Spatially Adaptive Augmented Reality

A Thesis
Presented to
The Academic Faculty

by

Enylton Machado Coelho

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
December 2005

Spatially Adaptive Augmented Reality

Aos meu pais, Nilmar e Luzia

pelo amor e encorajamento que me deram

durante toda a minha vida.

To my parents, Nilmar and Luzia

for their love and support

during my whole life.

PREFACE

This dissertation uses material previously presented at:

- IEEE Virtual Reality Conference (VR 2002) – “Estimating and Adapting to Registration Errors in Augmented Reality Systems” (Paper),
- the Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 04) – “osgAR: A Scenegraph with Uncertain Transformations” (Paper), and
- the Eighteenth Annual ACM Symposium on User Interface Software and Technology (UIST 2005) – “Supporting Interaction in Augmented Reality in the Presence of Uncertain Spatial Knowledge” (Technote).

ACKNOWLEDGEMENTS

The NIH National Research Resource in Molecular Graphics and Microscopy at the University of North Carolina at Chapel Hill, supported by the NIH National Center for Research Resources and the NIH National Institute of Biomedical Imaging and Bioengineering for the VRPN library.

OSGAR makes use of previously developed libraries, specifically for the low level communication with trackers, cameras, etc. Thus, it builds upon the work of the developers of the ARToolKit library, as well as Brendan Hannigan and Jeremy Johnson's implementation of the low level libraries that support the cameras and provide an interface for the classes that make video in the background possible.

Rob Kooper implemented the original software libraries that allowed us to have access to the IS-600 tracker reports, early in the days when we were still using Java and Java 3D. He also provided support during our initial development of spatially adaptable AR for early GVU demo days. Cindy Robertson patiently protected the English language and helped me improve the text on many documents and slides, besides testing not so reliable versions of my toolkit. I also acknowledge other members of the Augmented Environment Laboratory for the discussions, relevant ideas and comments on the project.

Simon Julier developed the unscented transform and kindly provided a version that is suitable to be used in scene graphs. He also helped with the mathematical concepts. Brian Rink helped with the software development of the train set tracker and extended the VRPN library to support the broadcast of the uncertainty information.

I'm thankful for the insightful comments provided by the members of my committee. Specially my advisor, Blair Macintyre, for providing the guidance necessary for me to navigate this unchartered route.

This work was supported by the Office of Naval Research under Contract N000140010361. I was a awarded a scholarship by CAPES, Brazil.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

One of the most important problems faced when creating real-time, mobile augmented reality systems is *registration error* – the misalignment between the computer generated graphics and the physical world the application is trying to augment. Such misalignment may either cause the information presented by the application to be misleading to the user or make the augmentation meaningless.

In this work, we question the often implicit assumption that registration error must be eliminated for AR to be useful. Instead, we take the position that registration error will never be eliminated and that application developers can build useful AR applications if they have an estimate of registration error. We present a novel approach to AR application design: *Spatially Adaptive Augmented Reality* (i.e., applications that change their displays based on the quality of the alignment between the physical and virtual world). The computations used to change the display are based on real-time estimates of the registration error. The application developer uses these estimates to build applications that function under a variety of conditions independent of specific tracking technologies.

In order to support Spatially Adaptive AR, this research establishes a theoretical model for AR that accounts for the static uncertainty in the system. We call this sub set of Spatially Adaptive AR, *Uncertainty Aware* AR. The reification of these theoretical contributions is a toolkit that supports the design of Uncertainty Aware AR applications: OSGAR. This work describes OSGAR in detail and presents examples that demonstrate how to use this novel approach to create adaptable augmentations as well as how to support user interaction in the presence of uncertainty.

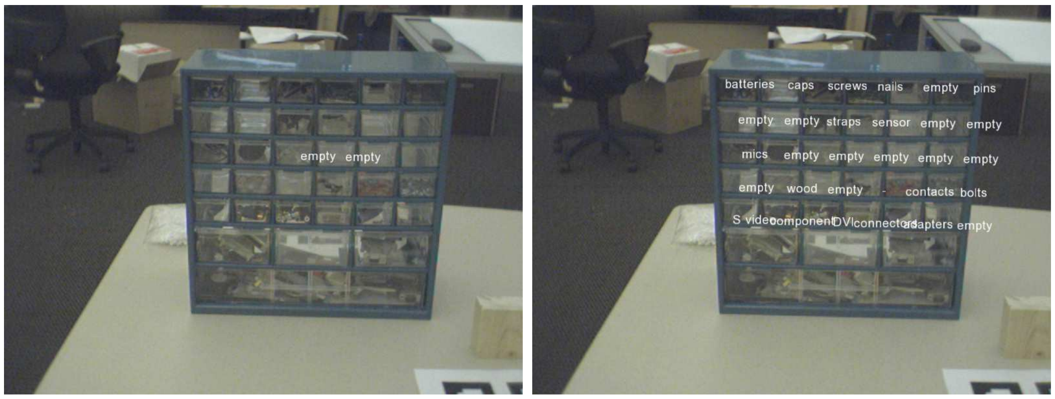
CHAPTER I

INTRODUCTION, THESIS STATEMENT AND CONTRIBUTIONS

1.1 Introduction

Augmented Reality (AR) allows one to see computer-generated graphics superimposed onto the environment around us. In a perfect world, the graphics would be flawlessly integrated with the physical environment and users would not be able to distinguish them. In practise, in additions to imperfections such as the lack of correct lighting parameters for rendering the computer generated graphics, the physical objects and the graphics created by the computer do not align precisely, causing what is called *registration error* . A significant percentage of the research in the AR community has been directed towards eliminating registration error, based on the often implicit assumption that AR applications will only be useful when virtual-physical registration is perfect. The pursuit of perfect registration has often led applications developers to assume that registration error would eventually be eliminated [4, 9, 56, 63]. Since this has not happened and tracker inaccuracy has been identified as one of the main causes of registration error [41, 42], AR applications are usually fine-tuned to make the best use of the capabilities and limitations of the tracking technology [26, 71]. The drawback of this approach is that if the tracker characteristics change, the application may fail to convey the information the application developer intended, because she had not planned for different characteristics.

This work questions the underlying assumption that perfect registration is necessary for AR to be useful. Instead, we believe that carefully designed applications that adapt in real time to the ever changing environmental conditions can enable reality to be augmented in the presence in registration error, if the amount of error is known by the application developer.


empty empty
batteries caps screws nails empty pins
empty empty straps sensor empty empty
mics empty empty empty empty empty
empty wood empty - contacts bolts
S video component DVI connector adapters empty

(a) Ambiguous augmentation (b) Contextual information helps the user

Figure 1: The registration error is the same in both pictures. Only the presentation has changed.

For example, in Figure 1 a tool-box is augmented with labels that specify the content of each drawer. (The tool-box is used because it presents a repeating structure – drawers – that are not easily differentiated from each other). This can also be the case for windows in a building, components on an integrated circuit, levers and buttons on a control panel, etc. If each individual object is completely different from the others in a set, humans may be able to understand the intent of the augmentation even under severe registration error. But ambiguity may arise when the objects the application is trying to augment are similar and the augmentation does not make it clear which object it refers to.

Figure 1(a) provides an example of how registration error can lead to misleading augmentations. Since each of the two labels appear over two adjacent drawers, one cannot tell which drawer the label refers to. Figure 1(b) shows the system working under exactly the same conditions. Because all labels are shown, one can count the labels and figure out to which drawer a label refers to. While this is obviously not an ideal solution for a real situation, since it would clutter the view space, it illustrates that changing the way information is displayed may allow the user to understand the intent of the augmentation. Even though the registration error is the same on both Figures 1(a) and 1(b), the intent in Figure 1(b) is clear, while the intent of the application in Figure 1(a) is not.

Inspired by the anecdotal evidence in the previous example, we argue that carefully designed augmentations may allow AR applications to successfully communicate the intent of the application even under severe registration error conditions. In order to choose an appropriate augmentation at run time, the application developer needs to have access to an estimate of the registration error for each individual object the application intends to augment. This work provides the application developer with an estimate of the registration error that is intuitive, yet accurately reflect the current condition. Moreover, the computation of such estimate is done in real time and does not significantly increase system latency.

This dissertation is structured as follows: The rest of this chapter enunciates the thesis statement and summarize the contributions of this work. Chapter 2 presents previous work that is relevant to this dissertation. Chapter 3 introduces the idea of spatially adaptive augmented reality applications. The reification of those ideas is a toolkit, OSGAR, described in Chapter 4. Chapters 5 and 6 show examples of how OSGAR can be used to create spatially adaptive augmented reality applications by changing the input and output respectively. Chapter 8 describes other projects that are made possible by this work, and thus further validate the thesis statement.

1.2 Thesis Statement

The real-time estimation of registration error allows straightforward spatial reasoning and supports the development of Augmented Reality applications that are robust in the presence of changing uncertainty in the spatial knowledge.

1.3 Contributions

1. **Introduce the notion of using Spatially Adaptive Augmented Reality to support a new class of AR applications.** In order to both cope with varying tracking characteristics and allow the application developer to easily reason about the spatial arrangement of the physical world and its implication for the augmentations, Augmented Reality infrastructure needs to be designed to support adaptive applications. Because spatially adaptive AR applications encourage the designer to create

systems based on the estimated registration error, such applications are not dependent on specific implicit assumptions.

2. **Make use of registration error estimates as a basis for Spatially Adaptive AR applications.** Using an estimate of the registration error that is view point dependent as the metric which drives the adaptable system is intuitive for the application developer, more correct than the current methods and practical to implement.

3. **Establish a theoretical framework that supports the creation of Uncertainty Aware AR applications.** This work identifies the limitations of current models and expands them to be expressive enough to support the use of the static uncertainty in the system to compute the registration error estimate.

4. **Present a toolkit that computes a registration error estimate and makes it available to the application developer.** The theoretical concepts are realized through a toolkit that allows the design of tracker independent, Uncertainty Aware AR applications. This toolkit uses a modified version of the unscented transform [51] suitable for real time computation on a scene graph. The original unscented transform works on a vector space whose dimension is dependent on the number of parameters. A recursive version is more appropriate for scene graphs and was developed.

5. **Demonstrate how the registration error estimates can be used to modify the augmentations in real time.** We developed a set of widgets that make use of the registration error estimates to properly select augmentations in order to create AR applications that adapt to the current environmental characteristics.

6. **Demonstrate how the registration error estimates can be used in real time to support user interaction.** The registration error estimate is used to provide the basis for the development of user interaction mechanisms that support user interaction in the presence of uncertainty.

These contributions will be discussed in much more detail in the next two sections.

1.4 Theoretical Contributions

The theoretical contributions of this work consist of the introduction of the notion of Adaptive AR applications, the use of the registration error estimate to allow for adaptive AR and a mathematical model that is more suitable for adaptive AR than the current computer graphics models being used by the AR community. Each of these contributions is described in more detail below.

1.4.1 Spatially Adaptive Augmented Reality

The development of spatially adaptive augmented reality will allow applications to be used in real life scenarios and industrial installations where it is not practical to guarantee that sensor accuracy will remain constant during the complete period of operation. This should also allow for further investigation of the most appropriate way to convey information to the end user.

1.4.2 Use of Registration Error Estimation

This work proposes to use the registration error estimate associated with each object in the scene to drive the way the system chooses which augmentation to be used, based on the specifications of the application designer. The choice to use the registration error estimate is based on the fact that this is intuitive to the application designer – since it is what the user will perceive – and at the same time practical to estimate.

1.4.3 Mathematical Model that Supports Uncertainty Aware AR

The current computer graphics models widely used in photo realistic rendering do not match the needs of adaptive AR. This work presents a model that is much more suitable for the types of applications that will be necessary to support the use of Augmented Reality in locations where there is not complete knowledge of the environment. This is done by taking into account the static uncertainty in the system when computing the registration error estimate.

