Estimating and Adapting to Registration Errors in Augmented Reality Systems

Blair MacIntyre
Enylton Machado Coelho

GVU Center, College of Computing
Georgia Institute of Technology
Atlanta, GA, USA
{*blair, machado*}*@cc.gatech.edu*

Simon J. Julier
ITT Advanced Engineering Systems
Virtual Reality Laboratory
Advanced Information Technology
Naval Research Laboratory
Washington DC
julier@ait.nrl.navy.mil

Abstract

All augmented reality (AR) systems must deal with registration errors. While most AR systems attempt to minimize registration errors through careful calibration, registration errors can never be completely eliminated in any realistic system. In this paper, we describe a robust and efficient statistical method for estimating registration errors. Our method generates probabilistic error estimates for points in the world, in either 3D world coordinates or 2D screen coordinates. We present a number of examples illustrating how registration error estimates can be used in AR interfaces, and describe a method for estimating registration errors of objects based on the expansion and contraction of their 2D convex hulls.

1 Introduction

For the past decade, we have been developing *augmented reality* (AR) systems that use see-through head-worn displays to overlay computer generated visual media on a user's view of the physical world (e.g., the KARMA system [4]). A successful AR system must be able to accurately *register* the computer generated visual media with the real world. If the graphics are not properly registered, the result will be distracting and annoying (at best), or misleading (at worst). For example, an AR maintenance system, such as KARMA, might need to identify a resistor on a circuit board by drawing an arrow which points to that resistor; the arrow should point at the correct resistor and not at some other component near it.

To achieve accurate registration, an AR system needs to know the geometric relationship between the user's eyes, the display(s) and the objects in the world. Careful measurement of these geometrical relationships will eliminate some of these registration errors, but it is usually not possible to know these relationships with absolute precision. For example, the user's location in the world (and thus the relationship between their eyes and and the objects in the world) is typically measured by some sort of tracking system, which introduces geometric errors. These errors are caused both by sensor inaccuracy, and by the unavoidable delay between sampling a sensor and modifying the display. While very good registration has been demonstrated in well-controlled laboratory experiments (e.g., [1, 8, 16]), and in time-delayed entertainment applications (e.g., the "1st and Ten" system that superimposes a virtual first down line on live football broadcasts [13]), perfect (pixel-level) registration will not be attained in real-time AR systems in the near future, especially when using optical see-through displays, even in well-controlled laboratory environments. Furthermore, errors are more difficult to avoid when we move out of the lab, and create mobile AR systems where the user roams over a large, imperfectly modeled environment.

We believe that the effects of registration errors can be mitigated through the development of adaptive user interfaces that tailor the information display as a function of registration errors. For example, if a mobile AR system needs to highlight a window on a building, but the user's position and orientation are not known precisely, it could render the highlight in a way that guarantees the window will fall within the highlighted area, as illustrated in Figure 1. Notice that, while such illustrations avoid misleading the viewer by highlighting the wrong window, the user may need additional information to disambiguate the resulting display (e.g., there are two windows within the upper-left window's highlighted area). To avoid this ambiguity, the system should change the highlighting techniques as a function of the error estimate. For example, in [12], we introduced the concept of a *level-of-error* (LOE) rendering technique, analogous to the popular *level-of-detail* (LOD) techniques used in traditional 3D graphics and VR. Unlike

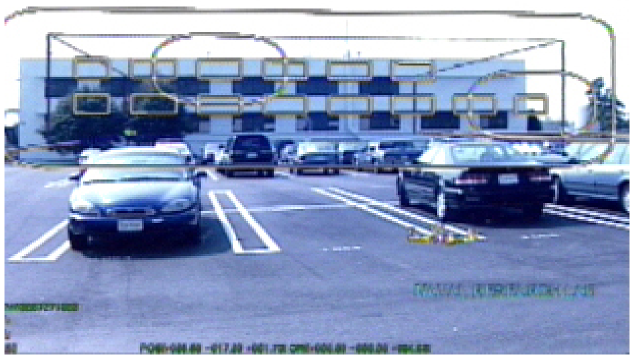


Figure 1. An example of the use of registration error estimation. In this image, the system highlights a building and two of its windows by outlining the area of the screen the objects could occupy. Each outline was created by growing the convex hull based on an estimate of the registration error. The simple, misregistered computer model of the building is shown for clarity. While the model is not registered with the world, the intended objects fall within the highlighted areas of the screen.

