

# High-Level Tracker Abstractions for Augmented Reality System Design

Enylton Machado Coelho and Blair MacIntyre  
GVU Center, College of Computing  
Georgia Institute of Technology  
Atlanta, GA, USA  
{machado, blair}@cc.gatech.edu

**Abstract**—In this paper we argue that because there is no ideal tracking system, the choice of which tracking technology to use ends up being the most crucial decision to be made and this in turn affects the whole application development process.

We claim that a tracker abstraction that help the application developer reason about the final result of tracker uncertainty (the registration error) instead of its source, is necessary for a new class of applications that conform to a heterogeneous tracking environment to be developed. We then introduce a method to achieve this intended result.

## I. INTRODUCTION

*Augmented reality* (AR) application design usually follows three steps: specification of the application requirements, definition of a tracking technology to be used, and application implementation [1]. Application designers either choose a convenient tracking system (e.g., one that is cheap, such as GPS, or one they already have access to), or purchase the cheapest one that most closely satisfies the application requirements. Choosing a tracking technology figures so prominently in current AR system design because there is no one, ideal tracking technology, and the ones that are available have vastly different, most often conflicting, characteristics. For example, high accuracy trackers are currently very expensive and typically only cover a small area, whereas cheaper and/or wide-area trackers typically have lower accuracy.

One implication of the need to make tradeoffs when choosing a tracking technology is that application development is often guided and limited by the underlying constraints of chosen tracking

technology. This tight binding to a specific tracker may even happen unconsciously. For example, a laboratory or company that invests tens of thousands of dollars in a specific tracking system will naturally tend to design applications to work within the constraints of the tracking system.

This approach works reasonably well for applications that naturally match the characteristics of existing trackers. For example, many trackers provide a fairly constant level of accuracy in a small fixed space, which matches the needs of applications for some medical procedures, maintenance and training tasks. However, many interesting AR applications will have different requirements at different times and locations, especially those that operate for longer periods of time over large areas. Examples include maintenance over large factory floors with many machines, military and policing applications, emergency response, wide area entertainment, and so on. Furthermore, in many of these application environments it is reasonable to assume the capabilities of the tracking system may change over time (e.g., through intentional or accidental damage to the environment), implying that the applications must adapt to changes in tracker characteristics to work as well as possible at all times.

Our research focus is on developing the capabilities to deal with multiple trackers with no *a priori* knowledge of the characteristics of those trackers, developing the algorithms necessary to estimate the registration error between the virtual and physical objects in real time, and designing program structures to help the application use the appropriate augmentation in each situation.

The rest of the paper is structured as follows: in section II we discuss the necessity and consequences of tracker abstraction. Section III discusses how registration error estimates can be computed. In section IV we present some uses of such estimates, and we conclude, in section V, with a brief description of our current and future work.

## II. ABSTRACTION LAYERS

We believe that applications with heterogeneous tracking characteristics require higher level tracker abstractions than are currently available, if they are to be properly designed, implemented, and deployed. The fundamental characteristic of these abstractions will be support for the application developer to deal with changes in tracking quality. We also believe that such abstractions will simplify the creation of all AR applications, by making them less dependent on the characteristics of the currently available tracking technology, and thus able to adapt to newer technologies as they become available. From a software engineering viewpoint, properly designed abstractions will encourage application developers to focus on how they want their applications to behave under various tracking conditions, and make explicit their assumptions about the affects of tracker accuracy on their application.

We view tracker abstractions and AR applications analogously to window systems and desktop applications. Prior to the creation of general purpose, hardware independent windows systems (such as X Windows), application developers built applications that were intimately tied to specific pieces of graphics hardware. While this allowed developers to squeeze the maximum amount of performance out of a specific set of hardware, it also locked the application into that hardware and intertwined the application requirements and design choices with the characteristics of the hardware, often blurring the reason behind specific choices and behaviors (i.e.. does the application behave this way because we want it to, or because the hardware forced it to?). The advent of systems such as X11<sup>1</sup> (and their

associated libraries and toolkits) allowed applications to be built independently of specific hardware. As more powerful hardware became available, the applications could use it, often without being re-compiled, especially if they were written in a way that reacted to the speed and power of the hardware. And porting to new platforms that support the same window system was often trivial.

And so we believe it will be for AR applications that are written on top of properly designed tracking abstractions. By separating the application from the specifics of individual trackers, applications will be better able to leverage new tracking technologies as they become available, can adapt to changing tracking conditions while they are running, and can be maintained and understood more easily because the tracking assumptions are more clearly identified in the application.

### A. Device Independence

Many libraries have been built over the years for VR and AR system to provide a basic level of tracker abstraction (e.g., VRPN from UNC Chapel Hill is a recent example). The goal of these libraries is typically to allow an application to receive tracker reports in a standardized format, independent of which tracker is being used to generate the readings, allowing applications to ignore the low-level details of communicating with the different tracking technologies.