1.5 Applied Contributions

The theoretical concepts presented above are demonstrated in OSGAR, a toolkit that automatically computes the registration error estimate and makes it available to the application developer in ways that are easy to use and intuitive, while also comprehensive and powerful. On top of OSGAR, we implemented a set of widgets that provide uncertainty aware augmentations. We also leveraged OSGAR to support user interaction in the presence of uncertainty by developing interaction mechanisms that benefit from the registration error estimate.

1.5.1 OSGAR: Supporting Uncertainty Aware AR Systems

OSGAR was designed to embody the concepts presented here, while closely matching the semantics and standard interface used by Open Scene Graph (OSG). Any application that already uses OSG can be turned into an AR application with very little alteration. The modifications are straight forward, since it follows the Design Patterns commonly used by OSG. As an example of the effort devoted to make it as compliant as possible with OSG, not a single line of code in OSG was changed. Instead, all implementations were done by extending the original classes. This required more work in some instances, but allows new versions of OSG to be used with minimal changes.

By using OSGAR, the application developer has access to either a region on the screen or a single numerical value that corresponds to the registration error estimate associated with an object. The choice of how to make the estimate available to the application developer reflects a concern to create a toolkit that is as intuitive as possible for an application designer using it. Moreover, the toolkit does not restrict itself to support only a small set of applications by encouraging extensions to be developed. For instance, while it does not have a predictive filter such as a Kalman filter, it provides support for one to be implemented and automatically used by the system.

In regard to handling the uncertainty associated to modeling the physical environment, while the current implementation does not define geometrical models that represent accuracy, their use would be possible since the computations happen on a per vertex basis. Thus, if models associate uncertainty to the vertices, this information could be immediately used without requiring any modification to the computations performed by OSGAR. OSGAR is described in detail in Chapter 4.

1.5.2 Uncertainty Aware Augmentations

The idea of changing the way information is displayed based on the registration error was first introduced by our work presented at the first IEEE and ACM International Symposium on Augmented Reality (ISAR 2000) [61]. Because at that time we lacked an automated way to estimate the registration error for an object, the choice of which augmentation to use was driven by heuristics we implemented that took into account the relative pose of individual objects in relation to the virtual camera. Once we had the capability to estimate the registration error in real time on a per object basis, as implemented in OSGAR and described in the previous section, we could begin developing a more comprehensive framework to support adaptive augmentations.

In order to do this, we developed a set of widgets that make use of the registration error estimate provided by OSGAR. We implemented three widgets: *Bounding Regions*, *Level of Error* widget and *Label Placer* widget.

Bounding Regions are used to display graphical representations of the regions computed for the geometrical models of physical objects. This widget can display the vertex ellipses, the inner region – corresponding to the area on the screen where the physical object must be, the outer region – corresponding to the area on the screen where the object might be, or any combination.

Level of Error automatically chooses between different children of a group node, allowing the application developer to specify different augmentations corresponding to the same real object. The most appropriate augmentation will be chosen according to the registration error estimate.

Label Placer computes where to position the labels for a given object based on the computed inner and outer regions of the model. Using this widget, the application developer has the option to keep the label always inside the object (based on the inner region) or guarantee that the label will never block the object (based on the outer region). This widget computes where to place the label to follow one of those constraints. Chapter 5 explains in detail how each of these widgets is implemented and provides examples on how they can be used.

1.5.3 Support for User Interaction in the Presence of Uncertainty

Interaction in AR is a complex problem. To all the challenges associated to 3D interaction, AR interaction adds the fact that the knowledge about the components of the system may be imperfect, the accuracy of the tracking device being one of the most common factor. Some authors have acknowledged this fact [72] and have tried to develop *ad hoc* methods to deal with it. For instance, Olwal and al. [69] use a cone instead of a single ray for object selection, since objects that are farther away from the user are more likely to present more inaccuracy in relation to the user's view point. Other researchers proposed using multi-modal interaction to disambiguate the object being selected by the user [52].

While those approaches are helpful in some contexts, we believe that interaction in the presence of uncertainty can be better supported by leveraging the registration error estimate mechanism implemented in OSGAR. We implemented a tool, very similar to *picking* provided in many other 3D libraries and toolkits, by detecting collision between the estimated registration error region of the pointer and the region of each object in the scene. We also provide virtual targets for objects in the case the user cannot select the physical object directly. The position of each virtual target is chosen depending on the registration error estimate associated to the pointer: the higher the registration error estimate, the bigger the distance between the targets. Determining the distance between the targets in real time and adjusting it to the registration error estimate of the pointer allows for unambiguous selection, since the user will only be able to select one artificial target at a time.

The selection mechanism we present provides a simple yet robust foundation for creating interaction techniques in AR environments, and can feed directly into the multi-modal approaches described by other authors [52, 69]. Chapter 6 describes the interaction mechanisms we developed, provide implementation details, and present some examples on how these can be used to support the construction of more elaborate interaction techniques.

1.6 Evaluation

This work can be evaluated in relation to the impact it caused and the research it has influenced or allowed to spawn. The ability to estimate the registration error of object in real time in an AR application based on the uncertainty of the components of the environment has enabled a new class of applications to be developed, showing how previously impractical applications can be implemented simply, elegantly and robustly. This work has changed the way many AR researchers think about building AR applications and spawned further development in the data reported by the trackers, research on how to better augment the environment given the estimates and which architecture to use to represent the uncertain spatial relations of each piece of the environment. Other research projects that were either influenced or made possible by this work are described in Chapter 8.

1.7 Conclusion

This work demonstrates a new approach for developing AR applications: adapt to the dynamically changing uncertainty of the system's knowledge of the surrounding environment, instead of using pre-determined accuracy estimates and hoping that the characteristics will not change during the time the application is running. This novel approach allows the application to leverage ideal conditions and to degrade gracefully out those ideal conditions. Implementing system behaviors for a range of conditions enables the creation of more robust AR applications.

In order to support the creation of such robust, uncertainty aware AR applications, we developed a toolkit that estimates the registration error in real time and makes it available to the application developer in a intuitive way. This work also demonstrates how such toolkit can be used to implement uncertainty aware AR applications by creating examples that make use of such estimates to change the augmentation in real time. The registration error estimates were also leveraged to develop interaction mechanisms that make it easier for the developer to support user interaction in the presence of uncertainty.

CHAPTER II

RELATED WORK

2.1 Introduction

The related work discussed in this chapter is grouped in five different categories. Firstly, we describe some applications that show the usability of Augmented Reality in the industry, medicine and outdoors. Secondly, studies by some researchers who have tried to deal with the registration error problem are described. Thirdly, some work related to adaptive interfaces is addressed. Then, work related to user interaction in AR are described. Lastly, some related work in other areas of research are introduced.

2.2 Augmented Reality Systems

This section describes some interesting Augmented Reality systems introduced over the last decade [2] or so, to present a broad overview of the types of applications researchers have tried to use Augmented Reality for. These applications have different requirements in relation to the acceptable level of registration error and the tracker accuracy necessary to conform with such requirements. They all have in common that their implementations rely on unchanging tracker characteristics to perform appropriately, although they were intended to work in environments where this cannot be guaranteed. In other words, they all lack a mechanism to allow them to adapt to the prevailing characteristics of the environment and would therefore benefit from the work described in this dissertation.

2.2.1 Industrial Applications

One of the first attempts at deploying an Augmented Reality application in an industrial installation was done at Boeing. Curtis et al. [18] implemented a system to help workers assemble the wires necessary on an airplane and reported on the problems they faced while trying to deploy such an application in a industrial setting. Besides airplane assembly, researchers have also suggested that AR can be used to support maintenance in power plants [57] and to support construction tasks [75]. Navab et al. [66] augmented the user's view of the physical world in industrial installations with legacy blueprints and building data.

In order to support those applications, ubiquitous trackers abstractions [68, 91] (i.e.: a unified tracker abstraction that would allow applications to have access to a multitude of tracker devices without actually having to access them directly) and system architectures have been proposed [6]. Also, to support the necessary mobility of these systems, the use of wearable computers in augmented reality [81] have been addressed. Each and every one of these systems could benefit from the approach presented here.

2.2.2 Medical Applications

Many medical augmented reality applications support the visualization of medical images [65] such as computerized tomography (CT), magnetic resonance (MRI), and ultrasound [4]. Maurer et al. [9] concentrated specifically on image-guided neurosurgical planning and navigation using a video see-through head-mounted display augmented reality system. One of the problems tightly linked to endoscopic surgery is the fact that, because of the narrow field of view, it is sometimes quite difficult to locate the objects that can be seen through the endoscope. Accordingly to Devernay et al. [24], this is especially true in cardiac surgery, where it is difficult to not confuse two coronary arteries on a beating heart. State et al. [83] presented a real-time stereoscopic video-see-through AR system applied to ultrasound-guided needle breast biopsy. The AR system was used by a physician during procedures on breast models and during non-invasive examinations of human subjects. The system merges rendered live ultrasound data and geometric elements with stereo images of the patient acquired through head-mounted video cameras and presents these merged images to the physician in a head-mounted display. The physician sees a volume visualization of the ultrasound data directly under the ultrasound probe, properly registered with both the patient and the biopsy needle.

Besides visualization, another important medical application that has drawn the attention of AR researchers is how to make use of AR systems to support surgeons in needle placement tasks [63, 64]. Traub et al. [86] stated that "optimal port placement and intra-operative navigation in robotically assisted minimally invasive cardiovascular surgery are essential for the improvement and success of a teleoperator based heart operation." They developed a new system incorporating both port placement planning and intra-operative navigation. The optimal port placement was planned off-line on a three-dimensional virtual reconstruction of the patient's computed tomography scan. This planned data was then used to perform an accurate in-vivo port placement, which was achieved by augmented reality techniques superimposing virtual models of the thorax and the teleoperator arms on their real world counterparts.

Most of the work addressed here present critical systems that rely on, but do not assess the application's accuracy. These systems, as implemented, would not alert the user if the accuracy cannot be guaranteed or falls below a pre-determined level considered necessary to the task it supports. We believe that the ideas presented in this work could be beneficial for the further development of medical AR applications. Such applications would detect when registration error is not enough low to perform the task in hand and notify the user, making them safer and more feasible to deploy in real situations.

2.2.3 Outdoors Applications

In [26], Feiner et al. described a prototype system that combines the overlaid 3D graphics of augmented reality with the untethered freedom of mobile computing to explore how these two technologies working together make possible wearable computer systems that can support users in their everyday interactions with the world. They introduced an application that presents information about buildings on the university's campus, using a head-tracked, see-through, head-worn 3D display, and an untracked, opaque, handheld, 2D display with stylus and trackpad.

Piekarski and Thomas [71] introduced a novel user interface idea, using a glove based menuing system and 3D interaction techniques designed to support applications that allow users to construct simple models of outdoor structures. The construction of models is performed using various 3D virtual reality interaction techniques, as well as using real-time constructive solid geometry, to allow users to build up shapes with no prior knowledge of the environment. Other researcher have also addressed the problem of how to use augmented reality to support outdoor construction [56].

Julier et al. [48] used filtering to prevent some of the models from being displayed and thus reduced information overloading. According to the author, the filtering mechanism would allow the quest for better ways to convey the intent of the augmentation without cluttering the user's view of providing ambiguous information.

Both Feiner's and Piekarski's system were designed to work outdoors using GPS devices to locate their position in space. Both systems fail to take into account that the GPS system may stop working or suffer some interference. For instance, if a person goes to a place surrounded by buildings or trees, the receiver will not be able to maintain a direct line of sight to the satellites, causing the quality of the report to deteriorate. This may not be very relevant for their specific applications, but could be a more serious problem, for instance in the case of an Emergency Response AR system. This work presents a mechanism that would allow outdoors AR applications to degrade gracefully, thus enabling them to still support the task they were designed for even under less than perfect environmental conditions.

2.3 Registration Error

Previous work has acknowledged that registration error is an important problem in the Augmented Reality domain and has tried to address it. Some relevant work is mentioned in this section.

