THE ROLE OF FEATURE VISIBILITY CONSTRAINTS IN PERSPECTIVE ALIGNMENT

Gilbert Verghese and John K. Tsotsos
Department of Computer Science, University of Toronto, Toronto, CANADA M5S 1A4
email: verghese@vis.toronto.edu

ABSTRACT

Perspective Alignment is a novel method of solving back-projection, the well-known problem of computing three-dimensional (3D) position and orientation (pose) of a model from two-dimensional (2D) image features. This paper demonstrates that previous back-projection methods can violate the *visibility* constraint by computing solution poses in which the model occludes features which should be visible. By definition, these *visibility errors* are associated with incorrect pose solutions. Yet they occur frequently when previous back-projection methods are used in *underconstrained* situations. We empirically analyze the frequency and consequences of visibility errors in previous back-projection methods. We then show how Perspective Alignment satisfies the visibility constraint during the pose solution process to eliminate these errors.

The algorithm has been implemented and used in a real-time model-based object tracking system. We describe the algorithm and results of tracking real objects in real-time. The algorithm also has implications for reducing the combinatorics of image-model feature pairing in model-based recognition.

1. INTRODUCTION

Perspective Alignment performs back-projection efficiently through a series of incremental geometric pose restrictions which satisfy any number of image edge constraints on model pose. Being constraint-based and incremental, Perspective Alignment can compute incomplete pose estimates in underconstrained situations and can apply additional pose constraints, such as visibility, to these estimates. It maximally constrains model pose when either partial, unique, or multiple pose solutions exist.

This paper shows how to satisfy visibility constraints during 2D-to-3D model-based back-projection. The most closely related previous work is by [Goad 86], [Ikeuchi 87], [Dhome et al 89], and [Lowe 91].

[Goad 86] and [Ikeuchi 87] use an inadequate representation for perspective visibility. They store object views taken from points on a viewpoint sphere around the object. That is a complete description of the object's orthographic views. However, consider a truncated hollow tetrahedron under perspective. The interior edges are all visible from within the truncated region, whereas interior edges eventually become occluded as the viewpoint translates away from the tetrahedron. It is well known that the Aspect space is two-dimensional in the orthographic case (the viewpoint sphere) and three-dimensional in the perspective case (all of 3D space) [Stewman and Bowyer 88], [Plantinga and Dyer 90]. Thus Goad and Ikeuchi use a representation of visibility which is inadequate for perspective projection.

[Dhome et al 89] eliminate visibility errors after pose computation in overconstrained situations only. [Lowe 91] proposed a method of computing pose in underconstrained situations. We show that his method produces visibility errors in those situations (Figure 2). The usefulness of the Perspective Alignment algorithm with visibility constraint satisfaction is demonstrated by the effectiveness of our new algorithm in eliminating these errors. A large number of simulations confirm our findings.

Let us define the *visibility constraint:* No model feature corresponding to a (visible) image feature may be occluded in the (opaque) model's posed perspective projection. We say a *visibility error* has occurred in pose solution whenever this constraint is violated. As far as we know, ours is the first perspective back-projection method that correctly satisfies visibility constraints in underconstrained situations.

2. VISIBILITY ERRORS BY PREVIOUS SYSTEMS

Figure 1 shows 2 views of a tetrahedron in which 1 occluded and 3 visible line-segment features align. This is an example where despite 4 correct image-model feature pairs and no remaining degrees of freedom, back-projection on the basis of geometric constraints is ambiguous. Most back-projection methods do not recognize this ambiguity and simply return one of the possible poses. This can have disastrous effects in applications, such as robotics, where correct pose is essential.

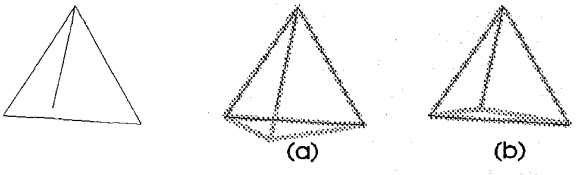

Figure 1.

Object perception generally stabilizes when we view real, solid, opaque objects rather than wire-frames. Likewise, a solid model can eliminate much of the ambiguity in pose solution, provided the object's opacity is used in the pose solution process. For example, the match in Figure 1(a) can be eliminated since it contains a visibility error.

