MIT LCS TM-561

Matching and Pose Refinement with Camera Pose Estimates

Satyan Coorg Seth Teller

Computer Graphics Group

December, 1996

Matching and Pose Refinement with Camera Pose Estimates

Satyan Coorg **Seth Teller**
MIT Computer Graphics Group
545 Technology Square NE43-217
Cambridge MA 02139
satyan@graphics.lcs.mit.edu, http://graphics.lcs.mit.edu

Abstract

This paper describes novel algorithms that use absolute camera pose information to identify correspondence among point features in hundreds or thousands of images. Our *incidence counting* algorithm is a *geometric* approach to matching; it matches features by extruding them into an absolute 3-D coordinate system, then searching 3-D space for regions into which many features project.

The absolute pose estimates reported by our instrumentation are accurate, but not perfect. Thus, we also consider the problem of *refining* these pose estimates, given feature matches from a set of images. We describe a pose refinement algorithm which decouples translation (position) estimates from rotation (attitude) estimates, and can incorporate matches from many hundreds or thousands of images.

1 Introduction

Many 3-D reconstruction algorithms rely on a *matching* or *correspondence* step to identify constraints corresponding to the scene geometry; these constraints are used to guide the 3-D reconstruction process. Typically, matching is performed using some image attribute (e.g., pixel luminance [?]) or some geometric attribute (e.g., length and orientation of edges [?]). While these techniques work well for images taken from nearby camera positions, they are less effective for disparate images taken from cameras that are far from each other.

In this paper, we design a matching algorithm that uses camera pose estimates (provided by physical instrumentation) to *over-constrain* the matching problem, identifying matches by applying geometric constraints imposed by the camera positions. In some ways, our algorithm is similar to use of the *epipolar* constraint in stereo vision [?], but generalizes that method in its incorporation of many cameras and images.

As the absolute pose estimates reported by our instrumentation are not perfect, we also consider the problem of camera pose refinement, i.e., computing accurate camera poses for many images, given matches between points and fairly accurate initial pose estimates.

Much of the existing research on pose refinement has revolved around the assumption that *no* 3-D information is available [?, ?]. The basis of these algorithms is the epipolar constraint between two images:

$$\tilde{\mathbf{m}}^T \mathbf{F} \tilde{\mathbf{m}}' = 0$$

where $\mathbf{F}$ is the 3×3 *fundamental* matrix relating two (projective) points $\tilde{\mathbf{m}}$ and $\tilde{\mathbf{m}}'$ in the two images. Determining the fundamental matrix is equivalent to determining the (relative) poses of the two cameras involved. Given eight or more point correspondences, it is possible to determine the fundamental matrix up to a scale factor using the *eight point* algorithm [?]. However, typical algorithms [?, ?] use more points than eight in order increase the robustness of the algorithm.

While this technique performs well for pairs of images, there are several disadvantages in using the fundamental matrix technique for a large number of images. First, these algorithms involve only pairwise matching; using them to compute pose for m cameras pairwise may result in large "drift" error. Second, they determine camera pose and 3-D positions only up to a projective transformation, which needs to be "corrected" as a post-processing step. Third, the use of projective matrices increases the complexity of the solution because of greater number of variables and more complicated constraints (such as singularity).

In contrast, we formulate the problem as a direct 3-D optimization algorithm that *refines* initial camera pose estimates. One advantage of this approach is that the number of unknown variables is less, increasing the robustness of the algorithm. Also, the algorithm can seamlessly incorporate matches across many images. Finally, from a practical standpoint, it is much easier to visualize and debug (using computer graphics) algorithms operating in 3-D; this would be much harder for algorithms that operate in more complex spaces.