Azuma and Bishop [3] argued that many Augmented Reality applications will not be accepted until we can accurately register virtual objects with their real counterparts. In previous systems, such registration was achieved only from a limited range of viewpoints when the user kept his head still. Their work claimed to offer improved registration in two areas. First, it demonstrated accurate static registration across a wide variety of viewing angles and positions. Second, dynamic errors that occured when the user moved her head were reduced by predicting future head locations by employing inertial sensors mounted on the HMD to aid head-motion prediction. They claimed that on average, prediction with inertial sensors produced errors 2-3 times lower than prediction without inertial sensors and 5-10 times lower than using no prediction at all.

Jacobs et al. [45] stated that "Registration (or alignment) of the synthetic imagery with the real world is crucial in augmented reality (AR) systems. The data from user-input devices, tracking devices, and imaging devices need to be registered spatially and temporally with the user's view of the surroundings." In their work, each tracker device had an associated delay between its observations of the world and the moment when the AR display presented to the user appears to be affected by a change in the data – the relative latencies. They identified the relative latency as a source of misregistration and tried to reduce it by employing general methods for handling multiple data streams with different latency values associated with them in a working AR system. They measured the latency differences (part of the system dependent set of calibrations), time-stamped on-host, and used these values to adjust the moment of sampling and interpolate or extrapolate the data streams. The authors claim that by using these schemes, a more accurate and consistent view was computed and presented to the user. While this dissertation doesn't deal with the temporal effects of the registration error estimate, their approach would be a natural way to extend the research presented here.

Höllerer et al. [40] addressed the problem that the position-tracking accuracy of a location-aware mobile system could change dynamically as a function of the user's location and other variables specific to the tracker technology used. This is especially problematic for mobile augmented reality systems, which ideally require extremely precise position tracking for the user's head, but which may not always be able to achieve the necessary level of accuracy. He claimed that while it was possible to ignore variable positional accuracy in an augmented reality user interface, this can make for a confusing system; for example, when accuracy is low, virtual objects that are nominally registered with real ones may be too far off to be of use.

All this research demonstrated attempts at minimizing the registration error. While this work would benefit from improved registration, the main concern is in allowing applications to function properly under varying environmental conditions. AR applications need to be developed to be useful not only when all conditions are ideal and every component of the system is functioning properly. They need to be guaranteed to degrade gracefully when one or multiple subsystems present some deficiency.

2.4 Adaptive Interfaces

Allowing the application developer to create interfaces that adapt to the best scenario presented by displays has been studied in the context of both 2D and 3D interfaces. This work uses an analogous approach: allow the interface to automatically change so that some requirement previously specified by the designer is satisfied to the best possible extent.

In [33] Gleicher described Bramble, a toolkit for constructing graphical editing applications. The primary focus of Bramble was to improve support for graphical manipulation by employing differential constraint techniques. A constraint engine capable of managing non-linear equations maps interactive controls and constraints to object parameters. This allows objects to provide mathematical outputs that are easily composed, rather than exposing their internal structure or requiring special purpose interaction techniques. The model of interaction used with the differential approach has a continuous notion of time, which provides the continuous motion required for graphical manipulation, Bramble provides a LISP-like extension language and support for other application features such as windows and buttons.

Stevens et al. [84] presents the architecture for an extensible toolkit used in construction and rapid prototyping of three dimensional interfaces, interactive illustrations, and three dimensional widgets. The toolkit provides methods for the direct manipulation of 3D primitives which can be linked together through a visual programming language to create complex constrained behavior, and employed a constraint resolution technique based on a dynamic object model similar to those in prototype delegation object systems.

Fogarty and Hudson [28] improved on previous work to develop an experimental toolkit that uses optimization-bases approaches to support optimization for interface and display generation. GADGET provides convenient abstractions of many optimization concepts, as well as mechanisms to help programmers quickly create optimizations, including an efficient lazy evaluation framework, a powerful and configurable optimization structure, and a library of reusable components. The authors state that "together these facilities provide an appropriate tool to enable exploration of a new class of interface and display generation techniques".

Gajos and Weld [32] introduced a system that treats interface adaptation as an optimization problem. When asked to render an interface on a specific device, their system searches for the rendition that meets the device's constraints and minimizes the estimated effort for the user's expected interface actions.

2.5 User Interaction

Previous attempts at supporting user interaction in AR systems realized that in some situations the user's ability to interact with the system will be impaired by the inaccuracy of the underlying tracking device. For instance, Olwal et al. [69] make use of geometrical shapes that correspond to the possible intrinsic error in the system. Specifically, to support selection, a virtual cone is used instead of the common ray shooting technique, because the angular accuracy of selecting an object decreases as the distance from the user increases, due to the innate angular uncertainty associated with tracking devices. Kaiser et al. [52] employ multi-modal interaction techniques to help the user select objects, in the hope that one mode of interaction can compensate for the deficiencies in the other modes in the same way the Kalman Filter is used to fuse information from multiple, diverse data sources.

Some AR researchers have also addressed the issue of changing the spatial location of the augmentation on the screen so that the computer generated information can be integrated as naturally as possible with the surrounding physical world and does not interfere with the user's view of the world. Bell et al. [7] introduced view-management component for interactive 3D user interfaces that maintains visual constraints on the projections of objects on the view plane, such as locating related objects near each other, or preventing objects from occluding each other.

While all these works recognize that the inaccuracy of the system is a hindrance to user interaction, the best they could do was guess what that inaccuracy was, since they did not have access to a model of such inaccuracy.

2.6 Related Research Areas

Modeling the inaccuracy of the system has been done in robotics and vision communities for over twenty years [23]. Their intent is to use this information to improve the quality of the estimates they are producing.

Haralick [37] described how to propagate approximately additive random perturbations. The only assumption is that the scalar functions have finite second partial derivatives and that the random perturbations are small enough so that the relationship between this function evaluated at the ideal but unknown input and output quantities and the observed input quantity and perturbed output quantity can be approximated sufficiently well by a first order Taylor series expansion. We employ a recursive version of the unscented transform [51] to propagate the uncertainty down the scene graph. Unlike Haralick, we do not use a Taylor series expansion. Details and discussion about it are presented in Section 3.9 (page 28) and Appendix B (page 100).

Dellaert et al. [22] argued that probabilistic approaches are among the most promising candidates to providing a comprehensive and real-time solution to the robot localization problem. They introduced the Monte Carlo Localization method, where they represented the probability density involved by maintaining a set of samples that were randomly drawn from it. By using a sampling-based representation they obtained a localization method that can represent arbitrary distributions and is faster, more accurate and less memory-intensive than earlier grid-based methods

Davison and Murray [21] introduced an active approach to sensing that provided a focused measurement capability over a wide field of view. Such approach allowed correctly formulated Simultaneous Localization and Map-Building (SLAM) to be implemented with vision, thus permitting repeatable long-term localization using only naturally occurring, automatically-detected features and addressing such issues as uncertainty-based measurement selection, automatic map-maintenance and goal-directed steering.

Until 2003, most work in simultaneous localization and map-building (SLAM) for mobile robots had focused on the simplified case in which a robot is considered to move in two dimensions on a ground plane. While this was often a good approximation, a large number of real-world applications require robots to move around terrain which has significant slopes and undulations, and it was desirable that these robots too should be able to estimate their locations by building maps of natural features. Davison and Kita [19] described a real-time EKF-based SLAM system permitting unconstrained 3D localization, and in particular developed models for the motion of a wheeled robot in the presence of unknown slope variations.

Kelly [55] claimed that when navigating from landmarks, the numerical computation of the differential behavior of the mapping from measurements to states is well understood. Their paper seeks to develop tools to foster an intuitive grasp of the spatial variation in the differential behavior of this mapping. Graphical methods are adapted from marine navigation in order to introduce the issues in low dimensional cases. The implicit function theorem and the pseudoinverse are then used in the general case to derive a quantity analogous to the geometric dilution of precision (GDOP) as it is used in satellite based guidance. An explicit expression is obtained for the conditioning of the triangulation computation as a function of position. This expression also directly provides the conditions under which the computation becomes singular.

Leonard and Durrant-Whyte [59] showed an application of the extended Kalman filter to the problem of mobile robot navigation in a known environment. They developed an algorithm for model-based localization that relies on the concept of a geometric beacon – a naturally occurring environment feature that can be reliably observed in successive sensor measurements and can be accurately described in terms of a concise geometric parameterization. The algorithm was based on an extended Kalman filter that utilized matches between observed geometric beacons and an a priori map of beacon locations.

2.7 Conclusion

This Chapter presented work that is related to this dissertation. Some industrial, medical, and outdoors AR applications were presented as examples of areas that could benefit from the spatially adaptive approach described in this work. Since the main component of this work is the real-time estimation of the registration error, previous work that addressed the registration error problem were identified. While their approach could be used with the mechanism proposed here and would be a natural extension of our approach, the main benefit of developing spatially adaptive AR applications is that they work not only when some conditions are met, but also when the environment presents more challenging conditions.

The mathematical approach used to estimate the registration error given the representation of the uncertainty as a PDF is similar to the tools used in other areas such as robotics. Some work presented in conferences in these related areas were highlighted here, since they were relevant in expanding the body of knowledge and establishing the mathematical tools that were used to estimate the registration error in OSGAR.

Once we can compute a real time estimate of the registration error, we propose that this information be used to change both the output and input of AR applications. Some adaptable interfaces have been developed both in 2D and in 3D, and we listed some of the work that are related to interfaces that change in real time to satisfied some pre-determined conditions created by the application developer. On the output side, some AR researchers tried to develop user interaction techniques that are more suitable to AR by using some *ad-hoc* techniques such as enlarging the selection pointer or employing multi-modal methods. They lacked a method to estimate a tight value that could be used to guide their approach. This work provides such method.

CHAPTER III

SPATIALLY ADAPTIVE AUGMENTED REALITY

3.1 Introduction

Recently, Honda and Microvision[1] evaluated a wearable maintenance system that uses a non-tracked see-through heads-up display to present *in-situ* automotive maintenance information to trained technicians. According to them, this system was demonstrably useful (resulting in a quoted 38% improvement over the non-wearable version) despite the fact that the graphics were not registered with the physical world. This system raises some interesting questions for the AR community: would an AR version of the system be even more effective? Would precise registration be necessary, or would a coarsely registered version using moderately accurate tracking be as effective (e.g., by allowing the current system's graphics to be generated from the technician's approximate viewpoint, even if they aren't registered)? Would some tasks benefit greatly from precise registration, while others would not?

When one considers practical issues of creating, deploying and maintaining such a system (such as cost and robustness) these questions become critical. To study these questions, we propose the creation of augmented reality applications that support augmentations spanning a range of registration levels – from not registered to perfectly registered. We call these applications spatially adaptive augmented reality, since they adapt to the quality of registration possible at a given location and moment.

3.2 Registration Error

Consider four common causes of registration error: translation and orientation inaccuracy on the user's head, and translation and orientation inaccuracy on objects in the world. Figure 2 shows the effects of **head** pose inaccuracy on an augmentation of a cube in a simple virtual world, while Figure 3 shows the registration effects of **object** pose inaccuracy

[1]For more information, see `http://www.microvision.com/nomadexpert/field.html`

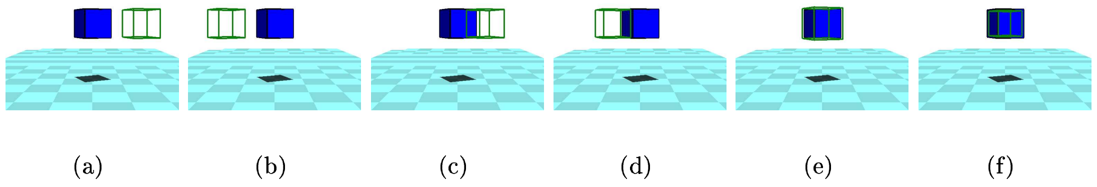


Figure 2: In these figures, a wireframe cube augments an object (represented by the opaque cube). (a) and (b) show the effect of head pose orientation errors of 10 inches to the left and right, and (c)-(f) show the effect of head pose translation errors of 1 inch to the left, right, front and back.