We believe this lowest level abstraction should be enhanced in a number of ways. First, the tracking system should provide details about the characteristics of the tracker (such as its range of operation, number of sensors, etc.). Second, the system should provide details about the estimated accuracy of each tracking measurement. Together, this information will allow developers to create applications that adapt to changing tracking characteristics. While it could be argued that precise estimates of the accuracy of some trackers are impossible to obtain, it is clear that application developers implicitly or explicitly incorporate accuracy estimates into their applications already. They may use the published accuracy numbers for a tracker, or experiment with the hardware to determine the accuracy for themselves. Incorporating estimates that are at least as good as these best case and black box estimates

<sup>1</sup>Systems like X11 provide a wide range of software engineering benefits beyond performance and portability, especially when you consider the various toolkits built on top of them. We will not summarize all the benefits here, as they have been widely written about.

should be easy, and developing better estimates using runtime knowledge from the tracking system should be possible in many cases. For example, a vision-based fiducial tracking system can estimate the accuracy of a camera pose based on the accuracy of the camera calibration, the optical angle of the pixels, and the computed position and orientation of the fiducial (which tells the system something about the accuracy of the real world position of the fiducial feature points used to compute the camera pose).

One intuitive way to make tracker information available to the application developer is to estimate the registration error associated with the objects that will be augmented. We think that this approach is effective because it allows the application developer to focus on the effect that she cares about, the misalignment between the object in the real world and the computer generated graphics, as opposed to the causes of registration error (tracker uncertainties and inaccuracies) that are hard to reason with directly.

### *B. Consequences*

We envision that applications that make use of the tracker abstraction layer would be able to automatically reconfigure themselves to deal with a loss of tracker or a discovery of a new tracking system available. For instance, consider the following scenario: a mobile AR application uses a differential GPS, a vision tracking system, and a hybrid tracking system such as Intersense IS-900, to track a person's head. When the user is outdoors, the differential GPS receives information from satellites and the base station. When the head-mounted camera captures an artificial marker, the vision system computes the camera's position in relation to the marker(s), but if there is no fiducial available, the vision tracker will not be used. The IS-900 system is active when the carried receivers are in the range of the stationary emitters.

The user begins outdoors. The application receives information from the GPS system, but has no information available from either the vision system or the IS-900. When this person enters the building, the GPS tracking system will no longer work. If this region of the building is equipped with artificial markers, the vision system will start working and the application will now receive these reports. If

a user approaches a region equipped with IS-900 emitters, this system will start working and the application will have reports from both the IS-900 and the vision system.

In this scenario, the application deals with three different tracking systems, discovers new trackers and loses others as the user moves around. Having this capabilities would allow for the development of a new class of AR applications that could be used for Emergency Response, military, maintenance, etc.

### *C. Discussion*

Usually there is a trade off between abstraction and performance: as more levels of abstractions are added, the performance may be negatively impacted. Augmented Reality applications are very demanding with respect to performance. Ideally, an AR application receives tracking updates at the same rate as it refreshes the display, and each update reports the precise location and attitude of the tracked object at the very moment the display is refreshed. Realistically, various delays exist between acquiring a tracking measurement and rendering the computer generated graphics. The longer the delay, the bigger the registration error is likely to be [2]. It is important to notice that here the analogy with the windowing system can break down, since one may want to be willing to trade off speed for elegance.

Abstraction vs. performance is a very important issue that must be dealt with when adding more levels of abstraction to an AR system. For instance, an application can be designed to use different components. One component could be responsible for the visualization while a completely independent component would be used for tracking. This approach provides a very reasonable level of abstraction and independence. The two components may even be developed separately by teams that specialize in the part they are responsible for, allowing for a better overall system. But, if not developed properly, it may severely compromise performance to the point that the computer generated graphics augmentation always lags behind the real world.

A tightly integrated system, such as ARToolkit, that uses a tight loop to both update the tracker reports and render the computer generated graphics, does not exhibit misregistration due to lag, since it



Fig. 1. An example of the use of registration error estimation. The system highlights a building and two of its windows by outlining the areas 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.

does not render any scene up until a final head position is calculated. (The downside of this approach is that the application will have to drop frames if it can't keep up, and the whole visualization will lag behind user movements).

### III. TRACKERS AND REGISTRATION ERROR

Beyond basic abstractions such as device independence, an important part of allowing application developers to create AR systems that are not intimately tied to trackers is to allow the application more access to the tracker report characteristics. As we mentioned before, we believe that giving the application developer an estimate of the registration error will permit better control over the final graphical result, improving the user's understanding of the situation. Exposing the registration error is both intuitive and powerful, since this is the final effect the application developer is likely to be interested in, and is much simpler to understand and deal with than the actual errors affecting tracker quality. (Such as system lag or absolute pose error).

