 Kinematic Hand Modeling

The occlusion relation from Eqn. 5 can be applied to any two links in a kinematic chain that share a revolute joint. Constructing a visibility ordering for the hand from this relation requires a detailed kinematic model. We represent object kinematics using the Denavit-Hartenberg notation, which is widely used in robotics [26]. Each link in the object has a coordinate frame whose 3D position is given by a series of matrix multiplications, parameterized by the state. For the hand, the state vector encodes the pose of the palm (seven states for quaternion rotation and translation) and the joint angles of the fingers (four states per finger, five for the thumb). The hand kinematic model we employ in our experiments is illustrated in Fig. 6. The total hand configuration is described by a 28 dimensional state vector.

We model the hand as a collection of 16 rigid bodies: 3 individual finger links (called phalanges) for each of the five digits, and a palm. From a kinematic viewpoint, the hand consists of multi-branched kinematic chains attached to a six DOF base. We make several simplifying assumptions in modeling the hand kinematics. First, we assume that each of the four fingers of the hand are planar mechanisms with four DOFs. The abduction DOF moves the plane of the finger relative to the palm, while the remaining 3 DOF determine the finger's configuration within the plane. Each finger has an *anchor point*, which is the position of its base joint in the coordinate frame of the palm, which is assumed to be rigid. Real fingers deviate from our modeling assumptions slightly, but we have found them to be adequate in practice. We employed this kinematic model in our earlier real-time hand tracking system called *DigitEyes* [22, 21]. The kinematic representation and its implementation are completely general, and different objects can be tracked simply by changing a parameter file. More details are available in [20].

3.3 Visibility Orders for Hand Templates

The kinematic properties of objects like the hand can be exploited in an algorithm for visibility ordering link templates. In this approach, templates are ordered within each finger chain using the revolute occlusion relation described above. Then the chains are compared as distinct objects, avoiding the complexity of testing each link against all the others. By exploiting the kinematic structure, the algorithm is efficient enough for on-line implementation. A more general approach to computing visibility orders from binary occlusion relations is described in Sec. 4.[2]

The hand consists of five planar kinematic finger chains and a rigid palm. As a result of planarity, the three joint axes in each finger are parallel and have the same joint plane. This greatly simplifies the application of revolute occlusion relations to finger ordering. A further simplification comes from the fact that all joint angles must be positive, reflecting physical limits on joint motion. As a result, each finger can be viewed as a convex planar shape. These two observations lead to a simple procedure for ordering templates within each link.

If the angle, θ_n, between the camera and the finger joint plane exceeds the threshold, Δ_n, described in Sec. 3.1, then the finger templates are disjoint and can be ordered arbitrarily.

[2] Note, however, that the revolute occlusion relation defined above applies only to pairs of links that share a joint. This definition would have to be extended to an arbitrary pair of links to meet the requirements of the general approach.

Otherwise, two applications of Eqn. 5 determine the ordering between links 1 and 2, and links 2 and 3 (see Fig. 6 for the link numbers, which are the same for each finger.) Convexity imposes strong constraints on the global pose of the finger, making it possible to generate the entire visibility order directly from the two pairwise tests, according to the following table:

$$\begin{array}{rcl} 1 \gg 2 \ \text{ or } \ 2 \gg 3 & \Rightarrow & 1 \gg 3 \\ 2 \gg 1 \ \text{ or } \ 3 \gg 2 & \Rightarrow & 3 \gg 1 \\ 1 \equiv 2 \ \text{ and } \ 2 \equiv 3 & \Rightarrow & 1 \equiv 3 \end{array} \tag{6}$$

The thumb is also a planar mechanism with joint limits, and the finger template ordering rules can be applied to it without modification.

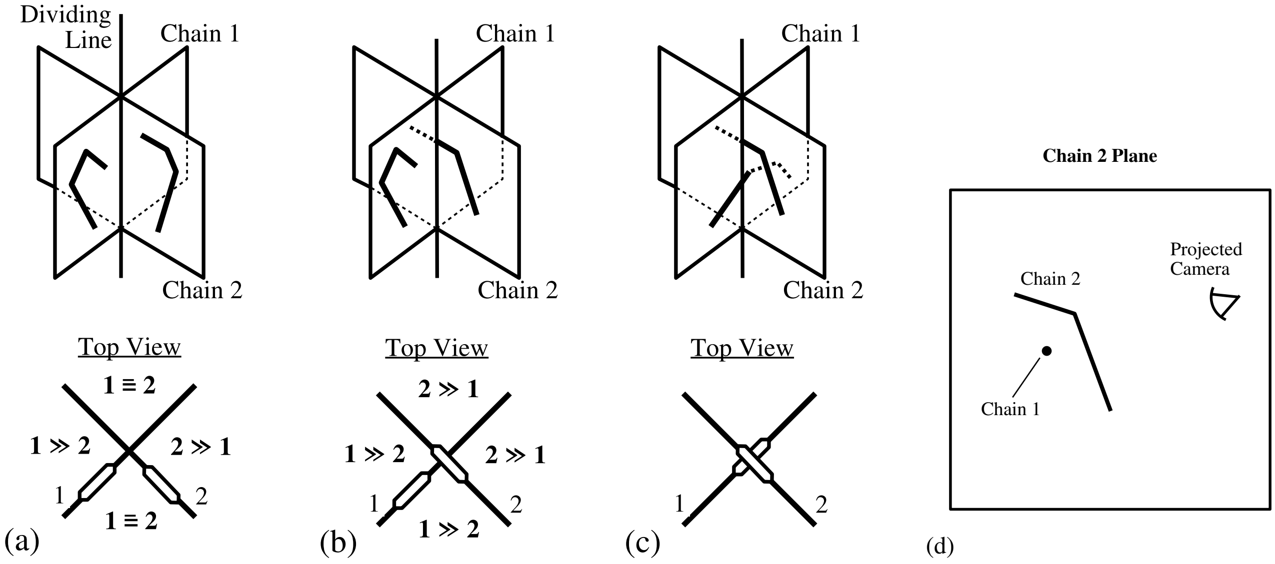


Figure 7: Types of intersections between two planar kinematic chains. In (a), chains are confined to separate sides of the dividing line at which their planes intersect. In (b) one chain crosses the line, and in (c) they both do. The viewpoint relative to the dividing line determines the visibility order. (d) shows the ordering test from (c) in the chain 2 plane.

Occlusions between fingers are almost always binary. This observation simplifies visibility ordering by removing the need to consider individual templates. When the planes for two fingers are parallel, they can be ordered by distance from the camera. When the planes intersect, there are three possibilities, illustrated in Fig. 7 (a), (b), and (c). In (a), neither chain crosses the dividing line formed by the plane intersection. In this case, the two planes divide 3D space into four quadrants, with associated visibility orders given in the figure. In (b), one chain crosses the dividing line, but the other does not. In this case the quadrant labels are different. Note that the transition from (a) to (b) either leaves the visibility order unchanged, or changes a disjoint situation to an ordered one.

The only case where nontrivial interaction between the chains occurs is (c), where they both cross the dividing line. This case requires additional analysis within the plane of the finger. The first step is to choose the plane closest to the camera, in which the occlusion effect is most visible, and project the camera viewpoint into that plane. Due to convexity, the other chain will intersect this plane at one point. Figure 7 (d) shows a sample configuration of links in this case. If the line in the plane joining the projected viewpoint and the intersection point passes through the chain, then the chain comes first in the visibility order. Otherwise, the intersecting chain comes first.

In order for the plane intersection test to be valid under bounded motion, it is necessary to model the effect of chain motion within the plane and motion of the plane itself on the outcome. If the test is applied between fingers and thumb on the same hand, then palm motion will not effect the type of intersection, but may change the camera's quadrant. Since everything hinges on whether each chain crosses the dividing line, this can be modeled by bounding the distance to the line for the closest part of each chain. The only nontrivial transition is from case (b) to (c). In this situation, a finger or thumb tip intersects the other chain's plane for the first time. The point of intersection can be predicted from the motion, or bounded by intersecting the bound on tip displacement with the plane.

Finger planes will intersect each other due to abduction. However, these planes are roughly parallel, and the intersections will almost always be of type (a) in Fig. 7. As a result, there is a simple visibility ordering algorithm for the fingers: sort the anchor points for each finger based on distance to the camera along the optical axis. This determines the finger ordering. The thumb plane can intersect the finger planes in a variety of ways depending on the motion, and the intersection tests described above must be applied in this case. In most situations, the outcome of the test between the thumb and first finger can be applied to the rest of the fingers as well.