2 Incidence Counting

The incidence counting algorithm is based on the following property of projection: if any *sparse* set of features in multiple images are extruded to 3-D, then it is likely that regions of high incidence (regions where extrusions from multiple cameras intersect) correspond to real 3-D features. Figure **??** illustrates

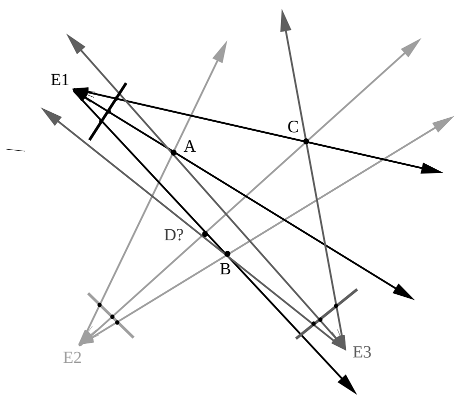


Figure 1: Incidence counting in two dimensions.

the idea of the algorithm in 2-D. In the figure, $E1$, $E2$, and $E3$ are cameras imaging three features (points A, B, C). The extrusions of the image features are *rays* originating from the camera and passing through the feature. If a feature is present in k images, k rays would intersect at the location corresponding to the feature (e.g., points A, B, C all have high incidence of $k = 3$). Thus, a simple way to identify matches would be search for regions with high incidence; an efficient method to perform the search is described in Section **??**.

Note that, in addition to the "true" features, there are also *spurious* regions with high incidence. For example, even though point D was not one of the features imaged by the cameras, D has the property that rays from all three cameras pass close to it; i.e., D is a possible candidate for a match. Section **??** provides a method to eliminate some spurious matches by associating an error value with each 3-D position. Future work will incorporate methods using image attributes (color and texture) to eliminate additional spurious matches.

2.1 Octree-Based Incidence Counting

Our algorithm for incidence counting requires two parameters in addition to the images and camera poses:

- ϵ, a "nearness" threshold. This is necessary to handle (small) errors in either the location of the image feature or in camera pose. The choice of ϵ depends on both the accuracy of the camera pose estimate, and the desired accuracy of the reconstruction.
- k, the incidence threshold. It is related to the *density* of camera positions relative to the features of interest – a reasonable value would be the average number of cameras imaging a feature.

Given these parameters, points of high incidence are those for which k or more rays pass by within a distance ϵ.

Possible methods of identifying high incidence are to check the above condition for (1) all k-cardinal subsets of the set of rays, or (2) all 3-D points in a discrete set (e.g., regular grid). Both these methods have disadvantages. Checking all possible subsets suffers from a combinatorial increase in complexity with k; checking only a discrete set of 3-D points suffers from the usual problems of point sampling (i.e., missing some 3-D feature (undersampling), or inefficiency (oversampling)).

Fortunately, rays constructed by extrusion exhibit the clustering property: while there are regions of high density (e.g., near features), there are large regions containing very few rays. We exploit this property by constructing an octree [**?**] to store the rays. The octree is constructed by associating the region of interest (a bounding-box overestimate of the cameras and the scene to be modeled) with the root node, and subdividing octree nodes until either each leaf node is associated with fewer than k rays[1], or its dimensions are less than ϵ. Once the octree has been constructed, each leaf node is examined to check whether the rays through it pass through within ϵ of each other. This can be performed by computing the (least-squares) best point lying on all these rays. The algorithm reports all the points (and corresponding rays) whose error is less than ϵ.

2.2 Eliminating Spurious Matches

As mentioned earlier, one drawback of the incidence counting algorithm is that it identifies even spurious matches. In this section, we design an algorithm to eliminate some spurious matches by enforcing the constraint that a single ray can contribute to at most one 3-D point. The algorithm given below uses the error metric associated with a 3-D point to choose at most one 3-D point for each ray. Informally, it uses the criteria that 3-D points with low error are retained, and those with high error are rejected.

This algorithm also has the property that it computes the minimum error valid configuration (in a lexicographic sense).