There are a variety of reasons why application developers have difficulty dealing with multiple trackers directly and estimating registration error:

- 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 [3].
- 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.
- Computing the registration error for nested transformations can be complex as well as time consuming, if not properly implemented.

Besides dealing with the intrinsic characteristics of tracking systems that are difficult to understand and model, applications that deal with heterogeneous tracking systems would also be required to handle the not so simple task of interfacing with all these systems. An increase in system complexity can also be caused by the system need to keep track of which tracking system is active and which ones are out of reach at every moment.

We think that providing application developers a framework that compute the registration error estimate would allow them to develop more complex applications, since they would be free of the burden of dealing with trackers characteristics directly.

#### A. Tracker Report

An important part of dealing with tracking reports, is to understand that they do not correspond to the precise pose of the object being tracked, but rather to the point of highest probability. Although

this is the most likely result, there is a possibility this data is wrong.

We propose that tracker reports not be treated as an absolute description of the pose (location and attitude) of the object being tracked, since this is actually a simplification. Rather, the tracking report should instead be modeled as a *Probability Distribution Function* (PDF). The tracking report received from the system would be the mean of such a PDF, instead of a singular point in space. The accuracy of the tracker can be encoded as the covariance of a Gaussian distribution, giving the programmer a relatively simple expression of the most likely location of the tracker and its accuracy, in a form that can be reasoned with. However, using a Gaussian as a model is still a simplification, since it is not clear if it is possible to measure and quantify the actual statistical characteristics of all tracking systems.

Using a consistent PDF representation to approximate each tracker reading allows for relatively sophisticated methods to be used to reason about the effects of tracker accuracy and make this information available to application developers.

In the following sections, we present a method to estimate the registration error, once the tracker reports are modeled as Gaussian distributions. Then, techniques are proposed to make use of these estimates to improve the quality of the augmentation presented to the user as well as the quality of interaction.

### B. Estimation of the Registration Error

Estimating the registration error for a whole object is a very complex problem. A similar and easier problem is estimating the registration error for a point in 3D. Given the PDF error characterization for all tracker reports, this is a feasible problem that can be solved with statistical tools such as the *unscented transform* [4, 5].

We use the unscented transform to compute a 2D ellipse on the screen that corresponds to the registration error region of a point in 3D. To calculate the registration error for an object, we compute this region for each vertex of a geometrical representation of an object and then aggregate them. We propose two aggregation methods, depending on whether the application wants access to a number or

a region as a representation of the registration error estimate. To produce a numerical representation, the maximum of each of the two main axis are aggregated. To produce a region as a representation of the error estimate, the polygonal representation of the 2D screen ellipses are aggregated by computing the convex hull of these points.

Thus, our framework for estimating the registration error of objects consists of two steps: estimating the registration error of the 3D vertices of the polygonal representation of the object, 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.
- Aggregate these values by either computing the max of the main axis, if we want a numerical representation; or computing the 2D convex hull of the covariance ellipses computed on the previous step, if we want a region.

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.

Although an exact contour is useful when we want to render the silhouette of a complex object, for other uses 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.

## IV. APPLICATIONS OF THE REGISTRATION ERROR ESTIMATE

Once an estimate of the registration error can be provided to the application, the application developer can use this information to improve the

perceived result. The uses of the registration error estimate include:

- Design different graphical representations that can be used to augment an object, based on the registration error estimate (Level of Error).
- Use the estimate to compute a region around the object to guarantee that the augmentation will not block the view of the object, and/or a region guaranteed to be inside the object.
- Determine regions that can be used to guide user interaction.

In sections IV-A through IV-C we examine these techniques in more detail.

#### A. Level of Error

Registration error estimates can be used to choose among available augmentations for the same object. The application developer designs different graphical representations that can be used to augment an object. [6]. The system will, in real time, choose the most appropriate augmentation to use for each circumstance. This idea is similar to the Level-of-Detail approach commonly used in computer graphics.

Consider for example the following scenario, for a maintenance worker responsible for verifying the status of heavy machinery in a factory. The factory is equipped with a low-accuracy tracking system over its entire floor and a high-accuracy tracking system in the region where the machine is located. When the person enters the factory, the system may display some high level information about a given machine that is located over a 100 ft from where the person stands. A text message (*drilling tool on the far right needs maintenance*) would be displayed and no attempt would be made to spatially superimpose graphics on top of the machine if the registration error is too big. As the maintenance person approaches the machine, and consequently the registration error is reduced, allowing graphical augmentations to be used, the system would automatically replace the textual augmentation with a low resolution graphical representation of the machine. A more detailed augmentation would be provided when the maintenance person gets closer to the machine.