Finally, the plane of the palm sweeps out a volume in space in the direction of the camera axis. If the tip of a finger or the thumb intersects this volume, then the palm comes before that chain in the visibility order, otherwise after. A visibility order for hand templates can be constructed from the tests outlined above. These tests should be very simple to implement, making it possible to update the ordering on-line whenever a new state estimate is available. Note that the fundamental assumption in the above analysis is the planarity of the kinematic chains comprising the object. This modeling assumption is also valid for arms and legs, suggesting that the ordering tests described above could also be applied to human figures.

4 Properties of Visibility Orders

The existence of a visibility ordering algorithm for the hand raises the question of what other objects can be treated under the same framework. In this section we develop general existence conditions for visibility orders. These results apply to a multibody system with arbitrary degrees of freedom.

4.1 Existence of Local Occlusion Invariants

A multi-body system has a local occlusion invariant if, for a given bounded motion, one of $A \equiv B$, $A \gg B$, $B \gg A$ is true for each pair of bodies, A and B. A visibility order can be constructed in this case, as we show in the next section. We model bounded relative motion between the two bodies in A's coordinate frame, letting $M(B)$ denote the union of all possible spatial positions of B.[3] In general, $M(B)$ will not be convex. But its convex hull, $CH[M(B)]$, can be partitioned from A by a separating plane if the occlusion is *unambiguous*. This is illustrated in Fig. 8 (a) for two 2D bodies viewed by a 1D camera. The relative motion in this case is rotation of B. The partition creates two half-spaces. If the image plane projections of A and the *motion image* of B, $CH[M(B)]$, don't overlap, $A \equiv B$. If they do overlap, the object in the half-space containing the camera will occlude the other object. In figure (a), $B \gg A$.

The case of occlusion ambiguity is illustrated in Fig. 8 (b), using the same two bodies.

[3]The spatial position of each body is defined with respect to the world coordinate frame. For convenience, we shift the reference frame to A.

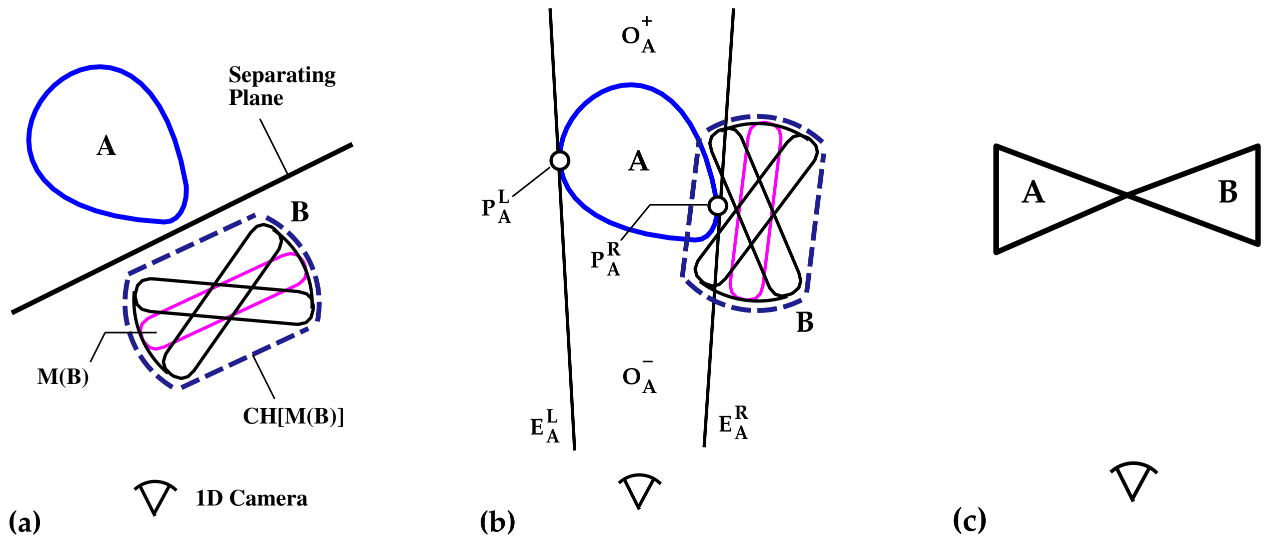


Figure 8: Occlusion relations for 2D objects viewed by a 1D camera. (a) Sufficient conditions for $A \gg B$, (b) geometric definition of occlusion ambiguity, and (c) degenerate configuration of two planar objects in point contact. No nonzero bound on relative translation can remove the occlusion ambiguity.

For this configuration, it is impossible to predict the occluder under the given motion bound. Ambiguity arises when $CH[M(B)]$ intersects the occluding limb of A. Referring to the figure, let $E_A^{L,R}$ denote the pair of line-of-sight tangents to A, with E_A^R closest to B. The points of contact, $P_A^{L,R}$, are the occluding limbs (in 3D this is a curve in the surface of A). The pair of tangents bound a region of space, O_A (a tangent cone in 3D,) which contains A and the camera viewpoint. O_A is divided into occluding and occluded regions, labeled O_A^+ and O_A^-. Occlusion ambiguity arises when $M(B)$ has a nonzero intersection with both regions. In this case, $CH[M(B)]$ intersects A and contains P_A^R, and both binary occlusion outcomes are possible. In general, the likelihood of an occlusion ambiguity decreases with the motion bound, but it can't be eliminated altogether, as figure (c) demonstrates.

When they exist, the set of ambiguous configurations will occupy a small subspace of the total configuration space, as they depend on a special combination of spatial proximity and viewing angle. An example of an ambiguous hand configuration is the "stop" gesture, with the hand held flat, fingers pressed together, and palm facing the camera. In this pose, rotation around the vertical axis changes the visibility order of the fingers. In a specific case like the hand, knowledge about ambiguous configurations can be used to aid tracking. Simple velocity-based prediction, for example, could be used to correctly interpret ambiguous cases. In general, high frame rates reduce the danger of an incorrect occlusion hypothesis, by making the mislabeled region of pixels as small as possible.

4.2 Visibility Ordering and Occlusion Graphs

The occlusion relations for a multi-body system can be represented by a directed *occlusion graph*. The graph is a pair (V, E), where the vertex set V contains all of the bodies. To construct the edge set, E, consider all pairs $x, y \in V$. Since there are no occlusion ambiguities, one of $x \equiv y$, $x \gg y$, or $y \gg x$ must be true. In the first case no edge is added, while the other two cases add directed edges (x, y) and (y, x) respectively. Consider the collection of 2D rigid bodies viewed by a 1D camera illustrated in Fig. 9. Figure 10 (a) shows the occlusion graph for the system under bounded translations in the plane.

When the object configuration admits a visibility ordering, it can be obtained by searching the occlusion graph. A configuration that can't be so ordered is illustrated in Fig. 11. In general, the occlusion graph must be *acyclic* to induce a natural order on the set of objects. The presence of occlusion cycles is fairly unusual, at least for convex bodies, as it involves a special arrangement of spacing and orientation. Cycles don't occur naturally in hand or body configurations, for example.

When the occlusion graph is acyclic, it can be topologically sorted by depth-first search [4] to produce a *visibility ordering*. Figure 10(b) shows the ordering produced by the sample occlusion graph. The sorted graph has the property that all edges are directed left to right. Taking the vertices in that order guarantees that no object will be occluded by an object that follows it in the list.

These results give sufficient conditions for the existence of a visibility ordering for an arbitrary object. Existence hinges primarily on the absence of occlusion ambiguities, which is determined by the relative motion and the temporal sampling rate. These results are useful in identifying the most likely configurations for occlusion ambiguities in a known object. They also suggest an algorithm for visibility ordering an arbitrary object, without

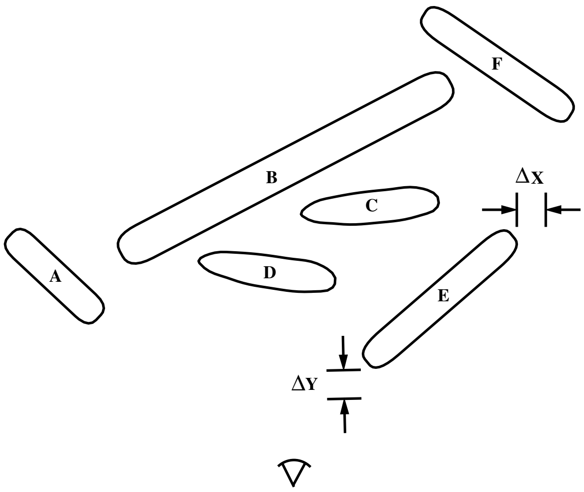


Figure 9: A collection of 2D rigid bodies under bounded translational motion relative to a 1D camera. Each body can translate by ΔX and ΔY, as shown for body E.