Algorithm Check-Spurious:

```
1. Sort all (say, $n$) high incidence 3-D points according to their error (the lowest error being first).
   $\mathbf{P_i}$ denotes the $i^{th}$ 3-D point.
2. foreach $1 \leq i \leq n$ do
   (a) if ($\mathbf{P_i}$ is invalid) continue;
   (b) Output $\mathbf{P_i}$ as a valid point.
   (c) foreach $i < j \leq n$ such that $\mathbf{P_i}$ and $\mathbf{P_j}$ share a ray, mark $\mathbf{P_j}$ as invalid.
```

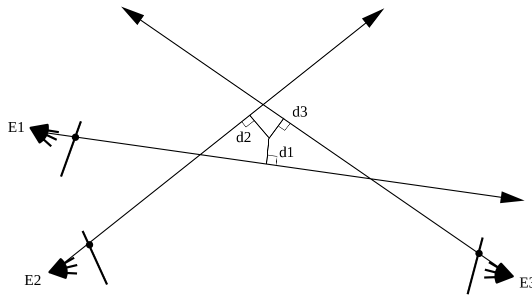


Figure 2: Reconstruction using least squares of distances.

3 A Direct 3-D algorithm for Camera Pose Refinement

We now present an algorithm for refining camera pose estimates, given matches across points in different images. Figure **??** illustrates (in 2-D) the idea behind our algorithm. If the camera poses are accurate, then the rays constructed by extrusion would pass through the reconstructed 3-D point. Typically, due to error in the camera pose estimates, they will diverge from the reconstructed point. This can be used to "correct" the camera poses so that the rays are as close as possible to the 3-D reconstruction.

Formally, the pose refinement problem is as follows. Given:

- For $1 \leq i \leq m$, $\mathbf{E}'_i$ and R'_i – the translation and rotation estimates of the i^{th} camera;
- For $1 \leq i \leq m$, $1 \leq j \leq n$, rays $\mathbf{v}_{ij}$ – unit vectors that correspond to projections of point $\mathbf{P}_j$ from camera i (in the camera's coordinate system);

compute $\mathbf{E}_i$, R_i for $1 \leq i \leq m$ (the true pose of each camera), and $\mathbf{P}_j$ for $1 \leq j \leq n$ (the correct 3-D positions each matched point).

We formulate the problem as a minimization of the following objective function[2]:

$$O = \sum_{i=1}^{m} \sum_{j=1}^{n} \|(\mathbf{P}_j - \mathbf{E}_i) \times R_i(\mathbf{v}_{ij})\|^2$$

Geometrically, this function represents the sums of the squared distances from reconstructed points to their corresponding rays (Figure **??**).

[1]We associate a ray with an octree node if it intersects the ϵ-extended box around the node.

[2]Note that $\mathbf{P}_j = \mathbf{0}$ and $\mathbf{E}_i = \mathbf{0}$ is a trivial solution to the minimization problem. This can be avoided by imposing a constraint that the sum of their magnitudes must be some non-zero constant. In practice, due to the use of initial pose estimates, we have found that the optimization converges to non-trivial solutions.

As the objective function does not have a linear least-squares form, we use an iterative method to solve for camera pose. Our approach is to consider the problems of finding each transformation independently (assuming the other is known accurately) and combining the two methods when neither translations nor rotations are known exactly. While this is equivalent to minimizing the objective function using partial derivatives with respect to translations and rotations, it is helpful to separate the two cases for clearer presentation; solutions to these two cases turn out to be quite different.

3.1 Translations

In this section, we solve for translations of the cameras, assuming that their rotations are known accurately. Thus, $R_i(\mathbf{v}_{ij})$ can be replaced by a (known) unit vector $\mathbf{v'}_{ij}$. The resulting objective function has the following form:

$$O = \sum_{i=1}^{m} \sum_{j=1}^{n} \|(\mathbf{P}_j - \mathbf{E}_i) \times \mathbf{v'}_{ij}\|^2$$

which can be written as:

$$O = \sum_{i=1}^{m} \sum_{j=1}^{n} \|\mathbf{L'}_{ij}(\mathbf{P}_j - \mathbf{E}_i)\|^2$$

where $\mathbf{L'}_{ij}$ is the 3×3 skew-symmetric matrix defining the cross product whose elements are determined by the components of $\mathbf{v'}_{ij}$:

$$\begin{bmatrix} 0 & v'_{ij,3} & -v'_{ij,2} \\ -v'_{ij,3} & 0 & v'_{ij,1} \\ v'_{ij,2} & -v'_{ij,1} & 0 \end{bmatrix}$$