To build an application for this scenario, it is necessary that the application developer provide

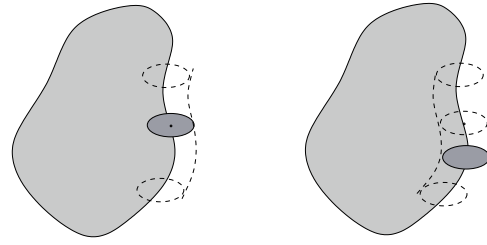


Fig. 2. Expansion adds all neighboring points at distance smaller than the radius of the kernel, in a given direction. Contraction removes all points closer than the radius of the kernel.

more than one augmentation for the object to be augmented as well as the registration error ranges for which each augmentation should be used. The underlying structure needs to be able to compute, in real time, the registration error associated with the object and choose the most appropriate augmentation to be used.

#### B. Bounding Regions

An important issue in AR is where on the screen to place the augmentations so that it does not block the view of the objects the user cares about. Bell et al. [7] revisited the idea of using *User Interface Management Systems* (UIMS) to automatically manage the augmentations' positions on the screen. To be able to do this, one needs to know which portion of the screen is occupied by which object. To help this task, two regions can be computed as follows:

**Inner Bounder** – This region is determined by removing all the points inside the original geometrical model whose distance to the border is less than the estimate. This region is guaranteed to be inside the object to be augmented. It can be used, for instance, to place labels on buildings.

**Outer Bounder** – The union of the inside of the model plus the points that are at a distance less than the estimate to any point inside the original region. The object is guaranteed to be inside this regions. If an application wants to display some graphics that do not block a object, one can compute the outer bounder for that object and place the augmentation outside that area.

These regions can be computed by *contracting* and *expanding* the original bounding volume using the estimated registration error as the kernel for these operations [8]. These operations are similar

to the morphological operations of *dilation* and *erosion*. (see figure 2.)

### C. Interaction

One of the most common interaction task, and probably the most important for AR, is the object selection task. Our interest is in using either *direct interaction* — where the user points directly at the object to be select, either by using her fingers or hand — or *indirect interaction* — where an intermediate device (be it a mouse, stylus, track-ball, etc.) is used to select the object. To help the user interact with the the system and make sure that the system is receiving the input the user intended to, one can increase the size of the pointer or rearrange the augmentations on the screen, depending on the registration error expected.

An estimate of the registration error is used to compute a region around the object, as presented on the previous section (IV-B). This region corresponds to the projection in 2D of the 3D space where this object is guaranteed to be, up to the confidence interval specified. The outer bounder region is then used as a target for user interaction. Then, the system computes the intersection between the region associated with the object and the region associated with the input data, be it provided by direct or indirect interaction.

## V. CURRENT AND FUTURE WORK

To test these ideas, we have being currently extending an open source scene graph to allow it to be used as an underlying structure to develop AR applications that support heterogeneous tracking systems. To the original scene graph we added support for AR (such as trackers, fiducial finding, video-mixed and optical see-through, etc.) and for uncertainty transformations. Most of the other additions derive from this later decision. For instance, a registration error estimator that traverses all relevant nodes and computes an estimate of the registration error. We also added some higher level building blocks to leverage these estimates. For example, Level-Of-Error nodes — an extension of 'switch' nodes that choose which child to use based on the error estimate.

## ACKNOWLEDGMENTS

This work was supported by Siemens via a GVU Industrial Affiliate Grant, and ONR under Grant N000140010361.

## REFERENCES

- [1] E. Foxlin, *Motion Tracking Requirements and Technologies*. Lawrence Erlbaum Associates, Inc., 2002, ch. 7.
- [2] R. L. Holloway, "Registration Error Analysis for Augmented Reality," *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 4, pp. 413–432, August 1997.
- [3] S. You, U. N. R., and Azuma, "Orientation tracking for outdoor augmented reality registration," *IEEE Computer Graphics and Applications*, vol. 19, pp. 36–42, 1999.
- [4] J. K. Uhlmann, "Simultaneous map building and localization for real time applications," University of Oxford, Tech. Rep., 1994, transfer thesis.
- [5] 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*, vol. 5, no. 3, pp. 477–482, March 2000.
- [6] B. MacIntyre and E. M. Coelho, "Adapting to dynamic registration errors using level of error (loe) filtering," in *International Symposium on Augmented Reality (ISAR 2000)*, Oct 5-6 2000.
- [7] B. Bell, S. Feiner, and T. Hollerer, "View management for virtual and augmented reality," in *ACM Symposium on User Interface Software and Technology*, Nov 11-14 2001.
- [8] B. MacIntyre, E. M. Coelho, and S. Julier, "Estimating and adapting to registration errors in augmented reality systems," in *IEEE Virtual Reality (VR 2002)*, March 2002.