We now demonstrate empirically that Newton-Raphson iteration [Lowe 92] violates the visibility constraint with high frequency.

In order to simulate 2D feature-detected images, wire-frame models were projected and hidden lines removed. A variety of convex and non-convex object models were used. We

0-8186-7310-9/95 $4.00

simulated images with resolutions of 256×256, 512×512, and 1024×1024 pixels, resulting in realistic amounts of spatial aliasing. Each back-projection problem was simulated as follows. The model was moved to a random pose. A set of image features was generated by projecting and clipping the target model from another random pose (in the field of view, rotated less than 30° about a random axis). The original pose and correct image-model feature pairings were then given to the back-projection system. Failure to align given features to within the extent of spatial aliasing was considered back-projection error. Resulting model self-occlusion of given features was considered visibility error. To simulate partial occlusion by other scene objects, we varied the number of given features from 7 to 1. We measured the relative frequency of error in 10,000 random back-projection problems. Newton-Raphson iteration with Levenberg-Marquardt stabilization [Lowe 92] produced a low frequency of back-projection error but a high frequency of visibility error. Figure 2 plots back-projection error frequency versus the number of features used for each object and image resolution.

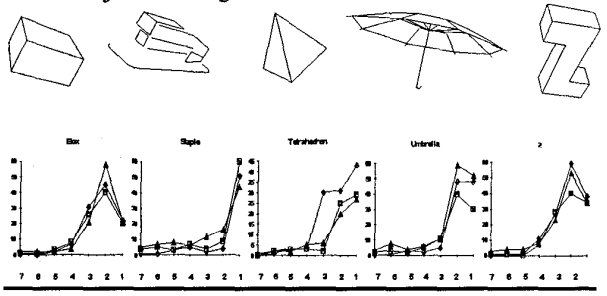

Figure 2. Frequency of Visibility Errors (0-60%) versus Number of Features (7-1)
Resolution: ▲ 256×256, ■ 512×512, ◆ 1024×1024

The average visibility error frequency over all objects and resolutions was 16% for 3 features, 36% for 2 features, and 37% for 1 feature. We can conclude from these simulations and analysis that the iterative locally-linear algorithm produces visibility errors with relatively high frequency when few features are provided. Notice that the error frequency is sometimes higher with 2 features than with 1. This is not surprising since the 2-feature case can have up to twice the likelihood of self-occlusion.

In contrast, the Perspective Alignment method explicitly avoids visibility errors, which, as we saw above, occur with high frequency in underconstrained situations. In the same simulations as above, Perspective Alignment with visibility constraint satisfaction produced no visibility errors.

3. APPLYING VISIBILITY CONSTRAINTS DURING PERSPECTIVE ALIGNMENT

As we saw in the previous section, visibility errors can occur frequently in pose solution. This section presents our back-projection algorithm for incrementally determining pose by applying Perspective Alignment and visibility constraints.

3.1 Overview

A line-segment in the image plane and the camera origin define a plane in space (the so-called *interpretation plane*). The corresponding model edge should lie in this plane. In that case, the model edge satisfies the *Perspective Alignment constraint*. In the Perspective Alignment algorithm, edge correspondences are considered in the order given. Alignment relationships between real-world features and their perspective projections are used to allow each edge in the sequence to contribute the maximal additional geometric constraint [Grimson 90] to the previous constraints on object pose. Thus, as many degrees of freedom as possible are determined given the available feature information [Verghese 93]. Remaining degrees of freedom or pose solutions are used to satisfy additional constraints, such as visibility.

3.2 Modeling

For the purposes of comparison with other algorithms, we have modeled objects as rigid polyhedra. Objects may be concave or consist of disjoint parts. In order to use object opacity in pose solution, we store model self-occlusion information. We store each object's wire-frame and augment this representation with line-segment (*model edge*) visibility information. We store a model-centered description of all viewpoints from which each model edge is visible (not occluded by the model itself) or partly visible (partially occluded by one or more model faces). If the model occludes a given edge, then we can translate and rotate the model to place the viewpoint in the nearest visibility region and hence make any edge visible (or partly visible).