Writing $\|\mathbf{x}\|^2 = \mathbf{x}.\mathbf{x}$ as $\mathbf{x}^T\mathbf{x}$, we obtain:

$$O = \sum_{i=1}^{m} \sum_{j=1}^{n} (\mathbf{P}_j - \mathbf{E}_i)^T \mathbf{L'}^T_{ij} \mathbf{L'}_{ij} (\mathbf{P}_j - \mathbf{E}_i)$$

This is of the form $\mathbf{x}^T\mathbf{A}\mathbf{x}$ for where $\mathbf{A}$ is a symmetric matrix. The derivative of this function with respect to $\mathbf{x}$ is $\mathbf{A}\mathbf{x}$.

Computing the derivatives of this function with respect to $\mathbf{P}_j$, and setting it to $\mathbf{0}$ yields:

$$\sum_{i=1}^{m} \mathbf{L'}^T_{ij} \mathbf{L'}_{ij} (\mathbf{P}_j - \mathbf{E}_i) = \mathbf{0}$$

Thus, $\mathbf{P}_j = \mathbf{A}^{-1}\mathbf{b}$, where

$$\mathbf{A} = \sum_{i=1}^{m} \mathbf{L'}^T_{ij} \mathbf{L'}_{ij}$$

$$\mathbf{b} = \sum_{i=1}^{m} \mathbf{L'}^T_{ij} \mathbf{L'}_{ij} \mathbf{E}_i$$

Geometrically, this solution gives the point that minimizes the sum of squared distances of $\mathbf{P}_j$ from the corresponding rays.

As the objective function is symmetrical in $\mathbf{P}_j$ and $\mathbf{E}_i$, setting the derivative with respect to $\mathbf{E}_i$ yields the equation $\mathbf{E}_i = \mathbf{A}^{-1}\mathbf{c}$, where

$$\mathbf{c} = \sum_{j=1}^{n} \mathbf{L'}^T_{ij} \mathbf{L'}_{ij} \mathbf{P}_j$$

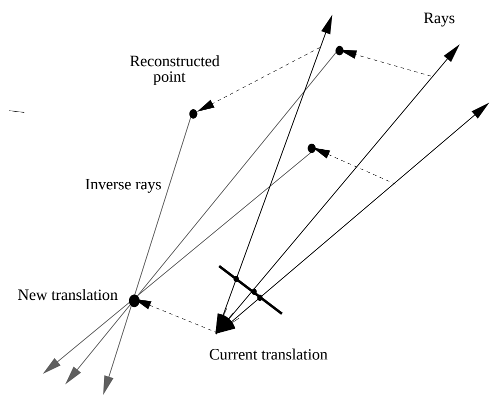


Figure 3: Translation estimate using inverse rays.

This is equivalent to finding the 3-D point that minimizes the sum-of-squared distances from the "inverse" rays through $\mathbf{P}_1 \ldots \mathbf{P}_n$ (Figure **??**).

The translation refinement algorithm alternately computes 3-D positions and camera translation estimates using the equations given above[3]. Convergence in the algorithm is detected by little change in the objective function.

3.2 Rotations

The first step in an optimization involving unknown rotations is to choose a representation for expressing rotations. A variety of representations are in use: orthonormal matrices, quaternions, Euler angles, etc. [**?**]. Each of these representations has its own advantages and disadvantages; the most appropriate representation depends on the application (e.g., quaternions provide closed form solutions for absolute orientation [**?**]). For this optimization, we chose to use Euler angles, i.e., rotation is represented by three rotations about the coordinate axes. This has the advantage that no additional constraints are needed to ensure rotational properties, in contrast to the orthonormality constraint for 3×3 matrices or the unit length constraint for quaternions. This allows use of simple (unconstrained) non-linear optimization methods such as the Newton-Raphson method [**?**] to solve for the rotation parameters.