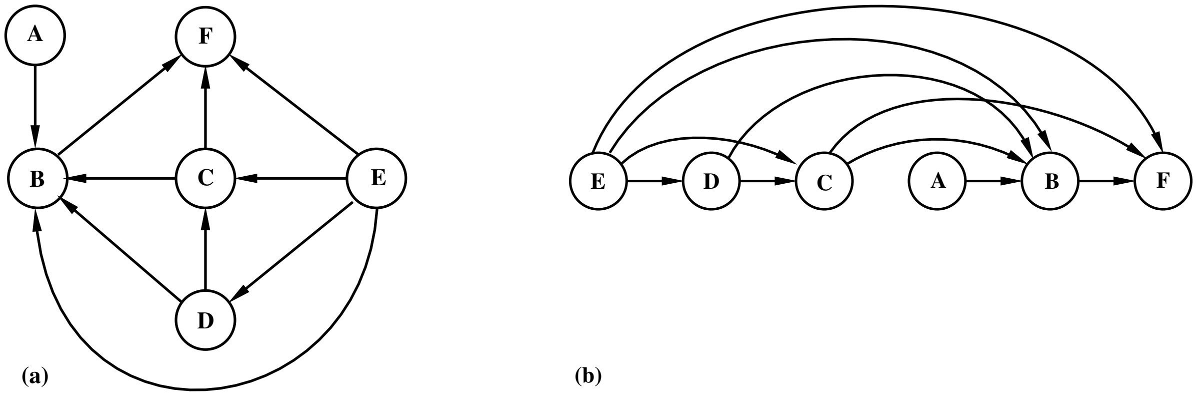


Figure 10: (a) Occlusion graph for the mechanism in Fig. 9, and (b) the visibility order produced by sorting the graph.

Figure 11: (a) A configuration of three objects and (b) its associated cyclic occlusion graph.

special case kinematic analysis, which is presented in Appendix A.

Looking beyond model-based tracking, there is increasing interest in layered representations for computer vision, because of their potential to simplify the 3D description of the world. Recently, several algorithms have been proposed for building layered descriptions of a scene from a single image or a motion sequence [17, 5, 29]. The results in this report provide general conditions under which a layered representation could be expected to exist, for a given type of moving object.

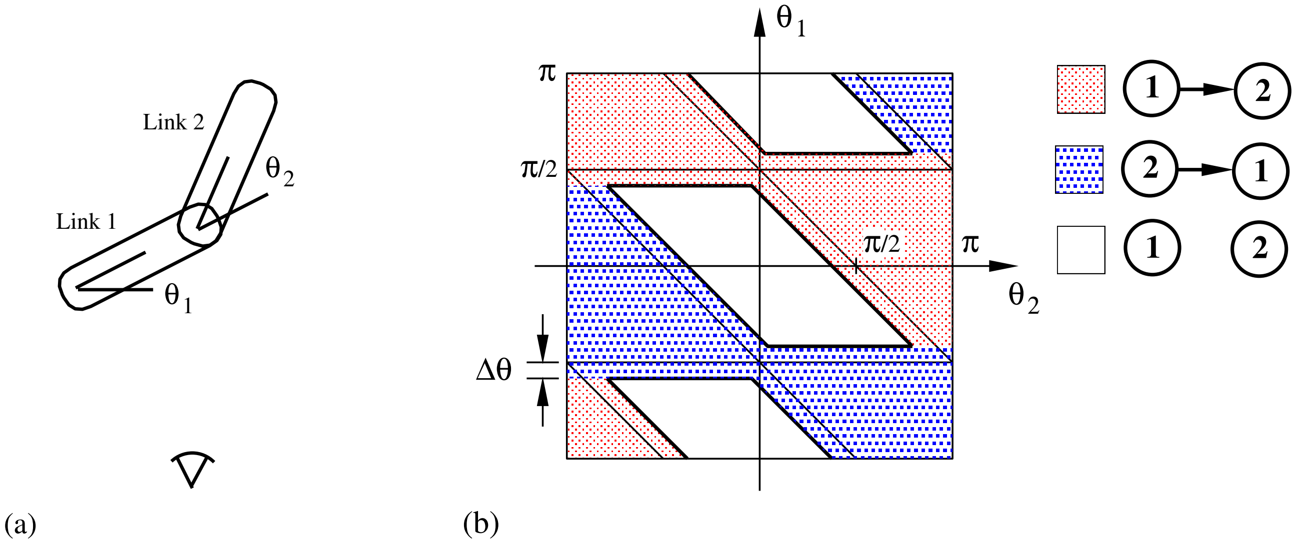


Figure 12: (a) Two link mechanism in 2D, and (b) associated occlusion meta-graph.

4.3 Occlusion Events and Global Models

The occlusion graph for a mechanism is a function of its state. A discrete change in the topology of the graph can be viewed as an *occlusion event*, analogous to the visual events introduced by Koenderink and Van Doorn [15]. These events partition the configuration (state) space into hypervolumes over which the occlusion graph is constant. We call this partition the *occlusion meta-graph* for the mechanism.

A sample occlusion meta-graph for a two link planar mechanism is given in Fig. 12. The 2 DOF state space is partitioned into three types of regions for which the occlusion graph is constant. The values of θ_1 can wrap around from $-\pi$ to π, but θ_2 is bounded away from both extremes, due to noninterpenetration. Each state is restricted to an interval of size

$2\Delta\theta$ between frames. Substituting the size of the interval into Eqn. 5 produced the occlusion event boundaries shown in the figure. This construction technique is analogous to obstacle growing in the configuration space approach to manipulator path planning [3]. In general, the hypervolumes will be n dimensional regions bounded by curved surfaces.

The visibility order used to register the current frame is given by the region containing the current state. Transitions between regions, which signify occlusion events, cause a change in the order. Note that an occlusion event *doesn't* imply that an occlusion has taken place, but rather that the roles of the bodies involved in occlusion have changed. The most interesting property of this example is that trajectories connecting the two occuding regions, where $1 \gg 2$ or $2 \gg 1$, always pass through the disjoint region where $1 \equiv 2$. These disjoint regions provide a kind of buffer zone, making it possible to adjust the ordering in advance of new occlusions between bodies. Similar observations are exploited in the hand visibility algorithm of Sec. 3.3. Moreover, occlusion ambiguities can be identified in this representation as points where a square window of size $2\Delta\theta$ intersects two occluding regions. Since the left and right edges of the graph are excluded in this case, as explained above, there are no occlusion ambiguities.

5 State Estimation

Articulated object tracking is a model-based sequential estimation problem: given a sequence of images of the hand, its kinematic model, a collection of templates, and the initial state, the estimator makes the state correction at each frame that minimizes the residual error between the image and the transformed, windowed hand templates. Since the template positions in the image are constrained directly by the state model, this is an example of a direct minimization (or "energy-based" [14, 28, 23]) approach to tracking. It stands in contrast to approaches, like our earlier *DigitEyes* formulation, in which residual computation follows a separate feature extraction stage. By enforcing the kinematic constraints at the earliest stage of image analysis, we gain robustness to both noise and the effects of self-occlusion.

5.1 Hand Template Models

The layered model for self-occlusion presented in the last section employs a template-based description of hand appearance. The template model has two parts, an image with an associated template coordinate frame, and a *boundary contour* in template coordinates that inscribes the template pixels. The appearance of each link is modeled by a collection of templates, each associated with a particular viewing direction. The number of required views is a function of the shape (and possibly photometry) of the link. For roughly cylindrical links, like finger phalanges, a single view may be enough, while an object like the palm or body torso will require more. Given the estimated pose of the link relative to the camera, the correct template for local tracking can be selected automatically. The experimental results in this report use a single view for each link.