Each edge's visibility region is represented by a *union of intersections of arbitrary half-spaces*. Each half-space is represented by a point and an oriented normal in model-centered coordinates. We can describe any plane-bounded region of space with this representation. The adequacy of unions of intersections of half-spaces is apparent from the observation that a polyhedron's edges have plane-bounded visibility regions [Tarabanis and Tsai 89, 92].

3.3 Algorithm

The flow chart in Figure 3 describes the order in which constraints are satisfied by the algorithm. At each step, special care must be taken not to violate previously satisfied constraints. The algorithm first satisfies all available Perspective Alignment constraints and then uses any remaining degrees of freedom to satisfy visibility. If no degrees of freedom remain, and multiple poses satisfy alignment, then candidate poses violating visibility are eliminated. Additional constraints can also be applied to select a final pose.

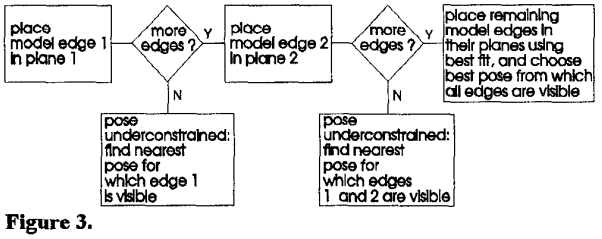

Figure 3.

3.3.1 Finding the nearest visibility point

The Perspective Alignment algorithm aligns the first model edge with its paired image edge, as described in [Verghese 93]. This computation places the model in its *interpretation plane* defined by the image edge and the camera origin. The model subsequently has 4 degrees of freedom: rotation about the edge and about the plane's normal, and two translational degrees of freedom within the plane. If there are no additional alignment constraints available, these degrees of freedom may be used to satisfy the visibility constraint.

The strategy for satisfying visibility of the model edge is to find a point p in the edge's visibility region nearest the current viewpoint (CVP), move the model relative to the camera to make p the new viewpoint, and finally rotate the model about the new viewpoint to resatisfy the alignment constraint (place the edge into its interpretation plane).

A model edge's visibility region is represented by a union of intersections of half-spaces. An intersection of half-spaces is a convex plane-bounded region in space, a convex visibility region (CVR). We find the nearest among the CVP's nearest points to each of the CVRs. Consider a single CVR. It is defined by planes P_i and oriented normals n_i (pointing into the CVR). Finding the nearest point appears to be a linear programming problem. We have convex plane-bounded regions within which to find an optimal point. However, the objective function to be minimized, Euclidean distance, is not linear. The pseudo-code below shows how to locate the point in the CVR closest to the CVP. It does so by eliminating candidate viewpoints in the following order: (1) CVP, (2) points in the faces of the CVR, (3) points in the ridges of the CVR, (4) vertices of the CVR.

```
find all  planes (half-space boundaries) that the CVP
  (current viewpoint) violates.
if none, then answer=CVP, done.
if the perp projection (PVP) onto any of the violated
  planes satisfies all the plane constraints, then
  answer=PVP, done.
clear the list of intersection lines.
for (i=0; i<numplanes; i++)
   for (j=i+1; j<numplanes; j++) {
      compute the intersection line L of P_i and P_j.
      take the outward pointing normals -n_i and -n_j of
      P_i and P_j.
      take their cross products with L.
      orient each cross product to agree (+ve dot prod.)
         with the other outward normal.
      if (perp vector from L to CVP agrees with both
            cross products) {
         /* answer may be in the ridge */
         PVP = perp projection of CVP onto L.
         if PVP satisfies all planes, then answer=PVP,
         done.
         else save the intersection line L for vertex
            checking.
      }
}
/* Checked ridges to no avail, so check the vertices
  using the intersection lines L. */
for (i=0; i<numlines; i++)
   for (j=i+1; j<numlines; j++)
      if intersection I of lines L_i and L_j exists and
         satisfies all planes
      /* when testing with planes, save the out pointing
         direction of each L */ {
         for (k=j+1; k<numlines; k++)  {
            if (L_k intersects I) save the outward
            pointing direction along L_k.
            if the vector from I to CVP satisfies all
               saved constraints, answer = I, done.
         }
}
```

3.3.2 1-edge case

Having found the nearest visibility point, we move the model, if needed, to satisfy the visibility constraint for one model edge. If we translated the model, the model edge remains parallel to its constraint plane. If necessary, we must rotate the model about the viewpoint to place the edge back in its constraint plane to satisfy the alignment constraints. Note that model rotations about the viewpoint maintain the edge's visibility by maintaining the viewpoint's relative position in the convex visibility region. To place the edge back into its plane, we rotate the required amount about the axis through the viewpoint parallel to the model edge.