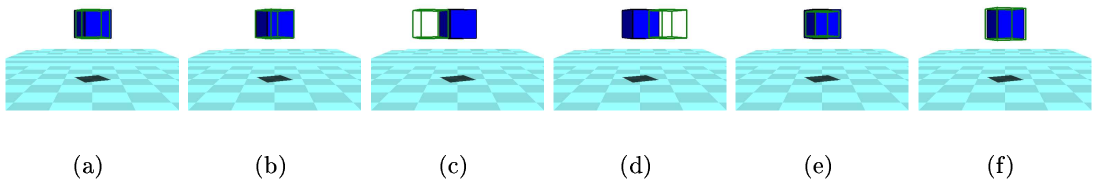


Figure 3: As in Figure 2, a wireframe cube augments an object. (a) and (b) show the effect of object pose orientation errors to the left and right, and (c)-(f) show the effect of object pose translation errors to the left, right, front and back. The magnitude of the error is the same as those in Figure 2.

of the same magnitude. Notice that orientation inaccuracy on the head pose has a much greater effect than any of the other pose inaccuracies illustrated here. Also, notice that translation inaccuracies have their greatest visible effect when they are perpendicular to the viewing direction (compare images 2(c) and 2(d) to 2(e) and 2(f)), and that head and object translation inaccuracies have equal but opposite effects (compare images 2(c) and 2(d) to 3(c) and 3(d)).

Based on these straightforward observations, one could try to create a set of intuitive rules and build a system that reasons about the relative distance and location of an object in relation to the user's head and decides on how to best augment it. Initially, we tried to programatically address the issue of varying registration quality. We created a *Level Of Error* tool [61] that chose the most appropriate augmentation based on the relative position of an object in relation to the virtual camera.

3.3 *Initial Attempt*

Our initial, not so successful attempt at creating adaptive applications made us realize that there is a need for a tool that automatically computes registration error estimates for each object in the scene. Although we were able to show a proof of concept and create an AR application that adaptively modified the way information was displayed as a function of the registration error, we were only able to do it for a single object and generalizing this approach proved very difficult.

Because of the complex interactions between objects and transformations, the solution to the problem of estimating the registration error requires a deep understanding of the sources of uncertainty and the development of an automated framework to automatically compute the estimate of the registration error for each object in real time.

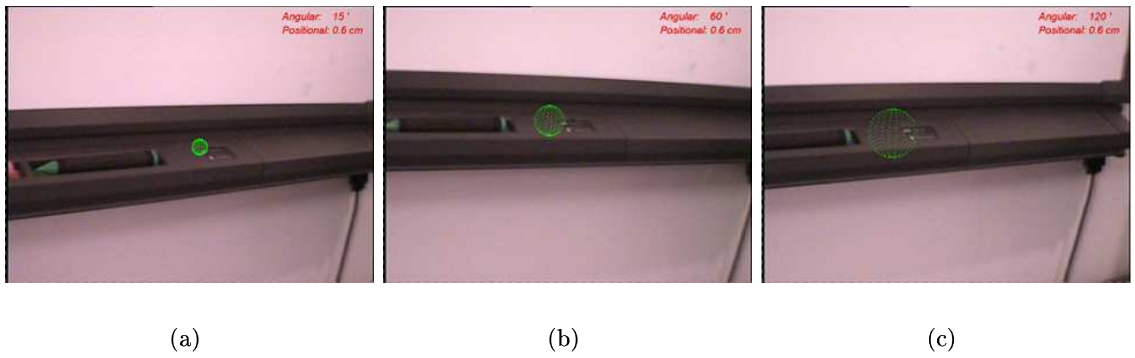
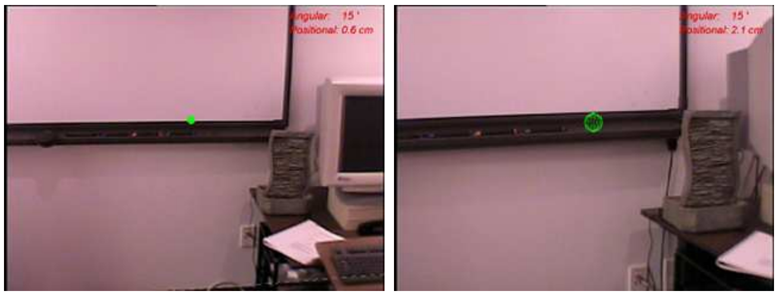


Figure 4: Augmentation with positional inaccuracy = $0.6cm$.

Albeit simple at first, such a heuristic becomes very complex when all possible combinations are taken into account. For instance, when objects are considered, rather than individual points, the visual angle of the registration error is not sufficient to quantify the perceived error. The perceived size and shape of the object also influence the perception. Moreover, the causes of registration error are rarely this simple; more precisely, they combine the effects of tracking, modeling, and calibration inaccuracies. Requiring an application developer to reason over these combined contributions is impractical, especially when they really only care about the end result: the registration error. The goal of this work is to allow the application developer to focus on the end result instead of concentrating on the



(a) (b)

Figure 5: Augmentation with angular inaccuracy = $15'$.

tracker characteristics. The approach described here allows the designer to focus on how best to convey the intent of the augmentation under various levels of registration error, enabling the creation of a new class of Augmented Reality applications that function under a wide range of conditions, namely: *Spatially Adaptive Augmented Reality* applications.

Every application that communicates information to and/or receives information from the user can benefit from this novel approach, be it a maintenance application with images, animations and text, or an outdoor tour guide that labels buildings. Even some AR applications that require perfect registration, such as AR systems for surgical procedures (where millimeter accuracy is required), would profit from using this technique since it can indicate when the system has fallen below a given threshold. However, other AR research areas, such as photorealistic AR, would not improve by using the techniques presented here. This research is geared towards real-time Augmented Reality applications that convey dynamic information to the user and cannot rely on a carefully controlled environment.

In the next two sections, we analyze problems with Augmented Reality applications as a user interface from both the application developer point of view and the user's view point. We then proceed to present our approach: *Spatially Adaptive Augmentations* (section 3.6), and comment on how it addresses both issues.

3.4 Application Developer's Perspective

From the application developer's point of view, Augmented Reality applications are fine tuned to make the best use of the capabilities and limitations of the tracking technology available at the development phase. This is very similar to what happened in the first days of computer graphics, when one would develop applications for specific graphics hardware.

Nowadays, the use of an abstraction layer (provided by libraries such as OpenGL) allows one to develop device independent applications, decoupling the application from the underlying hardware infrastructure. Beyond simply providing device independence, such libraries allow the programmer to query the capabilities of the hardware and adapt to them. Similar kinds of abstractions are needed in Augmented Reality if this technology will ever leave the research laboratories and be put to use in real life situations.

Adaptive Augmented Reality applications allow the application developer to focus on the application itself instead of on the underlying tracking technology.

3.5 User's Perspective

From the point of view of the user, the main concern is whether an Augmented Reality application conveys the information it is supposed to provide. If the registration error gets in the way of the user understanding the intent of the augmentation, the system becomes annoying at best and confusing at worst.

It has been almost ten years since Holloway published his PhD thesis [42] where he identified and analyzed the sources of registration error in Augmented Reality. The main sources of registration error he identified were system latency, tracker inaccuracy, optical distortions, etc. [41]. Despite the fact that the sources of registration error have been known for a decade, the community is still unable to create an AR system where the amount of registration error, among other problems (such as low quality of the HMDs, bulkiness of the hardware, etc.), does not prevent it from being deployed in real situations. Although the causes of the registration error are well known, we are still a long way from creating a real-time mobile AR system.

For the last decade, the main approach to deal with registration error has been to try to eliminate the sources that cause it by employing faster computers and more accurate trackers. The key observation on which this work is based is that, for many AR applications, perfect registration is not necessary. From the user's point of view, what matters is if the system can convey the proper information, even if severe registration error exists.

Adaptive Augmented Reality applications concentrate on the quality of the information presented to allow proper information to be communicated under varying circumstances.

3.6 Spatially Adaptive Augmentations

Spatially Adaptive augmentations address both problems. Developers can create device independent systems that automatically select and display the best possible augmentation to communicate the intent established by the designer, without confusing or misleading the user.

Depending on the task to be implemented, different requirements exist. For instance, an application to support a medical procedure requires much higher accuracy than a computerized tour guide. The common approach has been to determine these requirements beforehand and pick the appropriate technology. The problem with this approach is that any change in the tracker characteristics, due for instance to interference, may render the application unusable. To prevent this, it is necessary to support the creation of Augmented Reality applications that are not intimately tied to the tracking system being used [12], and that are able to degrade gracefully.

In order to deal with the issue presented at section 3.5 (User's Perspective), instead of trying to display the augmentation precisely superimposed on the objects, one can try to modify the augmentations being displayed so that the intent of the augmentation is preserved, but the display causes as little inconvenience to the user as possible [76]. How to best display the augmentations, which augmentation to use, how to transition between the different augmentations, etc. are all subjects for future research. All of these questions are currently difficult to investigate because of the complexity involved in estimating the registration error, a necessity for creating the scenarios that would allow different augmentations techniques to be tested. Once an automated computation of the registration error estimate is available, many different experiments can be conducted and the best technique for augmentations can be determined.

This work provides a method to automatically estimate such registration error [62] based on the static uncertainty in the system and shows how it can be used to create spatially adaptive AR applications. We call this subset of spatially adaptive AR, *Uncertainty Aware* AR applications. The next section (3.7) describes the sources of uncertainty this work deals with. Sections 3.8 and 3.9 address the problem of estimating the registration error given one can model the uncertainty in the system.

3.7 Sources of Uncertainty

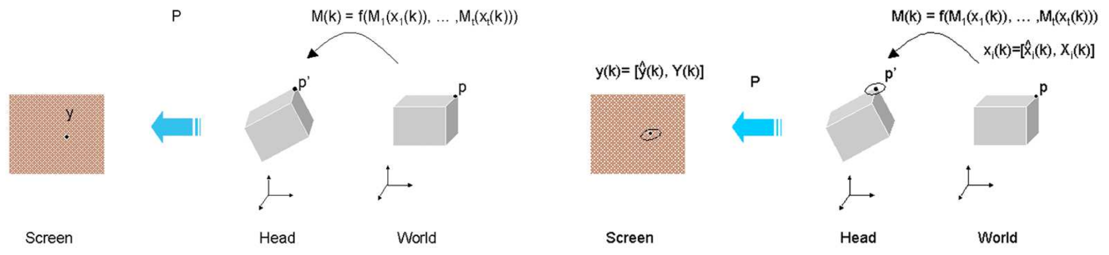
Uncertainty manifests itself as registration error. The sources of uncertainty considered by this work are *tracker inaccuracy*, *calibration imperfections*, and *model imprecision*

3.7.1 Tracking

This is perhaps the best-known and best studied causes of registration error [41]. Any tracking system measures the pose (position and orientation) of the camera using some kind of physical sensor. A large number of research papers (e.g.: [11, 30, 77, 82, 90, 92, 93, 95]) address how to improve tracking and describe different methods based on a variety of sensing technologies (e.g., magnetic, ultrasonic, inertial, computer vision [54], etc.), as well as hybrid systems that combine more than one of these technologies. However, as noted by Welch and Foxlin, there is no "Silver Bullet" that is likely to solve the tracking problem [94]. As a result, all trackers report pose with some kind of error.

3.7.2 Calibration

Related to, but distinct from tracking are the issues of alignment and calibration. Alignment is necessary when the rigid transformation between the sensors and the camera is required. Typically, alignment is carried out using precision machining and can be solved numerically from a set of measurements. However, errors can be introduced. Calibration occurs when it is necessary to calculate the intrinsic parameters of the camera. Many different distortion models with different suites of parameters and automated tools exist. This problem is



(a) Displaying a point. (b) Displaying an uncertain point.

Figure 6: In (a) a point associated with a series of rigid transformations is displayed. In (b) there is uncertainty associated with the same point's position in space.

exacerbated in systems in which precise, factory calibration is unknown (e.g., when using off-the-shelf technologies, such as webcams). Theoretically this is less of an issue for video-see through AR. However, it is an extreme burden on see-through AR systems: calibration is more difficult, as current methods require the user to align objects on the display with those in the physical world (e.g.: [17], SPAAM [87]). In either case, an estimation of the quality of the calibration can be extracted and used by the system.