As illustrated in Fig. 2, each template for a given link can be viewed as being painted on a plane which is fixed in the link coordinate frame. Given the state of the hand, the spatial position with respect to the camera of the link frame, and therefore the template plane, is specified by the kinematic model. The effects of rotation and foreshortening in the image are captured by projecting each template through the camera model. We currently use an affine approximation to the true perspective mapping at each link. This combination of kinematic and camera transforms is captured in a *deformation function* [23], $\mathbf{f}(\mathbf{q}, \mathbf{s})$, which maps template coordinates, $\mathbf{s} = [u \ v]$, to image coordinates, as a function of the hand state $\mathbf{q}$ and calibrated camera model. This mapping is illustrated in Fig. 13 for a finger tip template. The deformation function has the general form $\mathbf{f}(\mathbf{q}, \mathbf{s}) = \mathbf{CT}(\mathbf{q})\mathbf{s}$, where $\mathbf{T}$ transforms a point in template coordinates to a point in the template plane, and C maps the template plane into the image.

To illustrate the use of the deformation function, consider the 2D version of Eqn. 1, forming a composite image from two templates, I_1 and I_2:

$$I_c(\mathbf{q}, \mathbf{w}) = M_1(\mathbf{q}, \mathbf{w}) I_1(\mathbf{f}_1^{-1}(\mathbf{q}, \mathbf{w})) + [1 - M_1(\mathbf{q}, \mathbf{w})] I_2(\mathbf{f}_2^{-1}(\mathbf{q}, \mathbf{w})) \ , \tag{7}$$

where $\mathbf{f}_{1,2}^{-1}$ are inverse deformation functions for the two templates that map from image coordinates, $\mathbf{w} = [x \ y]$, to template coordinates as a function of the state. Since the

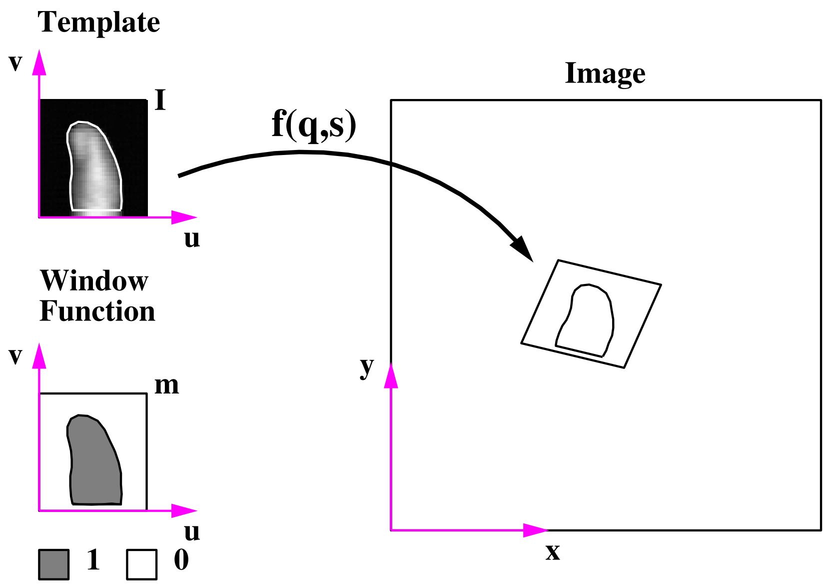


Figure 13: A template and its associated unit window function are illustrated for the finger tip. The boundary contour, drawn in white, encloses the template pixels. The deformation function maps the template into the image.

functions $\mathbf{f}_{1,2}$ are affine in the image coordinates, their inverses are well-defined. $M_1(\mathbf{q}, \mathbf{w})$ denotes the window function for template 1, positioned in the image. It is defined for a general template, I_j, as

$$M_j(\mathbf{q}, \mathbf{w}) = m_j(\mathbf{f}_j^{-1}(\mathbf{q}, \mathbf{w})) \ , \tag{8}$$

where $m_j(\mathbf{s})$ is a 2D unit window in template coordinates, that is equal to one inside the template's boundary contour and zero everywhere else, as illustrated in Fig. 13.

5.2 SSD Error Measure

The error function measures the difference between the input image and the image predicted by the collection of deformed hand templates. We use the standard sum of squared differences (SSD) error measure between filtered pixels. In the SSD approach, both the input image and the templates are convolved with a filter and subtracted to obtain the *residual error*,

$$R(\mathbf{q}, \mathbf{w}) = \hat{I}(\mathbf{w}) - \hat{I}_c(\mathbf{q}, \mathbf{w}) \ , \tag{9}$$

at each pixel w. $\hat{I}$ is the filtered input image and $\hat{I}_c$ the composite image formed from filtered templates. By changing the filter, different properties of the image can be emphasized. For example, the Laplacian of Gaussian (LOG) filter used in our experiments is sensitive to edge energy. A variety of cost functions can be investigated within this framework. The total error resulting from this residual choice,

$$E(\mathbf{q}) = \frac{1}{2}\int_I R(\mathbf{q},\mathbf{w})^2\,\mathbf{dw} \quad , \tag{10}$$

is a global measure, evaluated over the entire visible surface of the hand. If the background image is known, it can be incorporated to improve tracking performance.

For example, following Eqn. 3, the error function for two 2D occluding templates and a background can be written

$$\begin{aligned} E(\mathbf{q}) \;=\; & \frac{1}{2}\int_I [\hat{I}(\mathbf{w}) - M_1(\mathbf{q},\mathbf{w})\hat{I}_1(\mathbf{f}_1^{-1}(\mathbf{q},\mathbf{w})) - [1 - M_1(\mathbf{q},\mathbf{w})] \times \\ & \{M_2(\mathbf{q},\mathbf{w})\hat{I}_2(\mathbf{f}_2^{-1}(\mathbf{q},\mathbf{w})) + [1 - M_2(\mathbf{q},\mathbf{w})]\hat{I}_b(\mathbf{w})\}]^2\,\mathbf{dw} \quad , \end{aligned} \tag{11}$$

where $M_{1,2}$ are window functions for the two filtered templates, $\hat{I}_{1,2}$, and $\hat{I}_b$ is the filtered background.

5.3 Gradient-based Minimization

Given an error function such as Eqn. 11, tracking proceeds by a simple gradient descent minimization algorithm. If $E_k(\cdot)$ denotes the state-dependent error for input image I_k, the state update is given by:

$$\mathbf{q}_k = \mathbf{q}_{k-1} - \rho\frac{\partial E_k}{\partial \mathbf{q}}(\mathbf{q}_{k-1}) \tag{12}$$

where ρ is the step size. The update step can be iterated when the inter-frame motion is large. This is the simplest gradient-based minimization algorithm. We are currently investigating more sophisticated methods like Gauss-Newton.

The key to the gradient-based approach is a high image sampling rate, which limits both image and state space motion between frames. Templates will generate useful error signals only when they "see" a significant portion of the link they are tracking, making it important to limit motion in the image plane. In the state space, the gradient will generate a useful correction only when the new state is in the neighborhood of the previous estimate. The time required to compute the error gradient determines the maximum possible sampling rate.

The structure of the gradient computation can be seen by substituting Eqn. 9 into Eqn. 10 and differentiating to obtain

$$\frac{\partial E}{\partial \mathbf{q}} = \int_I R(\mathbf{q},\mathbf{w}) \frac{\partial R}{\partial \mathbf{q}}(\mathbf{q},\mathbf{w})\, \mathbf{dw} = \int_I R(\mathbf{q},\mathbf{w}) \frac{\partial I_c}{\partial \mathbf{q}}(\mathbf{q},\mathbf{w}) \ . \tag{13}$$

The *residual Jacobian*, $\partial R/\partial \mathbf{q}$, describes the change in the composite image as a function of the state. In practice, the residual and its Jacobian will be evaluated at discrete pixel locations, summed over the input image. At each pixel, the following three steps must be performed:

1. Assign the pixel to one of the hand templates or the background and determine its template coordinates, based on the current state estimate.

2. Compute the residual error between the input image pixel and its associated template pixel.

3. Compute the residual Jacobian from the composite image.

Performing step one for all pixels gives a segmentation of the input image. Two algorithms for determining this segmentation are described in Sec. 5.5. Once an image pixel has been assigned to a template, its corresponding template pixel is determined by the inverse deformation function, and the residual follows easily. The remaining step is the computation of the residual Jacobian.

5.4 Residual Jacobian Computation

Suppose an image pixel, $\mathbf{w}'$, originates from template I_j at template coordinate $\mathbf{s}_j$. Furthermore, let $\mathbf{p}'$ denote the 3D position of $\mathbf{s}_j$ in camera coordinates, as determined by the position of the template plane. The pixel at $\mathbf{w}'$ makes the following contribution to the residual

$$R' = \hat{I}(\mathbf{w}') - \hat{I}_c(\mathbf{q},\mathbf{w}') \ . \tag{14}$$

The window tree, illustrated in Fig. 4, gives the structure of $\hat{I}_c(\cdot)$. Consider Eqn. 11 as a specific example. It is formed by recursively descending a tree of depth two. If all of its terms are multiplied out, aside from the $(1 - M_i)$ terms, the resulting sum consists of the paths from the root of the window tree to its three leaves.