LOD techniques, which change how an object is displayed based on its distance from the viewer, the LOE technique changes how an object is displayed based on a quantitative estimate of how accurately the object can be registered with the world. In Figure 1, a programmer might use LOE to switch between precisely highlighting the window, highlighting it with a elliptical region (as shown), and adding a textual description when the registration becomes sufficiently poor. Other researchers have also explored how to build AR interfaces that adapt their presentation based on the accuracy of the trackers (e.g., [5]), although they do not compute a quantitative estimate of registration error, instead relying on qualitative heuristics based on the programmer's knowledge of the trackers being used.

For the purpose of the work reported in this paper, the causes of registration errors can be divided into two groups: *spatial errors* (such as those caused by inaccurate measurement of the geometric relationships in the system) and *temporal errors* (such as those caused by system latency). In [12], we estimated the registration errors caused by spatial errors in the tracking system using a straightforward metric based on the manufacturer-reported error range for a single tracker. In this paper, we presents a general, robust and efficient statistical method for estimating registration errors, and show how these error estimates can be used in AR interfaces. We currently estimate the effects of spatial errors arising from tracking and head mounted display calibration, but do not account for temporal errors (see Section 8). We assume that spatial errors may vary through time, and with the user's context, so our method is fast enough to be used to continuously estimate dynamically changing registration errors.

The structure of this paper is as follows. In the next two sections, we present a variety of uses for registration error estimates in AR interfaces, and discuss related work. In Section 4, we summarize our framework for estimating registration error bounds in an AR system. The first part of the framework is a statistical method to estimate the error bounds of a single 3D point as a probability distribution in 3D world coordinates or 2D screen coordinates; this method is described in Section 5. The second part of the framework is a method for estimating the registration error for an object, by aggregating the probability distributions of its vertices. In Section 6, a fast method of approximating these regions using 2D convex hulls is described. In Section 7, we summarize some important details of the testbed we used to test the algorithms. Future work is described in Section 8, and conclusions are drawn in Section 9.

2 Uses of Registration Error Estimates

This work is motivated by the need to know where physical objects appear on the display in a mobile AR system. In particular, we want to identify three possible regions: where a single point might appear on the display, where an object might appear on the display, and what part of the display definitely contains part of an object. In Section 5, we describe a statistical method for computing the region of the screen that a single 3D point might intersect. This region appears as an ellipse, as shown in Figure 2. The two remaining regions are illustrated in Figure 1. In Section 6, we describe how to find these two regions for convex objects. Intuitively, we can find the region of the screen that *might* be occupied by an object by expanding the object based on the its vertices. Similarly, we can find the region of the screen that is *definitely* occupied by an object by contracting the objects based on these error regions. Examples of specific uses of these regions are:

- A compelling method of highlighting an object in AR is to render a silhouette around the object. For convex (or nearly convex) objects, we can use the convex hull as a reasonable approximation to the object silhouette. By using the expanded hull, we end up with a highlight that encloses the region where the object is, even if it does not precisely highlight the object (see Figures 1 and 2).

- When an object is labelled, the label should remain on-screen as the user moves if part of the object is visible,

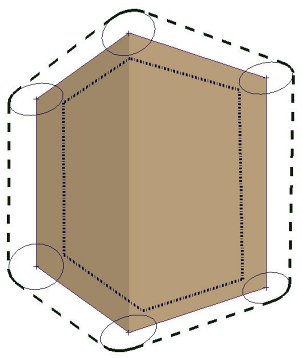


Figure 2. A simple 3D object, with error estimate ellipses displayed around its vertices. The convex hull of the object can be expanded (dashed outline) and shrunk (dotted outline) based on the elliptical error estimates.

as is done by the labeling system in [4] (which placed the labels at the top and bottom of the screen, and connected them to the physical objects with a line). If we clip the compressed hull (such as the one shown in Figure 1) to the screen, we have the region of the screen that is guaranteed to contain the object. Therefore, if we point to some point inside that region, we can be sure that the line connects the label to the object. Similarly, if we want the label to appear on the object, as was done by the rendering engine in [3], we can place the label in this region. This is especially useful for objects that occupy a large field of view, such as the buildings in [3], since the best place for the label will change dramatically as the user moves their head.