3.7.3 Modeling

Models are approximations of real objects they are intended to represent. A simple way to create a model is by using a tape measure or ruler to measure the object, while more advanced methods make use of depth scanners and range cameras [73, 74] to automatically generate the models. The mere use of these processes introduces uncertainty into the final models. The creation of automated, large-scale building models, for example using LIDAR data[10, 43] and/or aerial imagery [67], also entails errors, typically in the order of meters.

3.8 Error Estimation

The effects of uncertainty in the form of registration error 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, as illustrated in figure 6. We assume that the position of $\mathbf{p}(k)$ is known perfectly. If there is some uncertainty, due to modeling errors or the fact that $\mathbf{p}(k)$ might be a tracked object, the model can be easily extended to include these additional terms.

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 to registration errors through an erroneous value of $\mathbf{M}(k)$. These errors can be both spatial and temporal. Furthermore, the tracker value used is likely to be predicted from raw measurements using an implicit motion model and an assumed time step length. Both the model and the time step length can be inaccurate.

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 $\mathbf{y}(k)$, $[\hat{\mathbf{y}}(k), \mathbf{Y}(k)]$. To address this problem this work uses the unscented transform [51] to create a region on the screen that corresponds to the point in space, as illustrated in figure 6(b). The unscented transform works by generating values on the sample space (3D space for this work) and reconstructing the transformed function from the values on the target space (the view plane for this work).

The interested reader can obtain more details about the issue of statistical error estimation in Appendix A.

3.9 Error Propagation

The limitation of the methodology introduced in the previous section is that the dimension of the sample space is determined by the number of transformations that are considered in sequence. Such limitation makes it unsuitable for use in scene graphs, a very common data structure largely employed by the graphics community. A recursive implementation of the unscented transform is necessary if one want to use it for data structures such as scene graphs.

As the scene graph is assumed to be composed of arbitrary nodes with arbitrary affine transformations between them, no restrictions are placed on the form of the transformation matrices. Specifically, let $\mathbf{M}_i^j$ be the true relative transformation from node i to node j , $\mathbf{M}_r^n$ is the cumulative transformation matrix from the root node $\mathbf{R}$ to an arbitrary node $\mathbf{N}$. Let $\hat{\mathbf{M}}_i^j$ be the measured relative transformation from node i to node j .

Assuming that the error introduced at a node is independent of the error introduced at preceding nodes, the equation which propagates the error down the scene graph is as follows:

$$\begin{aligned} \hat{\mathbf{M}}_r^i &= \hat{\mathbf{M}}_{i-1}^i \hat{\mathbf{M}}_r^{i-1} \\ \delta \mathbf{M}_r^i &= \mathbf{M}_{i-1}^i \delta \mathbf{M}_r^{i-1} + \delta \mathbf{M}_{i-1}^i \mathbf{M}_r^{i-1}, \end{aligned} \tag{2}$$

More details about how to derive equation 2 are presented in Appendix B.

3.10 Discussion

Choosing to represent the transformation as the more generic form and assuming uncertainty to be additive has three main difficulties:

1. It is more computationally expensive. If one assumed, for example, that the matrix only encoded translation rotation and scale then only 9 or 10 parameters would be required. However, this is at the cost of introducing complicated nonlinear transformations at each node to recover the parameters (and their uncertainties) after a transformation is applied.

2. The representation does not capture the nonlinear constraints which exist between matrix elements. For example, large orientation errors are not simply additive. These could be partially overcome by using more sophisticated models. For example, the error could be treated as being multiplicative and of the form $\mathbf{I} + \delta \mathbf{M}_{i-1}^i$ where $\mathbf{I}$ is

the identity matrix. A recursive relationship exists in this case. The mean term is the same but the error propagation term becomes $\delta\mathbf{M}_r^i = \delta\mathbf{M}_r^{i-1} + \left(\mathbf{M}_r^{i-1}\right)^{-1}\delta\mathbf{M}_{i-1}^i\mathbf{M}_r^{i-1}$. However, any representation is always an approximation and, for this work, we chose the simplest usable approximation, since adding lag to the system would increase the registration error.

3. Only supports well-behaved distributions. Some trackers can report different distributions, for instance, some GPS may generate bimodal distributions when the signal reflects off buildings. Since we are assuming that our distribution can be approximated with a Gaussian, this model will not deal with those cases.

Despite these limitations, we believe the chosen representation is appropriate for the needs of Adaptive Augmented Reality for the following reasons:

- The transformation operations on each node are simple. The transformation consists of a single matrix multiplication which only involves basic arithmetic operations. More elaborate models could be used, but at the risk of introducing lag and thus causing larger registration errors.

- The complexity of specifying nonlinearities (e.g., tracker errors) are only introduced at the nodes where the errors occur.

- If nodes do not introduce their own sources of error, the first term in the error propagation is not needed, reducing complexity even further.

- Any type of matrix operation can be supported. Therefore, not just errors in the location of objects, but also their sizes, the position and projection properties of the viewer, etc., can be handled in a uniform way.

If one is developing a specific application, one can profit from high-level knowledge about the type of uncertainty the system will be subject to in order to tailor-make the representation of the uncertainty. Since one of the goals of this work was to support a large class of applications, a conscious choice was made to use a generic representation. If one decides to modify this model for an specific application, one would have to change the propagation equations (equation 2 on page 29).

3.11 Conclusion

This chapter proposes the use of a simple, yet comprehensive method to deal with static uncertainty inherent to AR systems: the use of uncertainty aware AR, that is, systems that change the input and output characteristics depending on the accuracy of trackers, geometrical models and calibration. Our approach is based on taking into account the system's uncertainties and using that information to estimate the registration error associated to each physical object. It is our belief that the use of uncertainty aware AR will allow the creation of robust applications that will work properly under varying conditions.

CHAPTER IV

SAMPLE ADAPTIVE AR: OSGAR

4.1 Introduction

To experiment with the ideas presented in the previous chapter, an open source scene graph was modified to support the specification of uncertainty at its transformation nodes. These values were then used to estimate the registration error associated with the objects in the scene graph.

OSGAR is built on top of OpenSceneGraph (OSG) [70], an open source, freely available scene graph. The main modification is the addition of a subclass of the original OSG transformation that allows for information about uncertainty to be stored. The consequence is that some internal nodes of the scene graph that can be created using OSGAR will allow the user to inject error in some of its nodes by specifying a covariance value.

The second structural modification is the extension of the group nodes with the addition of subclasses of this node that allow the computation of the estimate to be stored and its projection on the view plane calculated. To handle these new nodes, three new specialized scene graph traversals were implemented (they are called visitors by OSG, and we kept the nomenclature). These three visitors are responsible for optimizing, computing the view independent uncertainty and the view dependent error estimate respectively. Sections 4.3 through 4.6 will explain each component in detail. Figure 7 displays the architectural diagram for OSGAR.

4.1.1 Uncertainty

Uncertainty in OSGAR is represented by adding a covariance matrix to the transformation nodes in the scene graph. Any transformation matrix is considered to be the mean of a probability distribution function (PDF) for that transformation, instead of just a single discrete transformation. If no uncertainty information is specified for a transformation, it

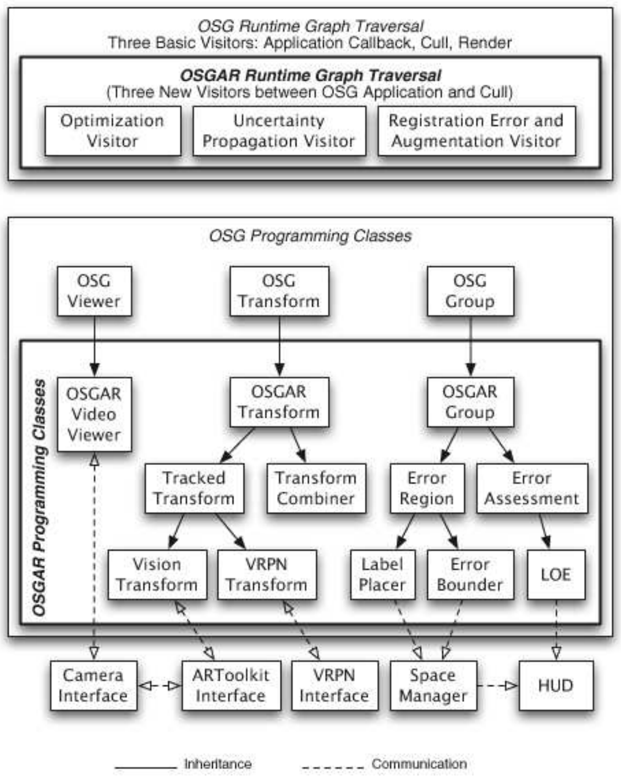


Figure 7: OSGAR programming classes diagram.

is assumed to be exact. The mean of the PDF (i.e., the original transformation) is used for culling, rendering, and so on, as in regular scene graphs. This work assumes that the PDF can be approximated by a Gaussian distribution and represented internally by its mean and variance. The reason for this choice, advantages and disadvantages of this approach as well as how this can be improved was already presented and discussed in section 3.10 (page 29).

4.2 Groups

Uncertainty is injected at transform nodes, propagated down the scene graph by specialized visitor, as described in section 4.5, and stored as matrix estimates at group nodes. The error estimates are stored together with the path from the root associated with that estimate value. Since scene graphs allow multiple paths to reach a node, a list of error estimates, tagged with the corresponding path, is stored on each group node.

4.3 Transforms

Transforms are a subclass of the OpenSceneGraph transformations that adds information about uncertainty. This additional information is stored in the form of a 16x16 matrix. By instantiating this class and specifying the uncertainty value, the application developer can inject uncertainty anywhere in the scene graph, be it due to tracker, model or calibration inaccuracy – identified in section 3.7 as the sources of uncertainty this work supports.

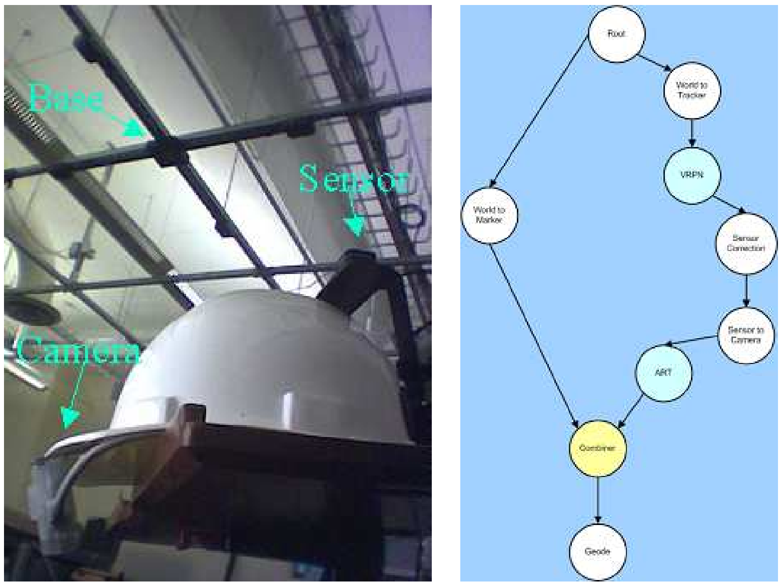
4.3.1 Tracked Transforms

A very useful subclass of a transform is a *TrackedTransform*. TrackedTransforms are specialized classes that will be used for representing the uncertainty associated to trackers (described in Section 4.6.2). They add a boolean that identifies whether or not a tracker is reporting. If a tracker is not reporting, the value of the transformation is set to infinity. The matrices values on Tracked transforms are updated with the values reported by the trackers.

4.4 TransformCombiner

An interesting subclass of a transform is a TransformCombiner. It combines multiple paths on the graph into one unique subtree. TransformCombiners allow an application developer to have more that one tracker define the pose (position and attitude) of an object. This object can be the camera, allowing the user to be tracked by multiple trackers. The TransformCombiner is a subclass of a transform.