We consider two possible cases for the pixel $\mathbf{w}'$: either it is in the *interior* of I_j, or it is on the *boundary* of I_j and a second template I_k, where $j < k$. The Jacobian calculations in these cases rely on two assumptions: that the window functions are constant in the template interiors and fall to zero at their boundaries, and that the bodies are opaque, so that no more than two templates can effect a pixel value simultaneously.

For the interior pixel case, we divide the ordered templates into a group, $\{I_1, \ldots, I_{j-1}\}$, that occludes I_j and a group, $\{I_{j+1}, \ldots, I_n\}$ that is occluded by it, as illustrated in Fig. 14 (a). Window functions and their gradients for the occluding templates are zero at $\mathbf{w}'$, leading to the simplification

$$\begin{aligned} R' \quad = \quad & \hat{I}(\mathbf{w}') - [1 - M_1(\mathbf{q}, \mathbf{w}')][1 - M_2(\mathbf{q}, \mathbf{w}')] \cdots [1 - M_{j-1}(\mathbf{q}, \mathbf{w}')] \times \\ & M_j(\mathbf{q}, \mathbf{w}')\hat{I}_j(\mathbf{f}_j^{-1}(\mathbf{q}, \mathbf{s}')) - I_c^- \ . \end{aligned} \tag{15}$$

The second term in Eqn. 15 is produced by descending the window tree to node I_j. The gradients of its window functions are zero at $\mathbf{w}'$, so its only derivative contribution comes from $\hat{I}_j$. I_c^- contains the occluded templates' contributions to I_c. Its derivative is zero, as all of its terms contain $[1 - M_j(\mathbf{q}, \mathbf{w}')]$, which vanishs at $\mathbf{w}'$ along with its derivative.

Changing variables to template coordinates and differentiating Eqn. 15 gives the Jacobian contribution of an interior pixel:

$$\mathbf{J}_j^I(\mathbf{s}') = \frac{\partial}{\partial \mathbf{q}}[\hat{I}(\mathbf{f}_j(\mathbf{q}, \mathbf{s}')) - \hat{I}_j(\mathbf{s}')] = \frac{\partial \mathbf{f}_j}{\partial \mathbf{q}} \frac{\partial \hat{I}}{\partial \mathbf{w}} \ . \tag{16}$$

This is the usual SSD Jacobian component, that would be present if there was only a single template and no window functions. It is a product of two terms, the first of which is the spatial velocity of $\mathbf{p}'$ as a function of the state, projected into the image plane through the camera model. The second term is the image gradient. So the intensity Jacobian at an image point is a weighted combination of the kinematic Jacobian of its associated link template point. It follows that kinematic Jacobians must be computed at all interior points in the link

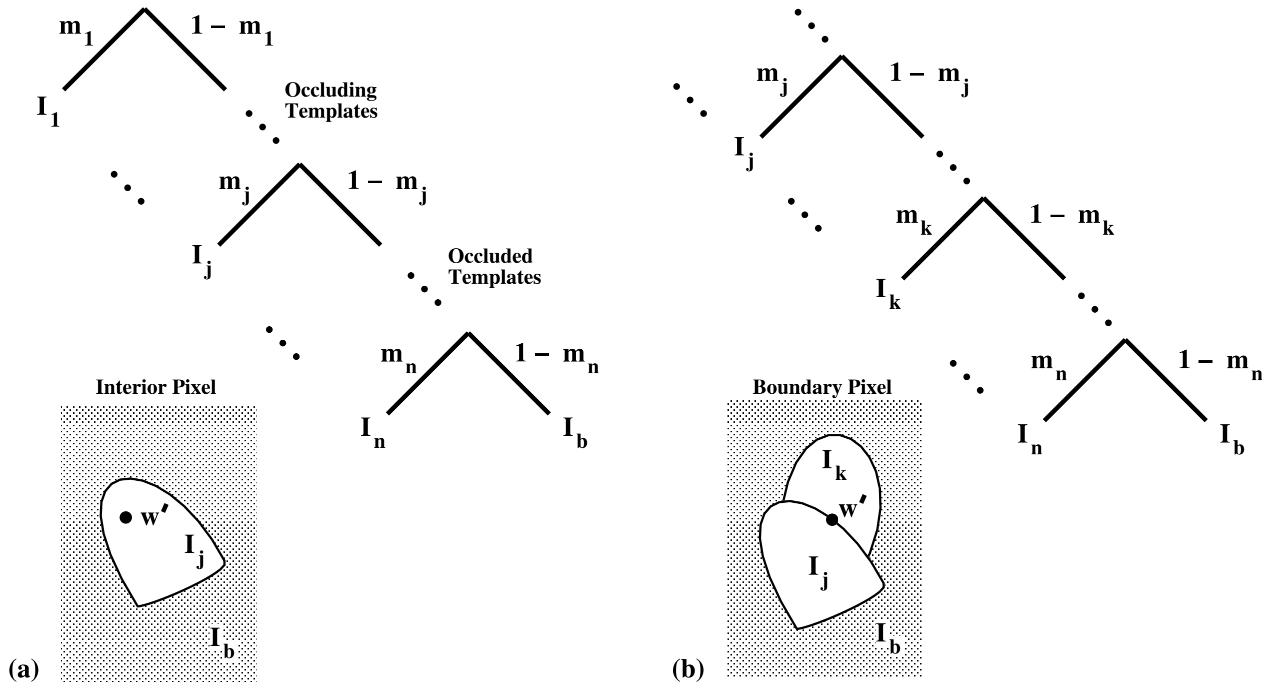


Figure 14: (a) A pixel $\mathbf{w}'$ in the interior of template I_j and the associated window tree, and (b) the same for a boundary pixel.

template plane. The calculation of these kinematic Jacobians is addressed in our previous work on the *DigitEyes* system [20].

Now consider the boundary pixel case, illustrated in Fig. 14 (b). The point $\mathbf{w}'$ is on the boundary of m_j at coordinate $\mathbf{s}_j$, and lies in the interior of m_k at $\mathbf{s}_k$. This implies that the templates between j and k in the tree vanish at $\mathbf{w}'$ along with their derivatives. Furthermore, as in the interior case, the templates occluding I_j and occluded by I_k make no contribution to the Jacobian. This results in the simplified residual,

$$\begin{aligned} R' &= \hat{I}(\mathbf{w}') - [1 - M_1(\mathbf{q}, \mathbf{w}')] \cdots M_j(\mathbf{q}, \mathbf{w}') \hat{I}_j(\mathbf{f}_j(\mathbf{q}, \mathbf{s}_j)) - \\ &\quad [1 - M_1(\mathbf{q}, \mathbf{w}')] \cdots [1 - M_j(\mathbf{q}, \mathbf{w}')] \cdots M_k(\mathbf{q}, \mathbf{w}') \hat{I}_k(\mathbf{f}_k(\mathbf{q}, \mathbf{s}_k)) \ . \end{aligned} \tag{17}$$

Since this is in the form of Eqn. 15, it has an interior Jacobian component as before. Window function gradients from the last two terms yield an additional component. In these terms, only $M_j(\mathbf{q}, \mathbf{w}')$ has a nonzero derivative at $\mathbf{w}'$. Substituting Eqn. 8 for M_j and differentiating yields the boundary Jacobian

$$\mathbf{J}^B_{j,k}(\mathbf{s}') = [I_k(\mathbf{s}_k) - I_j(\mathbf{s}_j)] \frac{\partial \mathbf{f}_j^{-1}}{\partial \mathbf{q}} \frac{\partial m_j}{\partial \mathbf{s}} \ , \tag{18}$$

This boundary component captures the effect of occlusion in covering and revealing pixels as the state changes.

The above discussion shows that the residual Jacobian for a template has two basic types of components: region contributions from Eqn. 16 and boundary contributions from Eqn. 18. This suggests a simple algorithm for Jacobian computation:

1. Scan the segmented image and compute the region contribution to the Jacobian at each visible pixel, using Eqn. 16.

2. Scan the discretized boundary of each template. If a boundary point is visible, identify the template it is occluding and compute the boundary Jacobian term from Eqn. 18.

The above algorithm, along with the segmentation algorithm described in the next section, forms the basis for gradient-based local tracking.