3.3.3 2-edge case

In this case we have satisfied alignment constraints for 2 model edges. [Verghese 93] shows that if these edges are not collinear, then model pose is left with one rotational and one translational degree of freedom. If there are additional features, then we consider their alignment constraints before satisfying visibility. If there are no additional features, our strategy for satisfying visibility will be to instantiate the free rotational parameter thereby restricting pose to translation and viewpoint to a line. We rotate the model up to 180° in each direction, alternating directions, and terminating if the orientation satisfies visibility.

When we fix the rotational pose parameter, viewpoint is restricted to a specific line in space. In our visibility data structure, we have one list of convex visibility regions for each edge, and the final viewpoint must be in at least one region from each list. (Recall that each convex visibility region is represented by a list of intersecting half-spaces. The intersection of a pair of regions is represented by the concatenation of their lists of intersecting half-spaces.) We take pairs of regions, one from each list, and determine whether the line restricting viewpoint intersects the regions' intersection. This is done simply and efficiently by cutting the line off at its intersections with half-space boundary planes. If any boundary plane cuts off the entire remaining portion of the line, the intersection with the line is empty, and we consider the next pair. If we run out of pairs, then we try another value of the free rotational pose parameter. Otherwise, the remaining portion of the line satisfies the alignment and visibility constraints.

3.3.4 3-or-more-edge-case

When there are three or more edges, then it is possible that pose is fully determined. In that case, there may be multiple (but finitely many) solutions. The Perspective Alignment algorithm computes all possible pose solutions. We simply eliminate those which violate visibility. If there remain more than one solution, then they may be useful in satisfying further constraints. If none remain, then there was an image-model feature matching error.

When there are three or more edges, it is possible that pose is not fully determined and that a translational degree of

freedom remains [Verghese 93]. In that case, we use the procedure described in the 2-edge-case for testing visibility when viewpoint is restricted to a line.

3.3.5 Results and Conclusions

Resulting pose solutions may be unique, multiple (finitely many), or infinitely many (but maximally constrained), depending on the number and type of constraints provided by the features given. In any case, the Perspective Alignment method finds all solutions and rejects those which violate the visibility constraints. Temporal continuity can be used, along with uncertainty measures, to resolve multiple solutions for tracking purposes. To achieve alignment of each image-model feature pair, we first move the model to the predicted pose if prediction is being used. We apply the Perspective Alignment algorithm to maximally align the model with the given features and to describe the remaining degrees of freedom. We then use these degrees of freedom to adjust model pose to satisfy visibility.

We used the Perspective Alignment algorithm with visibility constraint satisfaction in a real-time, model-based tracking system. Our real-time multiprocessor implementation uses a DataCube MV-200 image processor for 30 Hz, 512×480 frame-grabbing and edge detection, a SUN SPARC II for model overlay, and a Silicon Graphics Power Series 4D/380VGX with 8 processors for Perspective Alignment with visibility constraint satisfaction. The system processes approximately 22 frames/sec and tracks rigid polyhedral objects moving up to 132 pixels/sec in the image. This compares favourably with other tracking methods ([Dickmanns 90], [Stephens 90], [Lowe 92], [Daucher et al 93]).

The frames in Figure 4 (viewed left to right and top to bottom) show a Z object being tracked by our tracker. They are snapshots from a real-time motion-tracking sequence. Projected model segments are overlaid with original images to demonstrate model pose maintenance by the algorithm. The sequence included several changes in aspect accompanied by appearance and disappearance of object edges.