The *TransformCombiner* class is designed for situations where multiple pose estimates are available for an object. The programmer can set up a Combiner to automatically combine the pose estimates, can leverage application or tracker specific knowledge to improve the accuracy of the fusion, or can simply choose between the poses. If multiple poses are available for a node in a scene graph, there will be a path from the root to the node for each pose. Normally, if a node is linked to a scene graph via more than one path, the scene graph semantics dictate that it is rendered once for each path, usually at a different location in the 3D world.



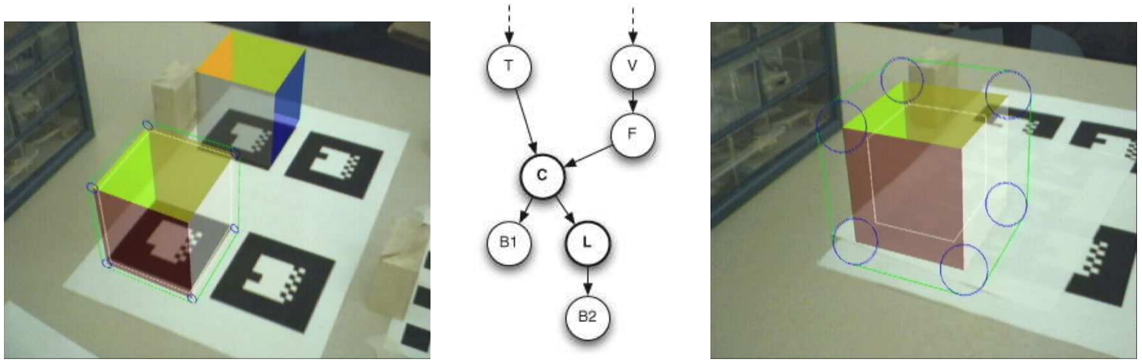
(a) Two tracking application (b) Scene graph

Figure 8: Application with vision and ultrasonic trackers and corresponding scene graph.

The TransformCombiner has different semantics. A programmer-supplied callback function determines which single incoming path to use for the subtree under the Combiner. The function is given the pose estimates for all of the paths leading into the Combiner, and returns a new estimate along with an indication of which incoming path to use for all subsequent passes. The effect is equivalent to the children of the Combiner being attached to that path from the root, and the other paths terminating at the Combiner.

4.4.1 Scenario

The most obvious scenario is when an object is tracked by multiple sensors, and the Combiner should fuse the poses using an approach such as the PDF intersection method proposed by Hoff [39]. However, there are other more mundane scenarios in which the combiner turns out to be a very powerful way of structuring an AR scene graph. For example, consider a user and object that are tracked by some reasonably accurate tracker (such as an Intersense IS600 or an RTK GPS system, both of which give 1-2cm positional accuracy), and the object can also be tracked using the camera on the user's video-mixed AR display.



(a) Using vision (b) (c) Using ultrasound

Figure 9: The combiner chooses between a marker based tracker and a ultra sound tracker.

If the system wants to add augmentations to the object, sensor fusion is not particularly useful — the absolute accuracy of a system such as the ARToolkit is not particularly good, but the registration obtained when using it is quite good. (While both the translation error along the direction of projection and the rotation error are large, the translation error is small perpendicular to the direction of projection.) In this situation, using the vision tracker when it reports, and falling back to the less accurate tracker otherwise, is probably the right thing to do.

For example, in figure 9 the TransformCombiner C chooses which path to cube *B1* (the near cube) to use for each frame, based on the error propagated from T (a receiver for a fixed tracker) and F (a fiducial marker, measured relative to the video camera V). In (a) the fiducial markers are visible, and has much lower estimated error than the fixed tracker, so the branch through F is used. The second cube is attached to C via an LOE L that only displays it when the error is small. In (b) the fiducial is not tracked, so the fixed tracker is used (and the second cube is hidden by the LOE)

Perhaps more interestingly, suppose an object is tracked intermittently (such as by a vision tracker), but the object always remains within a certain area, such as on a desk that it is tethered to by a cable. In most AR prototype systems, when the object is not tracked, its last known position is used. OSGAR supports more systematic solutions to this problem.

One simple solution would be to use a static Transform to specify a second pose for the object, and use a TransformCombiner to merge it with the tracked pose. The Transform would be given a mean in the middle of the desk and a large uncertainty, to capture the full range of the object's possible location. More complex solutions could also be implemented that take into account the time since the object was last tracked, or leverage other application-specific semantics. The Combiner would choose this static transformation when the object is not tracked, but use the tracked path when possible. In this case, the Combiner would propagate the fixed transformation with a large error estimate or the tracked transformation with a very small error estimate. If the programmer simply uses the provided transformation (i.e., the mean of the distribution) to render the augmentation, the resulting display would be nonsensical (the object would jump between the tracked location and the arbitrary "middle of the desk" location specified by the fixed transformation).

However, if the programmer makes use of the error estimation nodes discussed below, the augmentation itself can change automatically when the magnitude of the error changes, and do so in a way that adapts to other aspects of the viewing situation. If the fixed area is, say, the top of a desk and the user is very close to the desk, the registration error (the projection on the 2D display) will be huge, requiring alternative display methods (such as using a 2D inset window containing descriptive text). However, if the user is far from the desk (say, on the other side of a large lab), the magnitude of the registration error *on the 2D display* may be quite small, allowing a 3D augmentation to be used.

By specifying a range of augmentations to use, based on registration error on the 2D display, the system can adapt automatically to these very different situations. This example illustrates the power of OSGAR, and the approach programmers should take when building applications using it.

4.4.2 Implementation

The Optimization Visitor counts the number of paths that enter any Combiner node, and saves the counter in the Combiner. Then, during the view-independent uncertainty propagation traversal, the Combiner collects the error estimates for every path into it. When this visitor enters the Combiner via the final path, it activates the user specified callback function to determine the final pose estimate and the path to use for the subtree during the remaining traversals. Subsequent phases (registration error computation, culling, rendering) only traverse the subtree "under" the Combiner when they arrive along this single path. Currently, we have implemented a *SmallestCovarianceCombiner* that always chooses the smallest covariance, and uses it and its path as the values for the subtree.

Table 1 presents the C++ code used to implement the combiner.

Table 1: Implementation code for the Transformation Combiner.

```
void TransformCombiner::accept(osg::NodeVisitor& nv){
  switch(nv.getVisitorType()) {
    case(osg::NodeVisitor::UPDATE_VISITOR): {
      _visits = 0; _paths++; _estimateList.clear();
      break;
    }

    case(osg::NodeVisitor::CULL_VISITOR):   {
      _paths = 0;
      // only allow traversal of the chosen path
      if(nv.getNodePath().back() != _estimateList[_chosenPath].getParent())
      return;
      break;
    }

    case(osg::NodeVisitor::NODE_VISITOR):        {
      // If this NODE_VISITOR is an ErrorVisitor
      if (dynamic_cast<osgEstimate::ErrorVisitor* > (&nv) != 0)
      {
        osgEstimate::ErrorVisitor* ev =
          dynamic_cast<osgEstimate::ErrorVisitor* > (&nv);
        _visits++;

        // store the local estimate for each path.
        _estimateList.push_back(ParentEstimate(ev->getLocalMatrixEstimate(),
                                               nv.getNodePath().back()));

        // if this is the last path
        if(_visits == _paths)        {
          // the callback will choose which path to use
          if(_fusionCallback.valid()){
            _chosenPath = _fusionCallback->fusionImplementation(_estimateList);
            // replace the estimate on the ErrorVisitor with the chosen one
            ev->replaceLocalMatrixEstimate(
                _estimateList[_chosenPath].getMatrixEstimate());
          }
        }
        else
          ev->getLocalMatrixEstimate().resetP();
      }
      // If this NODE_VISITOR is an AugmentVisitor
      else if (dynamic_cast<osgAR::AugmentVisitor* > (&nv) != 0)
      {
        // only allow traversal of the chosen path
        if(nv.getNodePath().back() != _estimateList[_chosenPath].getParent())
          return;
      }
      break;
    }
  }// end switch

  Transform::accept(nv);
}
```

4.5 Scene Graph Traversals

OSGAR adds three traversals to the original Callback, Cull and Render traversals provided by OSG: Optimizer, view-independent and view-dependent.

4.5.1 Optimizer

To allow the system to run as fast as possible and thus prevent the introductions of lag, causing dynamic registration errors, optimizations are performed on the scene graph, so that unnecessary computations are avoided. This visitor detects which nodes need to be computed, marking the other nodes so that they are not visited by the subsequent visitors. It also precomputes a simpler representation for complex objects, so that less vertices are processed.

To mark which nodes will be visited, it uses a traditional bottom up approach where all nodes are marked as not visited and then the ones that need to be visited are marked as so. If any node has a child that needs to be visited, then it is also marked as requiring to be visited.

To simplify the representations of the objects being used, this visitor computes the 3D convex hull of the objects and store it together with the actual representation of the object. The simpler representation will be used in the computations, while the more complete will be used for rendering.

4.5.2 View Independent

This visitor computes the values that are independent of where the camera is. These values are stored at the group nodes. Since there might be more than one path to a group node, a list of estimates are stored in these nodes. Each estimate is tagged with the path so that the proper one can be retrieved.

4.5.3 View Dependent

This visitor uses the values computed by the previous visitor to compute the final registration error estimate onto the screen.

The view independent value computed for the camera is accumulated with the values compute for each path on the graph and the final result is computed. This computation also takes into account the uncertainty associated with the projection matrix that corresponds to the camera's intrinsic parameters.

For the computation of the registration error estimates, the Unscented Transform [51] was used.

4.6 Additional AR components

OSGAR provides a collection of facilities necessary for AR application development. The most typical facilities being the ability to display the video feed from a camera on the application's background in order to provide video AR applications, and the ability to use tracker reports to update the pose of graphical objects.

4.6.1 Video

In order to also support video-mixed AR experiences, OSGAR provides a *Video Viewer* class that allows a video stream to be texture mapped onto the background of the window. The *Camera Interface* also feeds video to the ARToolkit for fiducial recognition.

4.6.1.1 Video Source

The *video source* class encapsulates the library that allows access to a video camera such as web cameras, digital SLRs, mini DVs, etc. The class stores a video handle that is used to identify the video source. When the source is initialized, a video handle is created. Is is used to get a new frame from the video device. When *getFrame* is called, one has access to the current frame generated by the corresponding device. The device is locked until the frame is release. Multiple video sources can be used simultaneously.

4.6.1.2 Video Feed

The *video feed* class handles the copying of the frame image from the video source into texture memory and use it to texture map a polygon. If multiple video sources are available, the frame is copied from the active video source. This is implemented as a callback in the

OSG video facilities.

4.6.2 Trackers

OSGAR supports both the tracking of fiducial markers and a wide variety of spatial trackers via the VRPN tracker package. Both are handled internally by a centralized *tracker handler* that performs the marker detection on each new video image, polls VRPN once per frame, and updates the values of the associated Tracker Transformations in the scene graph when necessary. Trackers will eventually provide uncertainty estimates with their reports, although we have not finished extending VRPN and the ARToolkit to do this.

The way the *Tracker* class is implemented allows for easy extension to support new classes of trackers that might become available. To do this, one needs to create a subclass of *Tracker* and implement the update() method.

4.6.2.1 Simulated Tracker

Simulated trackers allow the application developer to simulate a tracker. It generates simulated reports, allowing one to debug and try an application without actually having a real tracker connected to the system.

4.6.2.2 Interface to VRPN

VRPN trackers create an interface to UNC's VRPN [89] trackers. It is implemented by creating a VRPN client that registers with VRPN a callback method that knows how to update each appropriate transform. When the Tracker Handler requests a VRPN tracker to update, the VRPN's mainloop() method is called.

4.6.2.3 Interface to Vision-Based Tracker

Vision-based trackers create an interface to marker-based vision trackers. When the Tracker Handler requests an update, it uses the result of the image detection procedure to update the values of the appropriate transforms.