Rotations are represented as:

$$\mathbf{R}^x(r_i)\mathbf{R}^y(s_i)\mathbf{R}^z(t_i)$$

where r_i, s_i, t_i are the Euler angles, and $\mathbf{R}^{\{x,y,z\}}$ are 3×3 matrices representing rotations about the coordinate axis. For example, $\mathbf{R}^z(\theta)$ is the matrix:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We use the iterative Newton-Raphson method using the gradient (a vector formed by the first partial derivatives) and Hessian (a matrix formed by the second partial derivatives) of the objective function to solve for the camera rotations [**?**]. Given initial estimates of r, s, t for some camera i (subscripts are omitted for clarity), increments $\Delta r, \Delta s, \Delta t$ are defined by the gradient and the Hessian:

[3]The solution is valid only up to a rigid (rotation, translation, uniform scaling) transformation. The "correct" transformation can be obtained by fixing the values of some three points in absolute coordinates.

$$\begin{bmatrix} \frac{\partial^2 O}{\partial r^2} & \frac{\partial^2 O}{\partial r \partial s} & \frac{\partial^2 O}{\partial r \partial t} \\ \frac{\partial^2 O}{\partial r \partial s} & \frac{\partial^2 O}{\partial s^2} & \frac{\partial^2 O}{\partial s \partial t} \\ \frac{\partial^2 O}{\partial r \partial t} & \frac{\partial^2 O}{\partial s \partial t} & \frac{\partial^2 O}{\partial t^2} \end{bmatrix} \begin{bmatrix} \Delta r \\ \Delta s \\ \Delta t \end{bmatrix} = \begin{bmatrix} -\frac{\partial O}{\partial r} \\ -\frac{\partial O}{\partial s} \\ -\frac{\partial O}{\partial t} \end{bmatrix}$$

The partial derivatives are obtained by symbolically differentiating the objective function with respect to r, s, t and evaluating the expressions using the current values of r, s, t. Some of the partial derivative expressions are listed in the appendix.

Given the current rotation in terms of r, s, t, the rotation refinement algorithm evaluates the partial derivative expressions and computes $\Delta r, \Delta s, \Delta t$. The new rotations are used to update the 3-D positions of the reconstructed points, and this process is repeated until convergence.

4 Results

Figure **??** shows snapshots of our incidence counting algorithm in action. We used a textured 3-D model of Technology Square (the building complex housing the MIT Laboratory for Computer Science) to generate input for this experiment. Point features necessary for the algorithm are extracted using the Canny edge detector [**?**]. The results show that the algorithm recovers fairly accurate 3-D structure from as little as three images.

Figure **??** shows the pose refinement algorithm in action. Data for this problem was generated by choosing $n = 20$ random points in a box containing $m = 10$ cameras, and perturbing the translations or rotations by a few percent. The results show that the algorithm is able to robustly recover the original camera positions from the perturbed estimates.

5 Conclusion

We presented the incidence counting algorithm that identifies matches using only the geometric constraints implied by camera pose. The algorithm performs fairly well for synthetic images and camera pose, but more experiments on real data are needed to fully evaluate its efficacy.

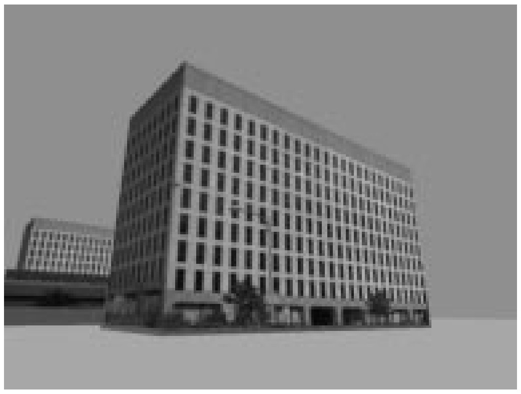
We also presented a direct 3-D algorithm to refine camera pose estimates given correspondences. Our algorithm operates directly in 3-D and can easily incorporate matches across hundreds or thousands of images. Results of this algorithm on synthetic data (random 3-D points, perturbed camera poses) were presented in this paper; we plan to experiment with real data when our pose-instrumented platform is operational. Also, we plan to investigate the convergence rates of the pose-refinement algorithms.