5.5 Algorithms for Image Segmentation

Each pixel in the input image must be assigned to a template in order to compute its contribution to the gradient. This segmentation problem is closely related to the visible surface determination problem in computer graphics: Given a set of polygons in camera coordinates, identify and scan-convert[4] the parts that are visible. Using this analogy, we consider two classes of segmentation algorithms: list-priority and scan-line (see [9], Sec. 15.11.)

Templates are scanned sequentially in visibility order in the list-priority approach, and the most visible template is converted last. Each template is scan-converted independently, and its pixels in the input image are labeled. The visibility ordering ensures that each pixel is correctly labeled at the end of this first stage. The labeled pixels are then rescanned in a second stage to compute the Jacobian, as discussed in the previous section. Pixels contained by overlapping templates are processed multiple times, but template conversion and pixel labeling is simple and fast. This is the segmentation algorithm used in our experiments.

[4] In scan-conversion, polygons (specified by a set of vertices) are mapped into their component pixels in the frame buffer.

Because of the visibility order, this approach is superior to the standard computer graphics depth sorting algorithm, which often splits polygons that can be correctly ordered ([9], Fig. 15.27.)

In contrast, scan-line algorithms sort the template edges on x and y, and scan the image one line at a time. When templates overlap, the visibility order determines the pixel assignment, and coherence is used to avoid unnecessary comparisons. The binary occlusion assumption plays the same role for coherence as polygon nonpenetration in the graphics case. Scan-line algorithms are more efficient than list-priority algorithms: Each pixel is processed once, avoiding redundant calculations. The Jacobian can be computed in one pass, avoiding a labeling stage. They are, however, more complicated to implement.

The scan-line approach can also be used in situations where a visibility order is not available. In this case, the depth at each pixel in the scan-line determines the template order. In this approach, template order is computed in conjunction with segmentation. However, preservation of the ordering under bounded motion is not guaranteed, and this approach may require a prohibitively high sampling rate to work in practice. Moreover, in the case where the template ordering is fixed for a number of estimation steps, this version of the scan-line algorithm is inefficient, as it recomputes the visibility order each time.

6 Experimental Results

We have implemented the representations and algorithms described above in the *DigitEyes* hand tracking system. The current off-line version of the system has demonstrated promising results on a sequence of natural hand intensity images with a significant amount of self-occlusion.

Figure 15 gives an example of the performance of our approach on a two finger motion sequence with significant self-occlusion. In the sequence, the first author's index finger curls into his palm while the hand and remaining fingers are held still. An 80 frame sequence was digitized from videotape and sampled for an effective frame rate of approximately 15 Hz. This resulted in an average finger tip displacement between frames of about three pixels. The camera was positioned at approximately 45 degrees to the table top, facing the palm. It was fully calibrated using the procedure of [24]. As a result of this camera position, the first finger occludes all of the remaining fingers during its motion. In this example, the visibility order does not change throughout the sequence. As a result, the order was computed once before tracking began.

We tracked this sequence with a 9 DOF kinematic model containing the index and middle fingers (3 planar joints per finger) of our full hand model, and allowing full translation. Each finger link was modeled with a single template. We obtained the template model for each link by manually segmenting it from a reference image. Both the input images and templates were convolved with an 13x13 LOG filter. We chose LOG filtering to emphasize the edge properties of the templates. The filter size was chosen empirically. We used unit window functions for all links and initialized the state by hand. The gradient descent algorithm was iterated twice for each frame in the sequence. By limiting the motion to the occluding bodies, any perturbation on the occluded links should be visible as nonzero motion.[5]

Figure 15 shows the estimated model configurations corresponding to the sample points, rendered from the calibrated camera viewpoint. The estimated state trajectories for the motion sequence are shown in Figs. 16 and 17. Deviations in the states of the unmoved finger are on the order of 0.2 radians. Deviances in translation are equally small. The four sources of error in the system are image noise, kinematic model error, photometric template model error, and template shape error.

These results demonstrate the potential of our approach to tracking self-occluding objects. From a classical feature detection perspective, the images in the sequence are quite difficult. All of the phalanges of the middle finger are partially occluded during some portion of the motion sequence, and the index finger is silhouetted against the fingers and palm for most of its motion. A significant advantage of our window-based approach is that it can tolerate any amount of occlusion and continue to extract useful information from the pixels that are visible. The successful tracking of this complicated motion testifies to the power of the kinematic model in constraining the interpretation of the image.

We are in the process of experimenting with more complicated motions involving more

[5] Note however, that there was a small amount of middle finger motion towards the middle of the sequence which is visible in the images.

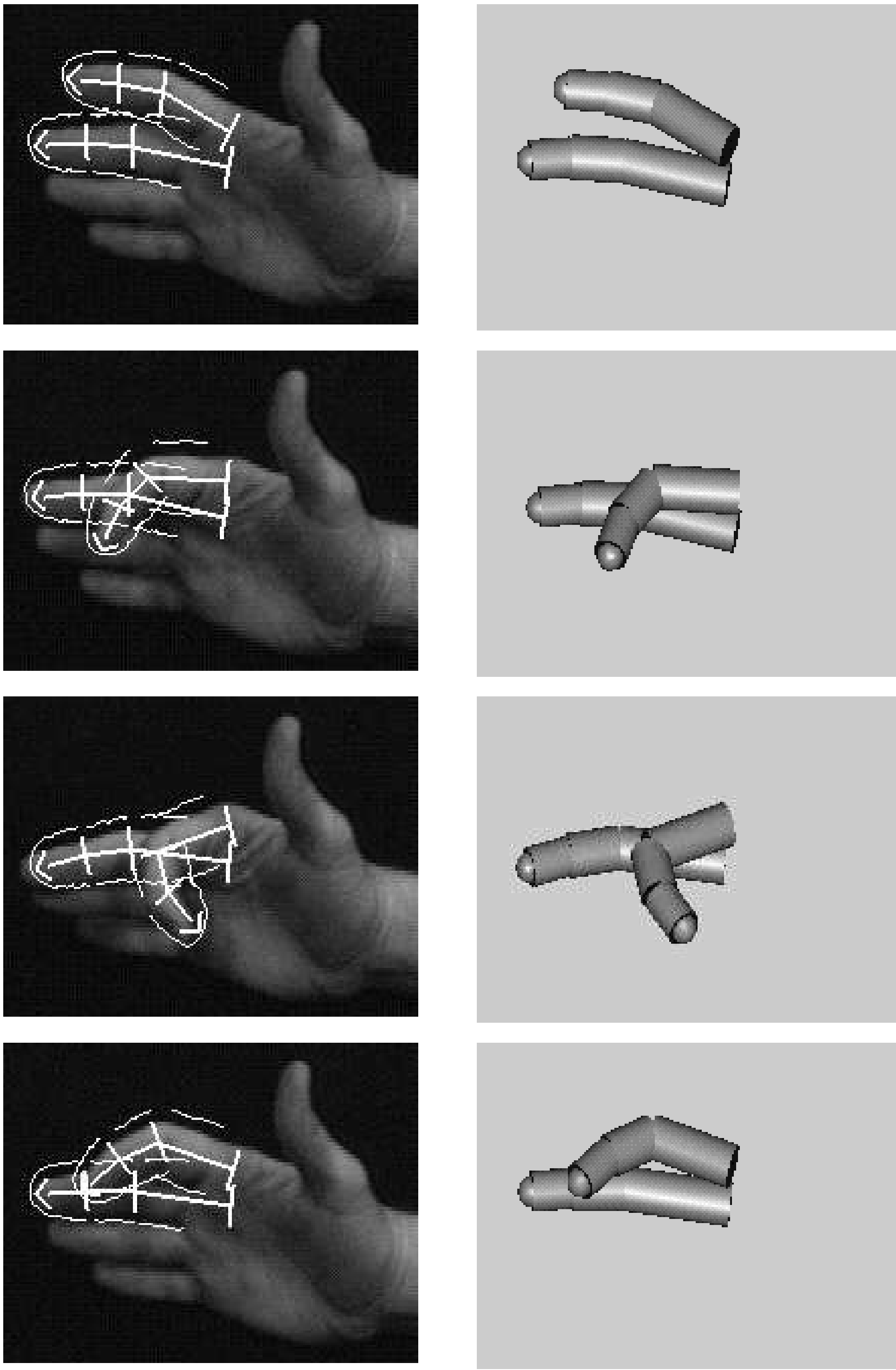


Figure 15: Sample input images and associated state estimates for frames 0, 13, 30, and 75 in the motion sequence. The two finger hand model is rendered with respect to the calibrated camera model using the estimated state. The overlays show the template boundaries and projection of cylinder center axes. These frames were selected for their representative self-occlusions.