- In the past, we have used gaze-based selection as an input technique for mobile AR systems (e.g., [3]). We have usually denoted an object (even a large one, such as a building) by a single point, or by its label. Knowing the region on the screen occupied by the object would let us more accurately approximate what the user is looking at. A combination of all three object regions (normal, contracted and expanded) could be used, depending on the density of the objects and the specific interaction techniques. For example, potentially ambiguous registration of two closely spaced objects might occur if their expanded error hulls overlap.

3 Related Work

A number of authors have attempted to quantify and deal with the effects of registration errors (e.g., [1, 6, 8, 10, 16]). Probably the best-known and most comprehensive analysis was conducted by Holloway, who studied the causes of registration errors in a surgical planning system [7]. His analysis accounted for a range of parameters, including system latency, optical distortion in the head-worn display, calibration and measurement errors, and dynamically changing tracker errors. The errors were expressed in terms of angular error, lateral error and depth error. He demonstrated that the hardest source of error to eliminate in a carefully engineered AR system was system latency, where a delay of a single millisecond could cause transient registration errors of the order of 1mm when the user or tracked objects are in motion.

Holloway's analysis provides a useful foundation for identifying the causes of modelling errors, but is not well suited for generating the error estimates needed by adaptive display algorithms. It does not present a single, comprehensive computational model which combines all of the components together. Furthermore, the analysis only discusses points, rather than objects. Objects are especially relevant in outdoor situations, where the size of an augmented object (such as a building) can occupy a significant portion (or even all) of the display.

Estimating error bounds, especially in vision-based tracking systems, is not a new idea. For example, Hoff used error estimates as the basis for fusing multiple sensors [6]. The contribution of our work is twofold. First, we describe a more general, robust and efficient method for estimating registration error. Unlike Hoff's method, for example, ours can take into account errors that affect any parameter in the entire viewing model (including the parameters to the projection operators caused by calibration errors), as long as those errors can be modelled as probability distributions. Second, we illustrate how to use these error estimates to improve the AR interface *in the presence of error*, rather than only focusing on minimizing the error.

4 Method Overview

There are a number of issues that must be addressed by any method for estimating registration errors in an AR system:

- The effects of these errors are a function of the distance of an object from the observer: as the object moves further away the effects of position-related errors diminish, while angular errors do not.

- The underlying causes of registration errors are time varying, and can depend on the user's current location or activity. For example, the noise of magnetic trackers increases with the distance from the transmitter. The accuracy of systems which rely on beacons (either passive or active) depends on the number of beacons which are visible, and their configuration. These issues affect systems ranging from GPS (each satellite is, in effect, a moving active beacon) to sonar-based navigation systems (such as the InterSense IS900) to optical systems (such as the Vicom). Another property of these systems is that the position and orientation errors are frequently correlated with one another and the errors cannot be correctly modeled independently.

- The causes of some errors may be well understood, but still be difficult or impossible to correct or to even quantify; for example, the presence of ferrous metal or magnetic fields can radically affect the direction reported by a magnetic compass, but the magnitude of the error cannot easily be detected [16].

Our framework for registration errors consists of two steps: estimating the registration error of individual 3D points, and aggregating these estimates for objects as follows:

- For each object to be augmented, construct its list of vertices.
- Calculate, for each vertex, the statistical properties of the registration errors. In particular, we calculate the mean and covariance of the registration errors for each point.
- Use the covariance ellipses for each vertex to expand or contract the object. When working in 2D screen space, we can use the 2D convex hull of the object instead of the object itself.

This method has several advantages. First, the registration errors are assessed in real time. Therefore, the dynamic effects of object position and time varying tracker errors can be incorporated. Second, robust error terms can be used. Third, the metric can be used to assess the impact of registration errors on non-point like objects. Fourth, the approach can be used to construct 3D error volumes or 2D error areas in the viewport. The latter is the main focus of the techniques illustrated in this paper. Finally, the 2D error measure has a direct intuitive appeal — it is the area of the viewport which is occupied by the registration errors.

The next two sections describe the steps for estimating the registration errors and constructing the convex hulls.

5 Statistical Registration Error Estimation

The effects of registration errors can be estimated by observing their effects on the projection equations. Because the analysis is fundamentally time varying, all the following quantities are referenced with respect to the discrete time index k. Each time step can, for example, refer to an individual frame. It is not necessary for each time step to be of equal length.