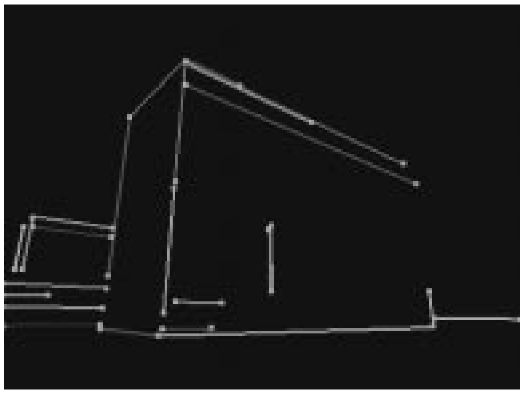
References

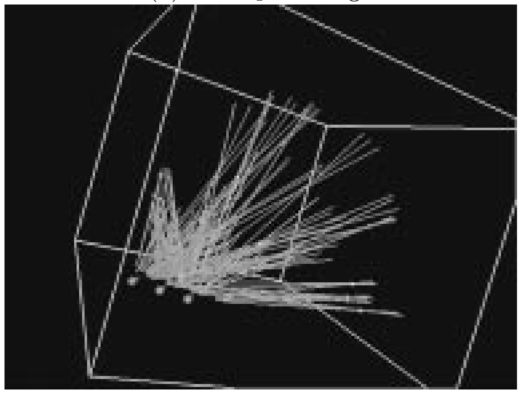
[1] Bentley, J. Multidimensional binary search trees used for associative searching. *Communications of the ACM 18* (1975), 509–517.

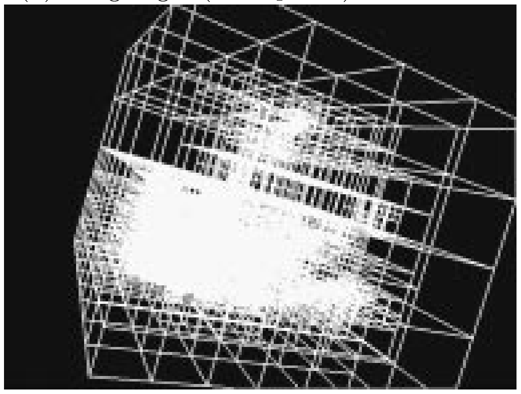
[2] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. *Computer Graphics, Principles and Practice, Second Edition.* Addison-Wesley, Reading, Massachusetts, 1990.

[3] Samet, H. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, 1990.


(a) An input image


(b) Long edges (> 10 pixels) and vertices


(c) Extruded rays


(b) Octree

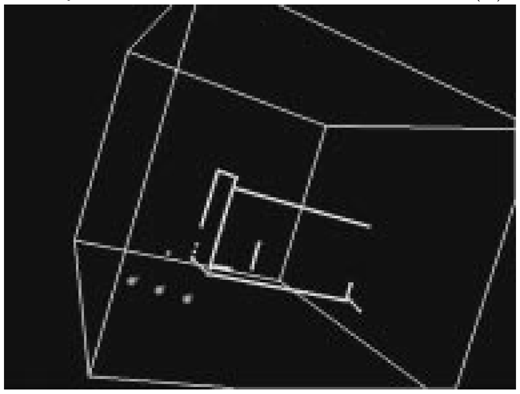
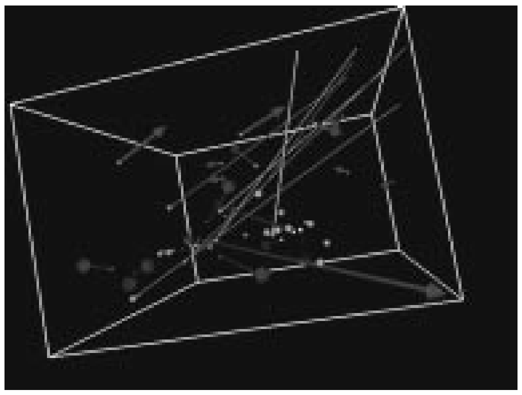
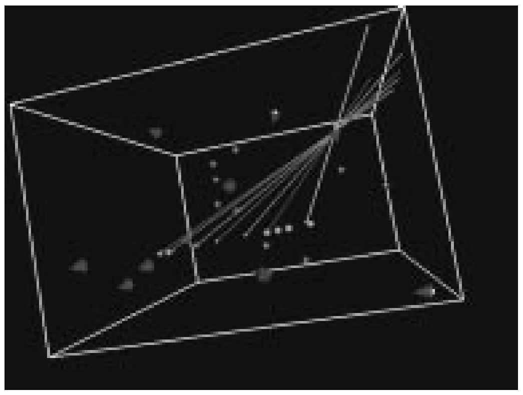
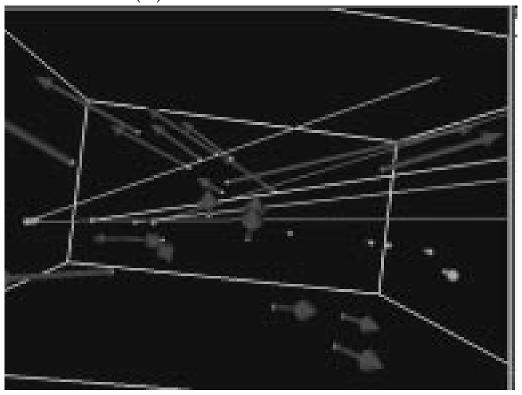