4.6.2.4 Tracker Handler

The TrackerHandler registers all trackers and updates the corresponding TrackedTransform. It is implemented by keeping a list of current trackers supplied by the application developer. It then traverses the list of registered trackers and ask each individual tracker to send updated reports. The new reports are then used to update each tracked transform.

The Tracker Handler is a centralized way to obtain tracker updates. The way this is implemented guarantees optimal performance, since each tracker is only asked to report new updates once, but all sensors associated with the trackers are updated.

4.6.2.5 Tracker Manipulator

Manipulators are used in OpenSceneGraph to allow an application to have different views of the scene. They are used to update a *scene view.* This is equivalent to the virtual camera in OpenGL, but OSG refrains from creating an explicit camera or even using this common nomenclature.

In Augmented Reality, usually the user's head is tracked and the scene needs to be displayed according to the its position. In order to support it, we implemented a *Tracker Manipulator*, that is, a manipulator that has its pose updated by a tracking device.

The *Tracker Manipulator* allows a camera node to be specified. It then computes the compounded transformation from the root to the camera node and uses its inverse to set the value for the OpenGL's modelview matrix.

4.6.3 Head-Up Display

The HUD class is used for displaying 2D augmentations. It is implemented as an orthographic projection attached to the camera position. Any kind of OSG subgraphs can be attached to the HUD. Specifically to this implementation, it is used to support the display of augmentations.

4.6.4 Space Manager

Currently a stand-in for a more powerful space manager, such as that proposed by Bell and Feiner [7]. The class collects the regions created by the Error Region classes into a set of

Hull objects, and uses the HUD to display those that should be visible. Hull objects store all the vertex error ellipses, the inner and outer hulls, the path on the scene graph, and the object's name. The Space Manager is also responsible for positioning labels that were not positioned explicitly by the Label Placers.

4.7 Performance

One of the main concerns of anyone developing a method to estimate the registration error is not to increase it. Since the system lag is one of the main sources of registration error, careful consideration needs to be given to the design of such architecture in order to prevent the introduction of system lag and the consequent worsening of registration.

4.7.1 Pruning

Pruning prevents visitors from traversing subtrees that do not need to have values updated. OSG provides the *node mask* capability to control which subtrees will be processed by visitors. We take advantage of this to prevent visitors form processing subtrees that do need require updating. This is done in a bottom up approach, where a node is marked only if one of its child is marked, as detailed in 4.5.1.

4.7.2 Simpler Models

Some models used for rendering may be very complex and contain thousands or even tens of thousands of vertices and since the algorithm we propose are based on convex hulls, a simpler model generated by computing the 3D convex hull of the original model can be used for computational purposes.

4.7.3 Vertices Removal

Internal vertices are not used for computation of the view dependent error estimation. Thus, we can take advantage of this and compute the 2D convex hull after the view independent computation of the error estimate but before the view dependent error estimate computation to discard the internal vertices.

4.8 Ease of Use

Careful consideration was given to the development of this toolkit to instantiate the concepts presented here, while making it match as close as possible to the semantics and standard interface used by OpenSceneGraph. Any application that already uses OpenSceneGraph can be turned into an AR application with very little alteration.

The modifications are also straight forward, since they follow the Design Patterns commonly used by OpenSceneGraph. As an example of the effort devoted to make it as compliant as possible with OSG, not a single line of code in OSG was changed. Instead, all implementations were done by extending the original classes. This required more work in some instances (for instance, when implementing the support for video on the background), but is expected to allow new versions of OSG to be used with minimal changes.

4.8.1 Video

Table 2 presents an example of how to use the video in the background facility we provide. osgVideo::VideoViewer is a subclass of osg::Viewer that uses a osgVideo::VideoSource to display video in the background. The video source needs to be initialized with a string parameter that specifies device, resolution, and image format.

In the example presented here, the application is requesting a camera connected to device 1 with resolution 640 x 480 reporting at 30 frames a second (lines 13-14). Camera format is 'yuv' and internal image format is 'rgb'. The camera calibration file is *firei0003578.cam.* Lines 16-27 implement the standard OSG calls the perform the call-back, cull and render traversals.

Table 2: Source code for an application that displays video on the background.

```
#include <osgVideo/VideoSource>
#include <osgVideo/VideoViewer>
...
int main( int argc, char **argv )
{
  // use an ArgumentParser object to manage the program arguments.
  osg::ArgumentParser arguments(&argc,argv);

  // construct the viewer.
  osgVideo::VideoViewer viewer(arguments);

  // create the windows.
  viewer.realize(new osgVideo::VideoSource(
    "vc:1 640 30 yuv rgb 1 1", "Data/firei0003578.cam"));
  ...
  while( !viewer.done() )
  {
    // wait for all cull and draw threads to complete.
    viewer.sync();

    // update the scene by traversing it with the the update visitor
    // which will call all node update callbacks and animations.
    viewer.update();

    // fire off the cull and draw traversals of the scene.
    viewer.frame();
  }
}
```

4.8.2 Trackers

OSGAR presents the application developer with simple, yet powerful mechanisms to use trackers. All one has to do is instantiate a VRPN tracker, a ARToolkit style marker-based tracker or even a simulated tracker, and register them with the tracker handler. The following subsections present examples of applications making use of some trackers.

4.8.2.1 Vision-Based Trackers

Table 3 presents an an example of how to use the marker based tracker. In order to register ARToolkit style markers, one needs to specify a file that contains the description of the markers (line 10). After that, the application can request transformations that are associated with each individual marker (line 11). The tracker handler is instantiated (line 13), and the vision based tracker is registered (line 14). Every frame the application request that the trackers be updated (line 22).

Table 3: Source code of an application that uses an ARToolkit tracker.

```
#include <osgTracker/ARTTracker>
#include <osgTracker/TrackerHandler>
...
// Create the ART Tracker
osg::ref_ptr<osgTracker::Tracker> artTracker;
...
int main( int argc, char **argv )
{
  // creates an ARTTracker
  artTracker = new osgTracker::ARTTracker("Data/panelsToolbox.txt");
  artTracker->getTransform(0)->setName("Marker 1");
  // create the tracker handler and register the ART tracker
  osgTracker::TrackerHandler* trackerHandler = new osgTracker::TrackerHandler();
  trackerHandler->registerTracker(artTracker.get());

  while( !viewer.done() )
  {
    // wait for all cull and draw threads to complete.
    viewer.sync();

    // update trackers
    trackerHandler->update();

    // update the scene by traversing it with the the update visitor
    // which will call all node update callbacks and animations.
    viewer.update();

    // fire off the cull and draw traversals of the scene.
    viewer.frame();
  }

}
```

4.8.2.2 VRPN

Table 4 presents an example of how to create a set of transformation that are updated by a VRPN tracker. A VRPN tracker is declared (line 5) and instantiated (lines 10). The VRPN server this client will listen to will be running at the *localhost* and will be named *IS-600* (line 10). Once initialized, sensor can be requested (lines 11-14). A tracker handler is instantiated (line 16) and the vrpnTracker is registered (line 17). Every frame the application request that the trackers be updated (line 24).

Table 4: Source code of an application that uses a VRPN tracker.

```
 1  #include <osgTracker/VRPNTracker>
 2  #include <osgTracker/TrackerHandler>
 3  ...
 4  // Create the VRPN Tracker
 5  osg::ref_ptr<osgTracker::Tracker> vrpnTracker;
 6  ...
 7  int main( int argc, char **argv )
 8  {
 9    // create a VRPNTracker that updates 2 transforms
10    vrpnTracker = new osgTracker::VRPNTracker("IS-600@localhost");
11    if(vrpnTracker.get()->requestSensor(1))
12      vrpnTracker->getTransform(0)->setName("IS-600 1");
13    if(vrpnTracker.get()->requestSensor(2))
14      vrpnTracker->getTransform(1)->setName("IS-600 2");
15    // create the tracker handler and register the VRPN tracker
16    osgTracker::TrackerHandler* trackerHandler = new osgTracker::TrackerHandler();
17    trackerHandler->registerTracker(vrpnTracker.get());
18    while( !viewer.done() )
19    {
20      // wait for all cull and draw threads to complete.
21      viewer.sync();
22
23      // update trackers
24      trackerHandler->update();
25
26      // update the scene by traversing it with the the update visitor
27      // which will call all node update callbacks and animations.
28      viewer.update();
29
30      // fire off the cull and draw traversals of the scene.
31      viewer.frame();
32    }
33  }
```

4.8.2.3 Tracker Manipulator

A tracker manipulator allows a virtual camera, as specified in OpenGL, to be controlled by a tracker device. Table 5 presents an example of the use of the tracker manipulator to update the OpenGL's virtual camera pose based on a tracker. The tracker manipulator is declared and instantiated (lined 6-7), and then added to OSG's cameraManipulator list (line 8). A node in the scene graph is determined to be the camera (line 11). This node is used by the tracker manipulator to update the camera's position (line 12).

Table 5: Source code of an application that uses a Tracker Manipulator.

```
1  #include <osgTracker/TrackerManipulator>
2  ...
3  int main( int argc, char **argv )
4  {
5    // register the camera manipulator.
6    osg::ref_ptr<osgTracker::TrackerManipulator> trackerManipulator =
7             new osgTracker::TrackerManipulator();
8    viewer.addCameraManipulator(trackerManipulator.get());
9    ...
10   vrpnTracker->getTransform(1)->addChild(sensorToCamera);
11   sensorToCamera->addChild(camera.get());
12   trackerManipulator->setNode(sensorToCamera);
13   ...
14   while( !viewer.done() )
15   {
16     // wait for all cull and draw threads to complete.
17     viewer.sync();
18
19     // update trackers
20     trackerHandler->update();
21
22     // update the scene by traversing it with the the update visitor
23     // which will call all node update callbacks and animations.
24     viewer.update();
25
26     // fire off the cull and draw traversals of the scene.
27     viewer.frame();
28   }
29 }
```

4.8.3 Transformation Combiner

As describe in section 4.4, a transformation combiner allows multiple trackers to be used by an application. The combiner automatically computes the proper tracker or combination of trackers to be used in run-time. The example provided in table 6 instantiates a transformation combiner (line 6). The combiner is then added to two differents paths: first to the root (line 7), then to a artTracker sensor (line 8).

Table 6: Source code of an application that uses a Transform Combiner.

```
1  #include <osgTracker/MyCombiner>
2  ...
3  int main( int argc, char **argv )
4  {
5    osg::Group* root = new osg::Group();
6    osgTracker::MyCombiner* myCombiner = new osgTracker::MyCombiner();
7    root->addChild(myCombiner);
8    artTracker->getTransform(0)->addChild(myCombiner);
9    myCombiner->addChild( createShapes());
10   while( !viewer.done() )
11   {
12     // wait for all cull and draw threads to complete.
13     viewer.sync();
14
15     // update trackers
16     trackerHandler->update();
17
18     // update the scene by traversing it with the the update visitor
19     // which will call all node update callbacks and animations.
20     viewer.update();
21
22     // fire off the cull and draw traversals of the scene.
23     viewer.frame();
24   }
25 }
```

4.8.4 Error Estimation

The registration error estimation is divided in three steps: Optimization, view independent traversal and view dependent traversal. The application developer needs to call each of these visitors in sequence.

Optimization is responsible for preventing subtrees that do not need recomputing to be visited by the other two visitor, speeding up the whole process considerably.