Consider the problem of determining the pixel coordinate $\mathbf{y}(k)$ of a point $\mathbf{p}(k)$ in world coordinates. The projection can be broken into two steps — transforming from world coordinates to head coordinates (with coordinates $\mathbf{p}'(k)$), and then applying the perspective transformation to project the point to the view plane. The transformation from $\mathbf{p}(k)$ to $\mathbf{p}'(k)$ is governed by the (homogeneous) model transformation matrix $\mathbf{M}(k)$,

$$\mathbf{p}'(k) = \mathbf{M}(k)\,\mathbf{p}(k)\,. \tag{1}$$

$\mathbf{M}(k)$ is a composite transformation that includes the inverses of the transformation matrices formed by the sensor readings. Therefore, tracking errors contribute registration errors through an erroneous value of $\mathbf{M}(k)$. These errors can be both spatial and temporal.

For simplicity, assume that the model transformation matrix is composed of t separate transformations derived from t different tracker readings (any fixed transformations between the tracker transformations can be collapsed into the tracker transformations, without loss of generality). Multiple nested tracker transformations can arise when both the user and an object in the world are tracked, or in mobile AR systems when two different sensing modalities can be used to estimate position (a GPS) and orientation (an inertial sensor). In these situations, the model transformation matrix is given by cascading the individual transformations together,

$$\mathbf{M}(k) = \mathbf{M}_1(\mathbf{x}_1(k)) \times \ldots \times \mathbf{M}_t(\mathbf{x}_t(k))\,.$$

where $\mathbf{x}_i(k)$ is the reading from the ith tracker. Such a situation arises in the mobile augmented reality system described in Section 7 — the position and orientation trackers are two physically different devices which yield a different set of measurements with different update rates and noise characteristics.

To calculate $\mathbf{y}(k)$, a perspective transformation is applied to $\mathbf{p}'(k)$. The projection matrix, $\mathbf{P}$, is given by:

$$\mathbf{P}(k) = \begin{bmatrix} \frac{f}{a} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} & \frac{2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{2}$$

where $f = \cot[\frac{\text{fov}}{2}]$, a is the aspect ratio and z_{near} and z_{far} are the near and far clipping distances. We define

$$\mathbf{y}'(k) = \mathbf{P}(k)\,\mathbf{p}'(k)\,. \tag{3}$$

Therefore, the pixel coordinate on the viewport is

$$\mathbf{y}(k) = \mathbf{y}_0(k) + \frac{1}{2y'_4}\begin{bmatrix} w(y'_1 + y'_4) \\ h(y'_2 + y'_4) \end{bmatrix} \tag{4}$$

where $\mathbf{y}_0(k)$ is the top left corner of the viewport, w and h are the width and height of the viewport, and y'_i is the ith coefficient in $\mathbf{y}'(k)$.

Combining Equations 1, 3 and 4, the projection operator can be written as

$$\mathbf{y}(k) = \mathbf{h}[\mathbf{p}(k), \mathbf{x}(k), \mathbf{u}(k)] \tag{5}$$

where $\mathbf{x}(k)$ is the n-dimensional tracker reading vector (including any parameters that have error associated with them) and $\mathbf{u}(k)$ is the control input vector (that includes the remaining parameters, which have no error distribution associated with them, such as the various control or calibration parameters that define the projection matrix). Because the tracker reading is corrupted by noise, it is modeled as a random variable with mean $\hat{\mathbf{x}}(k)$ and covariance $\mathbf{X}(k)$. We require $[\hat{\mathbf{x}}(k), \mathbf{X}(k)]$ to be a conservative estimate of $\mathbf{x}(k)$, but do not require it to satisfy other constraints (such as being zero-mean, uncorrelated, or Gaussian distributed).

The problem of calculating the error bounds can be stated as follows. For a point $\mathbf{p}(k)$ with tracker measurement $[\hat{\mathbf{x}}(k), \mathbf{X}(k)]$, calculate the mean and covariance of $\mathbf{y}(k)$, $[\hat{\mathbf{y}}(k), \mathbf{Y}(k)]$.

5.1 Limitations of Linearization

Because Equations 3 and 4 are nonlinear, the conventional assumption is to utilize *linearization*. Taking the Taylor series expansion and truncating the series after the first term, it can be shown that the mean and covariance are

$$\begin{aligned} \hat{\mathbf{y}}(k) &= \mathbf{h}[\mathbf{p}(k), \hat{\mathbf{x}}(k), \mathbf{u}(k)] && (6) \\ \mathbf{Y}(k) &= \nabla_x \mathbf{h}\, \mathbf{X}(k)\, \nabla_x^T \mathbf{h} && (7) \end{aligned}$$

where $\nabla_x \mathbf{h}$ and $\nabla_v \mathbf{h}$ are the Jacobians of $\mathbf{h}[\cdot,\cdot,\cdot]$ with respect to $\mathbf{x}(k)$ and $\mathbf{v}(k)$, evaluated at $\mathbf{x}(k) = \hat{\mathbf{x}}(k)$ and $\mathbf{v}(k) = \mathbf{0}$.