(e) Reconstructed geometry

Figure 4: Incidence counting on three synthetic images.


(a) Initial translations


(b) Final translations


(a) Initial rotations

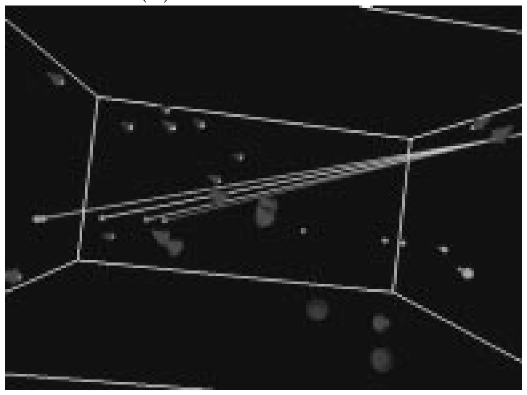

(b) Final rotations

Figure 5: Pose refinement on synthetic data. Note that rays do not pass through the initial reconstruction, but they do so after refinement. Also, the arrows connecting computed 3-D positions to original 3-D positions have become very short, indicating convergence to the original 3-D values.

A Rotational Partial Derivatives

We only list the partial derivatives with respect to t; the expressions for r and s are similar. Let,

$$\begin{aligned}
\mathbf{S}^z(\theta) &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
\mathbf{D}_j &= \mathbf{P}_j - \mathbf{E} \\
\mathbf{V}_j &= \mathbf{D}_j \times (\mathbf{R}^x(r)\mathbf{R}^y(s)\mathbf{R}^z(t)\mathbf{v}_{ij}) \\
\mathbf{V}_j^t &= \mathbf{D}_j \times (\mathbf{R}^x(r)\mathbf{R}^y(s)\mathbf{S}^z(t+\frac{\pi}{2})\mathbf{v}_{ij}) \\
\mathbf{V}_j^{rt} &= \mathbf{D}_j \times (\mathbf{S}^x(r+\frac{\pi}{2})\mathbf{R}^y(s)\mathbf{S}^z(t+\frac{\pi}{2})\mathbf{v}_{ij}) \\
\mathbf{V}_j^{tt} &= \mathbf{D}_j \times (\mathbf{R}^x(r)\mathbf{R}^y(s)\mathbf{S}^z(t+\pi)\mathbf{v}_{ij})
\end{aligned}$$

Then,

$$\begin{aligned}
\frac{\partial O}{\partial t} &= \sum_{j=1}^{n} \mathbf{V}_j . \mathbf{V}_j^t \\
\frac{\partial^2 O}{\partial t^2} &= 2\sum_{j=1}^{n} \mathbf{V}_j^t . \mathbf{V}_j^t + 2\sum_{j=1}^{n} \mathbf{V}_j . \mathbf{V}_j^{tt}
\end{aligned}$$