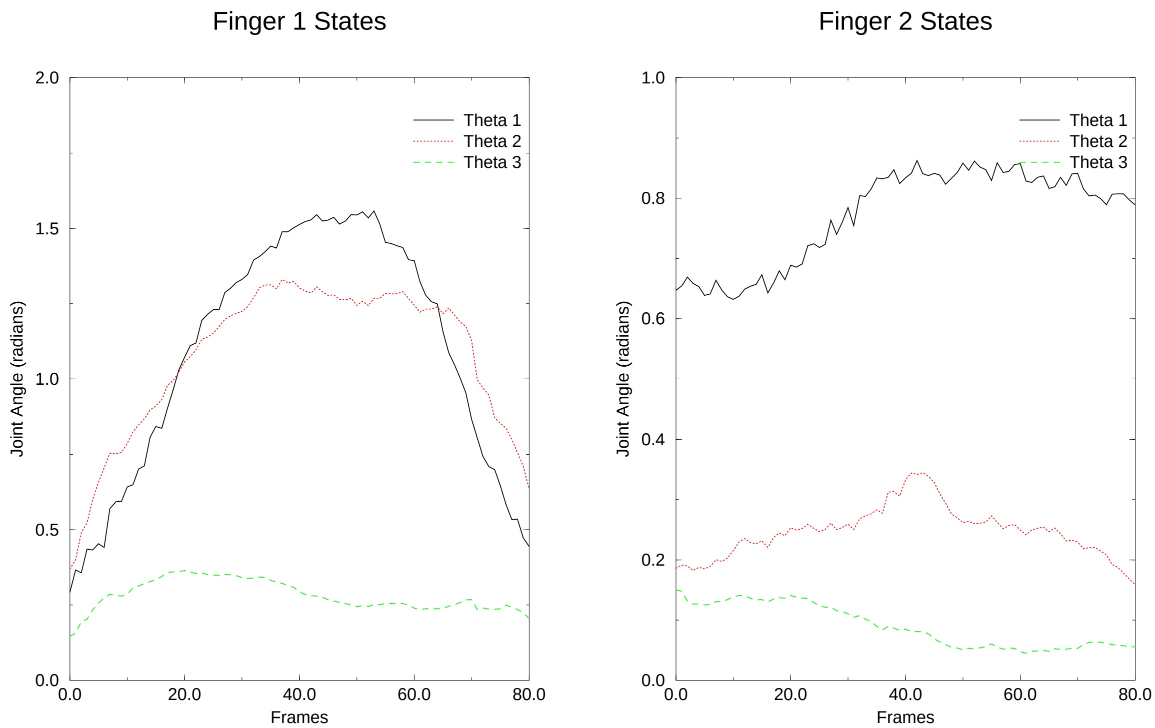


Figure 16: Estimated planar finger motions for two finger model.

DOFs, and investigating the effect of filtering on the tracking performance.

7 Previous Work

In two earlier reports [22, 21], we described the first version of the *DigitEyes* human tracking system. For motion sequences without significant self-occlusion, we demonstrated real-time tracking of a 27 DOF hand model, and developed an application to the 3D cursor user-interface task [21]. The common ground between our previous system and the current approach is the continued use of a 3D kinematic model and high image acquisition rate to constrain the interpretation of the image and facilitate local search. The primary difference is in the representation of the mechanism at the image level. The early system used a simple edge feature detection model which made real-time tracking possible, but was incapable of addressing self-occlusions.

Other previous work on tracking general articulated objects includes [12, 16, 30, 19, 11, 18]. In [30], Yamamoto and Koshikawa describe a system for human body tracking using kinematic and geometric models. They give an example of tracking a single human arm and torso using optical flow features. Pentland and Horowitz [19] give an example of tracking the motion of a human figure using optical flow and an articulated deformable model. In a

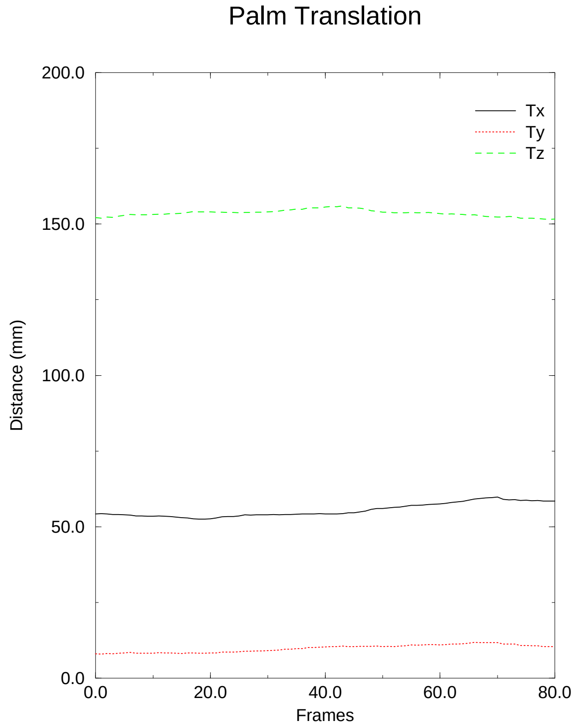


Figure 17: Estimated translation state of two finger model.

related approach, Metaxis and Terzopoulos [16] track articulated motion using deformable superquadric models. In [8], Dorner describes a system for interpreting American Sign Language from image sequences of a single hand. Holt et. al. [12] propose a decomposition of articulated figures into component parts in their approach to monocular tracking. Two of the earliest systems were developed by Hogg [11] and O'Rourke and Badler [18], who analyzed human body motion using constraint propagation. None of these earlier works have demonstrated 3D tracking in the presence of significant occlusions using natural image features.

In addition to previous work on articulated object tracking, many authors have applied general vision techniques to human motion analysis. In contrast to *DigitEyes*, these approaches analyze a subset of the total hand motion, such as a set of gestures [6, 25] or the rigid motion of the palm [2]. Darrell and Pentland describe a system for learning and recognizing dynamic hand gestures in [6]. Related work by Segen [25] takes a neural network approach to 2D hand gesture recognition. While these approaches work in real-time on unmarked hand images, they don't produce 3D motion estimates.

Although many frameworks for human motion analysis are possible, an approach based on full-state 3D tracking has four main advantages. First, by tracking all of the hand's

DOFs, we provide the user with maximum flexibility for interface applications. (See [27, 13] for examples of interfaces requiring a whole-hand sensor.) In addition, our general modeling approach based on 3D kinematics makes it possible to track any subset of hand or body states with the same basic algorithm. Another benefit of full state tracking is invariance to unused hand motions. The motion of a particular finger, for example, can be recognized from its joint angles regardless of the pose of the palm relative to the camera. Finally, by modeling the hand kinematics in 3D we eliminate the need for application- or viewpoint-dependent user modeling.

Our representation for self-occlusion is related to other work in tracking and motion coding. Layered representations based on clustering optical flow are presented in [1, 5, 29]. This work is largely concerned with automatically generating layered, velocity-based representations of a motion sequence that could serve as a model for coding or recognition. A layered representation based on the occluding contours of a single image is described in [17]. This work is complementary to our approach, which is concerned with making the best use of available models. In addition, our representation of self-occlusions is a generalization of layered representations based on depth ordering in the scene, since it is designed to exploit orderings within configuration space.

As a result of modeling self-occlusion in the image plane, we can formulate tracking as a direct optimization problem over an image-based residual error. This approach of coupling the image interpretation (feature detection) problem directly to the model was popularized by SNAKES [14] and has since been applied to a variety of other domains [28].

8 Conclusion

We are developing a visual sensor for human spatial motion. Such a sensor could be located in the user's environment (rather than on their person) and could operate under natural conditions of lighting and dress, providing a degree of convenience and flexibility that is currently unavailable. We model humans as articulated kinematic chains, and propose a framework for local, model-based tracking of self-occluding articulated motion. This model can be applied at a fine (visual) scale to describe hand motion, and at a coarser scale to describe the motion of the entire body.

Self-occlusion is an intrinsic visual property of articulated motion, and may be present even in the most carefully engineered applications. We have developed a novel representation of self-occlusion in configuration space, that generalizes current research in layered representations for motion analysis. We have developed a local tracking algorithm based on our representation, and tested it on image sequences of natural hand motion. We present the first experimental tracking results for nontrivial 3D self-occlusion.