However, there are two well-known problems with this approach. First, deriving the expression for the Jacobian matrix can be cumbersome, especially if the system is high order or nonlinear. Second, the errors due to the first order approximation can be significant. For example in [11], Lerro and Bar-Shalom illustrated that linearization errors in the transformation between polar and Cartesian coordinates can be significant at angular errors of only a few degrees. Since we are concerned with the problem of registration errors with potentially large angular errors, these difficulties need to be addressed. A number of authors have developed filters which are capable of propagating information at a higher order than the extended Kalman Filter (EKF). However, higher order filters require more information about the transformation equations (such as their higher order derivatives), and these difficulties compound the implementation problems. To overcome these difficulties, we utilize the *unscented transformation*.

5.2 The Unscented Transformation

The unscented transformation works on the principle that *it is easier to approximate a Gaussian distribution than it is to approximate an arbitrary nonlinear function or transformation* [9, 15]. A set of points (or *sigma points*) are chosen so that their sample mean and sample covariance are $\hat{\mathbf{x}}(k)$ and $\mathbf{X}(k)$. The nonlinear function is computing using each sigma point, and applied in turn to the point $\mathbf{p}(k)$ to yield a cloud of transformed points; the statistics of the transformed points $\hat{\mathbf{y}}(k)$ and $\mathbf{Y}(k)$ can then be directly computed.

The n-dimensional random variable $\mathbf{x}$ with mean $\hat{\mathbf{x}}(k)$ and covariance $\mathbf{X}(k)$ is approximated by $2n+1$ weighted points given by

$$\begin{aligned} \boldsymbol{\mathcal{X}}^0 &= \hat{\mathbf{x}}(k) & W_0 &= \kappa/(n+\kappa) \\ \boldsymbol{\mathcal{X}}^i &= \hat{\mathbf{x}}(k) + \left(\sqrt{(n+\kappa)\mathbf{X}(k)}\right)_i & W_i &= 1/2(n+\kappa) \\ \boldsymbol{\mathcal{X}}^{i+n} &= \hat{\mathbf{x}}(k) - \left(\sqrt{(n+\kappa)\mathbf{X}(k)}\right)_i & W_{i+n} &= 1/2(n+\kappa) \end{aligned} \tag{8}$$

where κ is a real scaling factor, $\left(\sqrt{(n+\kappa)\mathbf{X}(k)}\right)_i$ is the ith row or column of the matrix square root of $(n+\kappa)\mathbf{X}(k)$ (which can be calculated from the Cholesky Decomposition), and W_i is the weight which is associated with the ith point. Each $\boldsymbol{\mathcal{X}}^i$ can be thought of as a perturbation of the tracker reading $\mathbf{x}(k)$ that falls on the boundary of the estimated error region defined by the covariance $\mathbf{X}(k)$. The transformation procedure is as follows:

1. Instantiate the point $\mathbf{p}(k)$ through the function once for each $\boldsymbol{\mathcal{X}}^i$, to yield the set of transformed sigma points,
$$\boldsymbol{\mathcal{Y}}^i = \mathbf{h}\left[\mathbf{p}(k), \boldsymbol{\mathcal{X}}^i, \mathbf{u}(k)\right].$$
2. The mean is given by the weighted average of the transformed points,
$$\hat{\mathbf{y}}(k) = \sum_{i=0}^{2n} W_i \boldsymbol{\mathcal{Y}}^i. \tag{9}$$
3. The covariance is the weighted outer product of the transformed points,
$$\mathbf{Y}(k) = \sum_{i=0}^{2n} W_i \left\{\boldsymbol{\mathcal{Y}}^i - \hat{\mathbf{y}}(k)\right\}\left\{\boldsymbol{\mathcal{Y}}^i - \hat{\mathbf{y}}(k)\right\}^T. \tag{10}$$

In this application, the optimal choice of κ is $\kappa = 3 - n$ [9].