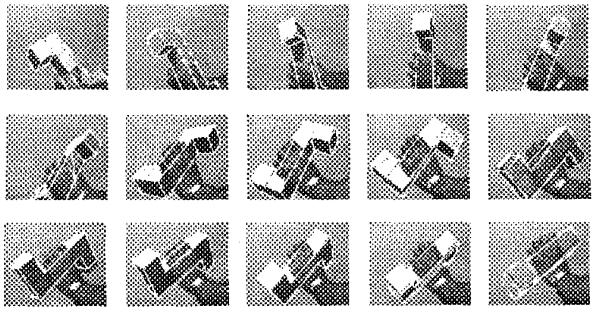

Figure 4.

When an image object feature disappears behind another, the two features can briefly coincide. It is important for the tracker to have knowledge of which corresponding model feature is at a visibility region boundary. It can then decide which model feature to match with the remaining image feature. Likewise, when an image object feature disoccludes from behind another, the tracker must know which model feature is near a visibility region boundary and which of the two image features is within that visibility region. Used in this manner, the visibility constraint allows us to continue tracking through aspect changes.

A limitation of the method is that it applies only to rigid, plane-faced objects. A possible extension is to model visibility of articulated object features for Perspective Alignment with visibility constraint satisfaction. Being constraint-based and incremental, Perspective Alignment lends itself well to computing pose of articulated objects. Efficient modeling and run-time computation of visibility information for articulated objects requires further research.

References

[Daucher et al 93] N. Daucher, M. Dhome, J.T. Lapresté, G. Rives, Modeled object pose estimation and tracking by monocular vision, *Proc. British Machine Vision Conference*, 1993, 269-258.

[Dhome et al 89] M. Dhome, M. Richetin, J. Lapresté, G. Rives, Determination of the attitude of 3D objects from a single perspective view, *IEEE Trans. Pattern Anal. Mach Intell.* 11 12, 1989, 1265-1278.

[Dickmanns 90] E.D. Dickmanns, Visual dynamic scene understanding exploiting high-level spatio-temporal models, *Proc. ICPR*, 1990, 373-378.

[Goad 86] Fast 3D model-based vision, in *From pixels to predicates: recent advances in computational and robotic vision,* A.Pentland, ed., Norwood, N.J.: Ablex Pub. Corp., 1986, 371-391.

[Grimson 90] W. E. L. Grimson, *Object recognition by computer: the role of geometric constraints*, MIT Press, Cambridge, Mass., 1990.

[Huttenlocher and Ullman 90] D.P. Huttenlocher and S. Ullman, Recognizing solid objects by alignment with an image, *Int. J. Computer Vision 5,* 1990, 195-212.

[Ikeuchi 87] K. Ikeuchi, Generating an interpretation tree from a CAD model for 3d-object recognition in bin-picking tasks, *Int. J. Computer Vision 1, 2,* 1987, 145-165.[Lowe 91] D. G. Lowe, Fitting parameterized three-dimensional models to images, *IEEE Trans. Pattern Anal. Mach Intell. 15*, 1991, 441-451.

[Lowe 92] D. G. Lowe, Robust model-based motion tracking through the integration of search and estimation, *Int. J. Computer Vision* 8 2, 1992, 113-122.

[Plantinga and Dyer 90] H. Plantinga and C.R. Dyer, Visibility, occlusion, and the aspect graph, *Int. J. Computer Vision 5,* 1990, 137-160.

[Stephens 90] R. Stephens, Real-time 3D object tracking, *J. Image and Vision Computing* 8, 1, 1990, 91-96.

[Stewman and Bowyer 88] J. Stewman and K. Bowyer, Creating the perspective projection aspect graph of polyhedral objects, *Proc. ICCV*, 1988, 494-500.

[Tarabanis and Tsai 89] K. Tarabanis and R. Tsai, Viewpoint planning: the visibility constraint, *Proc. DARPA Image Understanding Workshop*, 1989, 23-26.

[Tarabanis and Tsai 92] K. Tarabanis and R. Tsai, Computing occlusion-free viewpoints, *Proc. CVPR*, 1992, 802-807.

[Verghese 93] G.Verghese, Perspective Alignment back-projection for monocular tracking of solid objects, *Proc. British Machine Vision Conference*, 1993, 217-228.

[Verghese and Tsotsos 94] G. Verghese and J.K. Tsotsos, Real-time Model-based Tracking Using Perspective Alignment, *Proc. Vision Interface,* 1994, 202-209.