In future work, we plan a real-time implementation of our occlusion-handling algorithm and experimental evaluation of its 3D tracking accuracy. We are interested in the on-line adaptation of our model parameters, articulated structure from motion problems, and the application of this technology to novel user-interfaces.

9 Acknowledgements

The authors thank Luc Robert for making his camera calibration code available, and Fabio Cozman and Heung-Yeung Shum for their careful reading of this report and many useful comments.

A Algorithm for General Visibility Orders

Section 3.3 gives an algorithm for constructing visibility orders for hand templates based on a planar kinematic chain model. Such special case analysis is always the most efficient solution for a specific tracking problem. Using the results from Sec. 4, however, we will now derive an ordering algorithm for arbitrary multi-body objects. This algorithm demonstrates the sufficiency of the existence conditions for visibility orders presented in Sec. 4. Because it isn't based on explicit kinematic analysis, it is considerably more expensive. The main cost is the construction of the motion image of each body, and the associated pairwise occlusion tests.

The general ordering algorithm is an extension of the standard depth-search based graph sorting algorithm [4] to minimize the number of binary object comparisons by only making comparisons to unvisited ("white") vertices.

V is the set of n vertices, Q a LIFO queue of sorted objects in order of "decreasing" visibility (the head of the queue is unoccluded.) The *extent* of an object is the projection of its motion image into the image plane. The routine *Overlap-Extent*(u, v) returns true if the extents of the two bodies intersect. *color*$[u]$ gives the vertex's color, which indicates whether it's unvisited (white), active (gray), or visited (black).

Order(V)

```
for each vertex u ∈ V
    do color[u] ← WHITE
for each vertex u ∈ V
    do if color[u] = WHITE
        then Order-Visit(u)
```

Order-Visit(u)

```
color[u] ← GRAY
for each vertex v ∈ V
    do if color[v] = WHITE
        then if Overlap-Extent(u, v)
            then if Will-Occlude(u, v)
                then Order-Visit(v)
color[u] ← BLACK
Enqueue(Q, u)
```

In general, depth first search is $\Theta(V + E)$. ORDER will always make $n(n+1)/2$ pair-wise occlusion tests. Note that the order produced by sorting the graph is not unique in general, but depends on the starting vertex and the choice of paths.

B Visibility Orders and BSP Trees

The two segmentation algorithms described in Sec. 5.5 are modifications of standard visible surface algorithms that exploit the known visibility order of the object. This appendix develops the connection to computer graphics further, by comparing our visibility ordering approach to another list-priority representation, the Binary Space Partition (BSP) tree [10]. In a BSP tree, each node contains a set of polygons to be rendered. The node's children are formed by dividing its polygons into two sets with a separating plane. Each separating plane partitions the scene into two half spaces, and no polygons on the side containing the viewpoint can be occluded by polygons on the other side. By recursively subdividing the scene space, the tree captures the occlusion relations between a static set of polygons under all possible viewpoints. A visibility-ordered list of polygons can be generated by testing an arbitrary viewpoint against each separating plane during an in-order traversal of the tree.

The BSP tree is a *global* representation of self-occlusion for a six DOF rigid body system.[6] To extend the BSP approach to our articulated DOF domain, we could replace each object by its motion image and apply the BSP tree building algorithm. There is an important problem, however: most of the separating planes will intersect one or more objects. In the rigid body case, polygons can be split in two and assigned to opposite sides of the plane. Even in this case, polygon growth due to splitting is a major practical concern for the algorithm. In the articulated case, splitting is a much more serious obstacle, compounding the already difficult problem of generating the motion images in the first place. There has been some work on perfect binary partitions [7] that don't split objects, but it is not mature enough for application.

References

[1] E. H. Adelson. Layered representation for image coding. Technical Report 181, MIT Media Lab, 1991.

[2] A. Blake, R. Curwen, and A. Zisserman. A framework for spatiotemporal control in the tracking of visual contours. *Int. J. Computer Vision*, 11(2):127–145, 1993.

[3] J. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1988.

[4] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, Boston, MA, 1990.

[5] T. Darrell and A. Pentland. Robust estimation of a multi-layered motion representation. In *Proc. of IEEE Workshop on Visual Motion*, pages 173–178, Princeton, NJ, 1991.

[6] This follows from the definition by viewing the change in viewpoint as rigid body motion relative to a fixed camera.

[6] T. Darrell and A. Pentland. Space-time gestures. In *Looking at People Workshop*, Chambery, France, 1993.

[7] M. de Berg, M. de Groot, and M. Overmars. Perfect binary space partitions. Technical Report RUU-CS-93-23, Utrecht University Dept. of Comp. Sci., 1993.

[8] B. Dorner. Hand shape identification and tracking for sign language interpretation. In *Looking at People Workshop, IJCAI*, Chambery, France, 1993.

[9] J. Foley, J. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990.

[10] H. Fuchs, Z. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):124–133, 1980.

[11] D. Hogg. Model-based vision: a program to see a walking person. *Image and Vision Computing*, 1(1):5–20, 1983.

[12] R. Holt, A. Netravali, T. Huang, and R. Qian. Determining articulated motion from perspective views: A decomposition approach. In J. Aggarwal and T. Huang, editors, *Proc. of Workshop on Motion of Non-Rigid and Articulated Objects*, pages 126–137, Austin, Texas, 1994. IEEE Computer Society Press.

[13] S. B. Kang and K. Ikeuchi. Grasp recognition using the contact web. In *Proc. IEEE/RSJ Int. Conf. on Int. Robots and Sys.*, Raleigh, NC, 1992.

[14] Micheal Kass, Andy Witkin, and Demitri Terzopoulos. Snakes: Active contour models. *Int. J. Computer Vision*, 1(4):321–331, 1987.

[15] J. Koenderink and A. van Doorn. The singularities of the visual mapping. *Biological Cybernetics*, 24, 1976.

[16] D. Metaxis and D. Terzopoulos. Shape and nonrigid motion estimation through physics-based synthesis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(6):580–591, 1993.

[17] M. Nitzberg and D. Mumford. The 2.1-d sketch. In *Proc. Third Int. Conf. on Comp. Vision*, Osaka, Japan, 1990.

[18] J. O'Rourke and N. Badler. Model-based image analysis of human motion using constraint propagation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2(6):522–536, 1980.

[19] A. Pentland and B. Horowitz. Recovery of nonrigid motion and structure. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(7):730–742, 1991.

[20] J. Rehg and T. Kanade. Digiteyes: Vision-based human hand tracking. Technical Report CMU-CS-TR-93-220, Carnegie Mellon Univ. School of Comp. Sci., 1993.

[21] J. Rehg and T. Kanade. Digiteyes: Vision-based hand tracking for human-computer interaction. In J. Aggarwal and T. Huang, editors, *Proc. of Workshop on Motion of Non-Rigid and Articulated Objects*, pages 16–22, Austin, Texas, 1994. IEEE Computer Society Press.

[22] J. Rehg and T. Kanade. Visual tracking of high dof articulated structures: an application to human hand tracking. In J. Eklundh, editor, *Proc. of Third European Conf. on Computer Vision*, volume 2, pages 35–46, Stockholm, Sweden, 1994. Springer-Verlag.

[23] J. Rehg and A. Witkin. Visual tracking with deformation models. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 844–850, Sacramento, CA, April 1991.

[24] L. Robert. Camera calibration without feature extraction. *Computer Vision Graphics and Image Processing*, 1995. Accepted for Publication.

[25] J. Segen. Gest: A learning computer vision system that recognizes hand gestures. In R. Michalski and G. Tecuci, editors, *Machine Learning IV*. Morgan Kaufmann, 1993.

[26] M. Spong. *Robot Dynamics and Control*. John Wiley and Sons, 1989.

[27] D. Sturman. *Whole-Hand Input*. PhD thesis, Massachusetts Institute of Technology, 1992.

[28] D. Terzopoulos, A. Witkin, and M. Kass. Energy constraints on deformable models: Recovering shape and non-rigid motion. In *Proc. of AAAI-87*, pages 755–760, 1987.

[29] J. Wang and E. Adelson. Layered representation for motion analysis. In *Proc. IEEE Conf. Comput. Vis. and Pattern Rec.*, pages 361–366, 1993.

[30] M. Yamamoto and K. Koshikawa. Human motion analysis based on a robot arm model. In *IEEE Conf. Comput. Vis. and Pattern Rec.*, pages 664–665, 1991.