5.3 Advantages of The Unscented Transformation

The unscented transform has three important advantages. First, it is easy to implement and can be readily extended to include additional error terms. The projection operator $\mathbf{h}\left[\mathbf{p}(k), \boldsymbol{\mathcal{X}}^i, \mathbf{u}(k)\right]$ can be treated as a "black box" — given an input set of parameters, the function need only generate the output results. Therefore, $\mathbf{h}[\cdot,\cdot,\cdot]$ can be implemented in any appropriate manner (for example, OpenGL's matrix routines could be used, or the non-decomposable projection matrix returned by a calibration procedure such as [14] could be used). Furthermore, additional error terms (caused, for example, by rendering latencies, or modeling errors in the graphics scene) can be incorporated in a simple and uniform manner.

Second, it can be proved that this algorithm is more accurate than linearization [9]. The reason is that the UT precisely captures the first two moments of the distribution of $\mathbf{x}(k)$. Therefore, the mean and covariance terms precisely capture the second order terms (linearization only captures the first order term in the mean). Furthermore, the κ parameter allows the transformation to exploit partial fourth order information, without any increase in computational costs. Third, by changing the point set, it is possible to extend or change the statistical information which is propagated. Recent work has also proved that, for n dimensions, only $n+2$ sigma points are required.

6 Convex Hulls

The *convex hull* of a set of points S is the smallest convex region R that contains the points in S. The convex hull of Q is denoted as $CH(Q)$. A planar convex hull can be computed efficiently in $O(n \log n)$ time, using well known algorithms such as the Jarvis's March algorithm (Gift Wrapping) [2].

We generate the convex hull of a 3D object by first projecting its vertices into 2D screen coordinates, and then taking the convex hull of the 2D points. The error propagation algorithm described above can be used to generate an error estimate for each vertex as a 2D ellipse in screen coordinates. We use these ellipses to *expand* (grow) or *contract* (shrink) the 2D convex hull, as illustrated in Figure 2 (these operations are similar to the morphological operations of *dilation* and *erosion*).

6.1 Expansion and Contraction

Expansion of the set A by the set B, denoted by $A \oplus B$, is defined by:

$$A \oplus B = \{x| \ \ x \in B, \ B_c \subseteq A\},$$

where B_c is the centroid of B.

The contraction of a set A by a set B is denoted by $A \ominus B$ and is the dual of the expansion in the sense that it is the complement of the expansion by B of the complement of a

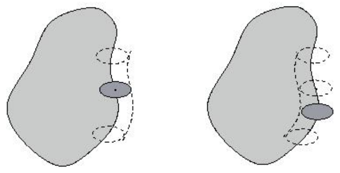


Figure 3. Expansion adds all neighboring points at distance smaller then the radius of the kernel, in a given direction. Contraction removes all points closer then the radius of the kernel.

set A:

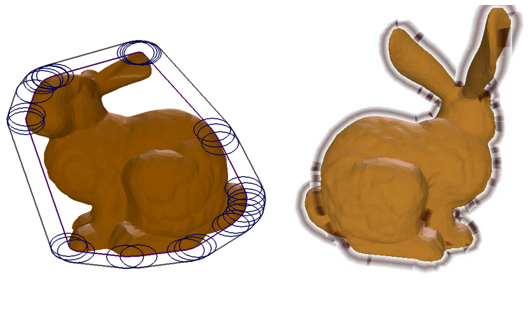
$$A \ominus B = [A^c \oplus B]^c$$

Both operations can be visualized by positioning the centroid of the smaller set, called the *kernel*, at every point of the border of the larger set (Figure 3). Expansion adds the region swept by the kernel to the original region. Contraction removes the region swept by the kernel from the original set.

For our purposes, set $A = CH(V)$ is the polygonal convex hull of the projected points V of the object. Since the error ellipses can be different for each vertex in V, the kernel varies as it is swept along the boundary of the hull. In our algorithms, we implicitly assume that the error bounds should be interpolated along the edge joining any two vertices. Since we are interested in obtaining a polygonal representation of expanded and contracted hulls, we use simple algorithms that operate on polygonal approximations of the error ellipses, rather than on the implicit form of the ellipses. The two operations are implemented as follows:

expansion – The solution to this operator is the convex hull $CH(VE)$, where VE is the union of the points on the boundaries of the error ellipses surrounding each vertex in $CH(V)$ (as shown in Figure 2). Using polygonal approximations to the error ellipses, the algorithm computes the convex hull of the union of the vertices of these polygons. Intuitively, the precise representation of $CH(VE)$ includes all the points on a collection of curved segments, where each ellipse contributes one curved segment. For each ellipse, there is an exterior tangent line between the ellipse and each of the two neighboring ellipses. The curved segment for an ellipse is the boundary of the ellipse between the intersection points of the ellipse with these two tangent lines. We introduce slight error by computing the convex hull of the approximated error ellipses, rather than determining the exact representation and then approximating the curved line segments with polylines.

contraction – To contract $CH(V)$, we first determine the interior tangent lines between each pair of adjacent error ellipses. Then, for each ellipse, the intersection of these two tangent lines is computed. The solution to the contraction operator is the convex hull $CH(VC)$, where VC is the set of these intersection points (as shown in Figure 2). The tangent line between any two error ellipses can be computed straightforwardly. First, the joint convex hull of the polygon approximations of the two ellipses is computed. By maintaining vertex order information, we can immediately find the edge of the convex hull that lies along the tangent line. This is the precise representation of $CH(VC)$ (i.e., $CH(VC)$ contains no curved segments), although we introduce slight error by computing the tangent line between the approximated ellipses rather than determining the exact tangent lines.


(a) The convex hull can be inadequate for rendering an object silhouette on a non-convex object, such as the rabbit shown here.

(b) Using the error bounds at each vertex to grow the silhouette yields better results.

Figure 4. Using Error Estimates with Concave Objects

6.2 Discussion

We use the convex hull of the vertices of an object to approximate the silhouette of that object for two reasons. First, convex hulls can be computed quickly, while there is no fast algorithm to compute the external contour of an object from an arbitrary view point. Second, even if we could compute the silhouette quickly, most of the uses we envision for this information would either not benefit significantly from a more accurate (non-convex) silhouette, or the algorithms to implement them would become significantly more complex when given an non-convex polygon.

However, an exact contour is useful when we want to render the silhouette of a complex object, such as the rabbit in Figure 4. For other uses (see Section 2), we could get better results by having the programmer break down a concave object into a collection of convex parts, and use iterative algorithms on these multiple parts. For example, if we are using the compressed hull to find a good place to label the rabbit, we could break the model into three parts that are closer to being convex (e.g., the body and two ears), and simply try to use each part in turn to find possible locations for the labels. This approach also has the advantage that we can prioritize the part of the object we wish to label.

7 Experimental Results

The framework was implemented and tested on a mobile AR system. The system is composed of 6DOF trackers (an

Ashtech GG Surveyor real-time–kinematic GPS for position, an InterSense IS300Pro for orientation), a see-through head-worn display (Sony LDI-D100B Glasstron), a wireless network (a WaveLan), and a wearable computer with 3D hardware graphics acceleration. Using this system, a user is able to walk around an outdoor environment and see information such as the names and locations of buildings as well as "fine-grained" details such as individual windows and doors. Our framework is especially well suited for this type of environment because most objects are "box like" and are convex.

As noted in Section 4, our trackers provide limited information about the magnitude of their sensor error, so we provide conservative estimates for their error bounds. In our test system (see Figure 1), the error volumes are calculated for a static set of objects. Therefore, the only error contributions arise from the tracking errors. Implemented on a Pentium II 333MHz processor, the set of sigma point projection matrices can be calculated in approximately 46μs. The time required to project and construct the error statistics for each point is approximately 17μs. For an object consisting of 8 vertices, the error ellipses and the convex hull can be constructed in approximately 720μs.

8 Discussion and Future Work

As can be seen, the method directly handles multiple nested trackers, as well as any other errors that can be modeled as probability distributions over the input values to the coordinate system transformations. The method is also relatively fast, although the basic method can be optimized significantly in the context of a typical AR system. For example, while the set of composite transformation matrices $\mathbf{PM}\boldsymbol{\mathcal{X}}^i(k)$ will need to be recomputed when the tracker values change (just as the matrices used for rendering must be recomputed), the entire process does not need to be redone from scratch each time. Intermediate values can be cached, and for many trackers, the error model may not change from frame to frame.

Notice, also, that n is usually small for systems with only a few trackers (e.g., each tracker introduces at most six parameters with error), and that many places in the graph share the same set of error parameters $[\hat{\mathbf{x}}(k), \mathbf{X}(k)]$. If two nodes share the same error parameters, most of the computation necessary to create the $2n+1$ matrices can be reused.

If the overhead of computing error distributions becomes high, even with optimization, the distributions may be computed asynchronously (and more slowly) from the display. Only a small amount of error may be introduced by reusing $\mathbf{Y}(k)$, the covariance of the screen space error distribution, for multiple frames. Similarly, it may be sufficient to modify the trackers to update the covariance of the input errors more slowly, thus requiring less computation.

We hope to extend our system to estimate the effects of temporal errors soon. It should be possible to model temporal errors in a way that fits directly into this method, by integrating the prediction algorithms into the set of variables used to compute the locus of points. If we model the tracker report as an equation of time (with the time variable being the distance into the future of the prediction), then this time interval becomes an error term. However, we suspect that a different approach may be needed, since the effects of temporal error (e.g., system lag and tracker sensor latencies) must be estimated much more rapidly, and will change much more frequently, than other errors in the system.

9 Conclusions

This paper has described a method for real-time estimation of dynamically changing registration errors. The approach is adaptive: at any time, it takes into account the current tracking accuracy, as well as the geometric properties of objects which are to be augmented. The algorithm can be readily extended to incorporate other errors that affect the transformation matrices, and to use more detailed tracker error information if it could be obtained from the tracking hardware. We showed how error estimates for single points can be aggregated to estimate the registration error of an object, by computing what part of the screen *might* be, and what part is *definitely*, occupied by the object. Finally, we showed how these error estimates can be used in AR interfaces.

10 Acknowledgements

This work was supported by Siemens via a GVU Industrial Affiliate Grant, ONR under Grant N000140010361, and equipment and software donations from Sun Microsystems and Microsoft.

References

[1] R. Azuma and G. Bishop. Improving static and dynamic registration in an optical see-through HMD. In *Computer Graphics (Proc. ACM SIGGRAPH '94)*, Annual Conference Series, pages 197–204, Aug. 1994.

[2] M. D. Berg. *Computational Geometry: Algorithms and Applications*. Springer Verlay, 2000.

[3] S. Feiner, B. MacIntyre, T. Höllerer, and A. Webster. A Touring Machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. *Personal Technologies*, 1(4):208–217, 1997.

[4] S. Feiner, B. MacIntyre, and D. Seligmann. Knowledge-based augmented reality. *Commun. ACM*, 36(7):52–63, July 1993.

[5] T. Hoellerer, D. Hallaway, N. Tinna, and S. Feiner. Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system. In *2nd Int. Workshop on Artificial Intelligence in Mobile Systems (AIMS '01)*, August 2001.

[6] W. Hoff. Fusion of data from head-mounted and fixed sensors. In *Proceedings of the First International Workshop on Augmented Reality*, pages 167–182, 1998.

[7] R. L. Holloway. Registration Error Analysis for Augmented Reality. *Presence: Teleoperators and Virtual Environments*, 6(4):413–432, August 1997.

[8] M. Jacobs, M. A. Livingston, and A. State. Managing latency in complex augmented reality systems. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, pages 49–?54, April 1997.

[9] S. J. Julier, J. K. Uhlmann and H. F. Durrant-Whyte. A New Approach for the Nonlinear Transformation of Means and Covariances in Linear Filters. *IEEE Transactions on Automatic Control*, 5(3):477–482, March 2000.

[10] G. Klinker, T. Reicher, and B. Brugge. Distributed user tracking concepts for augmented reality applications. In *International Symposium on Augmented Reality (ISAR 2000)*, Oct 5-6 2000.

[11] D. Lerro and Y. K. Bar-Shalom. Tracking with Debiased Consistent Converted Measurements vs. EKF. *IEEE Transactions on Aerospace and Electronics Systems*, AES-29(3):1015–1022, July 1993.

[12] B. MacIntyre and E. Coelho. Adapting to dynamic registration errors using level of error (loe) filtering. In *International Symposium on Augmented Reality (ISAR 2000)*, Oct 5-6 2000.

[13] SportVision, Inc. 1st and Ten. *http://www.sportvision.com*.

[14] M. Tuceryan and N. Navab. Single point active alignment method (spaam) for optical see-through hmd calibration for ar. In *International Symposium on Augmented Reality (ISAR 2000)*, Oct 5-6 2000.

[15] J. K. Uhlmann. Simultaneous map building and localization for real time applications. Technical report, University of Oxford, 1994. Transfer thesis.

[16] S. You, U. N. R., and Azuma. Orientation tracking for outdoor augmented reality registration. *IEEE Computer Graphics and Applications*, 19:36–42, 1999.