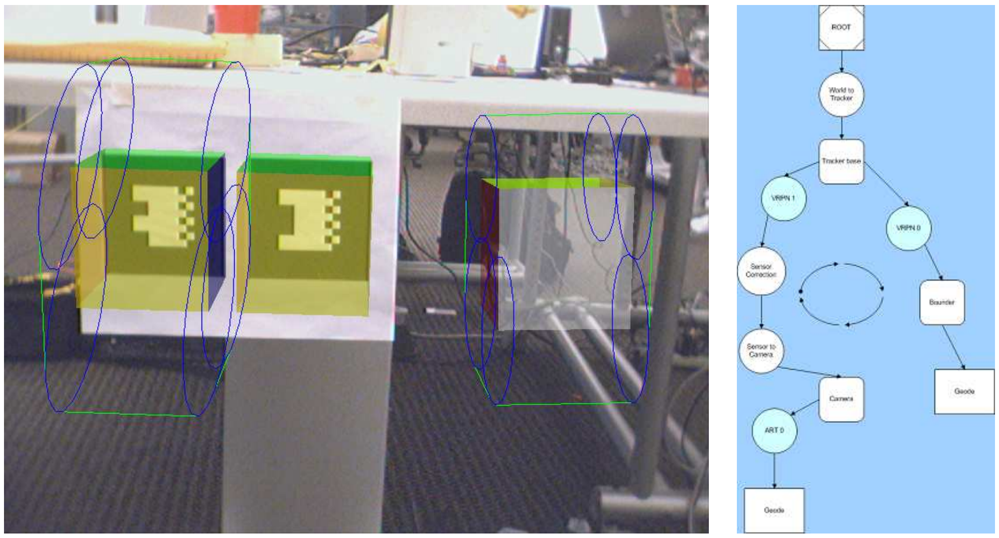
View Independent computes the 3D error estimation associated with individual paths on a scene graph.

View Dependent computes the screen space error region estimate given the 3D view independent estimates computed by the previous traversal.

Table 7 presents an example of how to use the visitor to compute the error estimate for objects in a scene graph. The three visitor are instantiated: the *optimizer visitor* on lines 8-9, the *error visitor* on lines 12-13, and the *augment visitor* on lines 16-17. These three visitors need to be called in order (lines 33-43) after the trackers are updated (line 27) and OSG executes the registered callbacks (line 31).

Table 7: Source code of an application that uses the error estimation mechanism.

```
#include <osgEstimate/ErrorVisitor>
#include <osgEstimate/OptimizerVisitor>
#include <osgAR/AugmentVisitor>
...
int main( int argc, char **argv )
{
  // creates an OptimizerVisitor
  osg::ref_ptr<osgEstimate::OptimizerVisitor> optimizerVisitor =
           new osgEstimate::OptimizerVisitor();

  // creates an errorVisitor
  osg::ref_ptr<osgEstimate::ErrorVisitor> errorVisitor =
           new osgEstimate::ErrorVisitor();

  // creates an AugmentVisitor
  osg::ref_ptr<osgAR::AugmentVisitor> augmentVisitor =
           new osgAR::AugmentVisitor();
  augmentVisitor->setSpaceManager(spaceManager.get());
  augmentVisitor->setCamera(camera.get());
  ...
  while( !viewer.done() )
  {
    // wait for all cull and draw threads to complete.
    viewer.sync();

    // update trackers
    trackerHandler->update();

    // update the scene by traversing it with the the update visitor
    // which will call all node update callbacks and animations.
    viewer.update();

    // prevents the errorVisitor from traversing unecessary nodes
    optimizerVisitor->reset();
    viewer.getSceneData()->accept(*optimizerVisitor);

    // compute the view INDEPENDENT error estimate
    errorVisitor->reset();
    viewer.getSceneData()->accept(*errorVisitor);

    // then the view DEPENDENT estimate
    augmentVisitor->reset(&viewer);
    viewer.getSceneData()->accept(*augmentVisitor);

    // fire off the cull and draw traversals of the scene.
    viewer.frame();
  }
}
```



(a) Augmentations (b) Scene graph

Figure 10: The object directly attached to the camera (on the left) should present smaller registration error than the other object (cube to the right) attached to the world.

4.9 Existing Issues

While developing OSGAR, two issues were identified:

1. Because the error estimate is propagated down the scene graph starting from the root instead of from the camera node, some objects that should have smaller errors were actually showing bigger estimates;

2. Some nodes depend on more than one other node, and sometimes the dependence may reverse from one frame to the next.

4.9.1 Exaggerated Estimates

For the computation of the view dependent values (Section 4.5.3) it is necessary to use the inverse of the view independent uncertainty that corresponds to the camera position to be able to compute the values for each node.

The problem with this approach is that since the uncertainty is positive and symmetric, the inverse will also be positive and the final result will be additive causing the error estimate for some objects to be bigger then they should be (Figure 10), if the camera is not at the root.

For instance, if the camera position is determined by a tracker device, the transformations to compute the camera's position would be:

$$T_{world}^{camera} = T_{world}^{base} \times T_{base}^{tracker} \times T_{tracker}^{camera}, \tag{3}$$

where T_i^j is a transformation from the coordinate system i to j.

Since any of these transformations may have uncertainty associated with it, the uncertainty is computed as:

$$U_{world}^{camera} = U_{world}^{base} \times U_{base}^{tracker} \times U_{tracker}^{camera}, \tag{4}$$

where U_i^j is the covariance matrix that represents the uncertainty from associated with the transformation from the coordinate system i to j.

To compute the registration error of an object in the world, the inverse of the camera's position is compounded with the object's position in the world, rendering the correct result:

$$T_{camera}^{object} = (T_{world}^{base} \times T_{base}^{tracker} \times T_{tracker}^{camera})^{-1} \times T_{world}^{object}, \tag{5}$$

and

$$U_{camera}^{object} = (U_{world}^{base} \times U_{base}^{tracker} \times U_{tracker}^{camera})^{-1} \times U_{world}^{object} \tag{6}$$

If another object is positioned in relation to the base of the tracking system, the correct transformation and error estimate associated with such object are:

$$T_{camera}^{object} = T_{camera}^{tracker} \times T_{tracker}^{base} \times T_{base}^{object} \tag{7}$$

with the uncertainty associated with such transformation equivalent to:

$$U_{camera}^{object} = U_{camera}^{tracker} \times U_{tracker}^{base} \times U_{base}^{object} \tag{8}$$

Nevertheless, if the approach used in scene graphs is applied, the final result is as follows:

$$\begin{aligned} T_{camera}^{object} &= \\ &= (T_{world}^{base} \times T_{base}^{tracker} \times T_{tracker}^{camera})^{-1} \times T_{world}^{base} \times T_{base}^{object} = \\ &= T_{camera}^{tracker} \times T_{tracker}^{base} \times T_{base}^{world} \times T_{world}^{base} \times T_{base}^{object} = \\ &= T_{camera}^{tracker} \times T_{tracker}^{base} \times T_{base}^{object}, \end{aligned} \tag{9}$$

since

$$T^{world}_{base} \times T^{base}_{world} = I, \tag{10}$$

where I is the identity matrix.

Computing the uncertainty using the same approach as before:

$$\begin{aligned} U^{object}_{camera} &= \\ = (U^{base}_{world} \times U^{tracker}_{base} \times U^{camera}_{tracker})^{-1} \times U^{base}_{world} \times U^{object}_{base} &= \\ = U^{tracker}_{camera} \times U^{base}_{tracker} \times U^{world}_{base} \times U^{base}_{world} \times U^{object}_{base} &= \end{aligned} \tag{11}$$

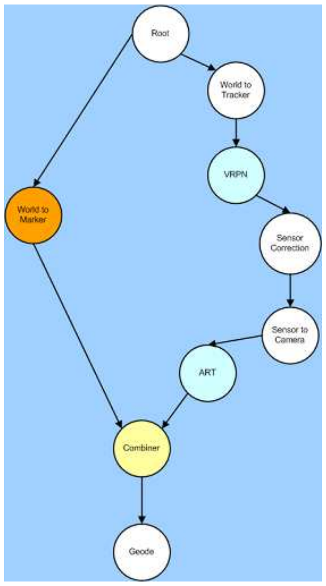
For this example, the computed result using the procedure largely employed in scene graphs yields the correct pose, but the estimated error is bigger then the correct result.

4.9.2 Multiple Dependence

As a side effect of the introduction of the TransformCombiner (4.4) to solve the problem of having multiple trackers detect the pose of an object, some locations in space depend on more than one other coordinate system, creating cycles in the data structure used to represent the world we are rendering (figure 11). These cycles are solved at render time, so that the scene graph algorithms that traverse the DAG to render the scene still work. As a consequence of the cycles and they way they are resolved for the rendering process, some nodes of the data structure may depend on each other, at different times, depending on whether some trackers are reporting or not.

Consider for instance an object that has its location known in relation to the origin of the world that is being used to specify an AR application. This same object has a artificial marker associated to it, so that vision tracking can also be used. To properly take into account the existence of both trackers for this object, one can create a TransformCombiner.

We take advantage of this fact to implement a *Pending Transformation*. The general concept is that of a transformation for which the value is not known at design time but will be "discovered" at run time. The version we implemented only allows the simplest case, that is: when the value can be easily computed by matrix multiplication. A more elaborate implementation would allow for rules to be determined by the application developer



(a) Scene graph

Figure 11: Cycles in the scene graph are exploited by a transformation that will only have an assigned value at run time.

and a system of equations would be solved in order to satisfy a target function by either maximizing of minimizing it.

4.10 Conclusion

This Chapter presented an implementation demonstrating the real-time computation of the registration error estimate, based on statistical methods that propagate the uncertainty of an AR system down the scene graph. Careful consideration was given to make this toolkit both efficient and easy to use. We believe that the use of this toolkit by AR application developers will lead to the creation of AR systems that are robust and work under varying environmental conditions.

CHAPTER V

SUPPORT FOR ADAPTIVE AUGMENTATIONS

5.1 Introduction

The idea of changing the way information is displayed based on the registration error was first introduced by our work presented at the first IEEE and ACM International Symposium on Augmented Reality (ISAR 2000) [61]. Because at that time we lacked an automated way to estimate the registration error for an object, we implemented heuristics that took into account the relative pose of individual objects in relation to the virtual camera to drive the choice of which augmentation to use (section 3.3, page 22). Once we had the capability to estimate the registration error in real time on a per object basis, as described in the previous chapter, we could develop a more comprehensive framework to support adaptive augmentations.

In order to do this, we extended OSG's *group* class to store registration error estimates that were propagated down the scene graph and used such estimates to drive the behavior of the rendering pass. It is interesting to notice that because of the way it works, the unscented transform is suitable to estimate the registration error of individual points in space. This is done by generating a region on the screen that corresponds to where the projection of this point can be. Because we are interested in computing the registration error for an object, not just a single point, we divide the process in two steps:

1. Compute the region on the screen for each vertex of the object, and
2. Aggregate all the regions computed on the previous step to generate a region that corresponds to the object as a whole.

The separation of the computation process in two steps will be exposed for the subclasses to be introduced below. For the *assessment* (section 5.3), individual callbacks can be specified for each step and we provide with examples. For the *region* (section 5.4), we provide as an example two options for the second step.

The next section introduces our extension of the original OSG's group class. This is the base class used to support adaptive augmentation in the presence of uncertainty. Section 5.3 and 5.4 present specialized subclasses that help the programmer deal with uncertainty. Section 5.5 describes the operations of contraction and expansion (used to aggregate the regions) and how to implement them. Section 5.6 shows examples of how to use this framework to create adaptive augmentations and discuss some implementation details.

5.2 Group

The OSGAR Group class is pure abstract, and provides accessor methods to retrieve the view independent uncertainty estimates that are computed by the propagation visitor. Recall from section 4.5 that the registration error visitor computes the 3D convex hull of the geometry in the subtree underneath all Group nodes. The Group nodes use the view-dependent error estimate to compute a 2D view-dependent convex hull of the points in this 3D hull. The points in the 2D hull are used to compute the registration error estimates. The Assessment and Region subclasses provide access to two different representations of the registration error estimate of the objects in the tree under them.

5.3 Assessment Class

The assessment class provide the programmer with a single floating point value representing an assessment of the magnitude of the registration error. The assessment is computed using two user-defined methods:

metric is run on each of the vertices in the 2D hull, giving a floating point value for each. This corresponds to the first necessary step to compute the registration error assessment for each vertex. There are many possible metrics that could be used by an Assessment object, such as the maximum of the main axis of the ellipses, the area of the ellipse, etc.

aggregator is run on the set of vertices and floating point values, and returns the single

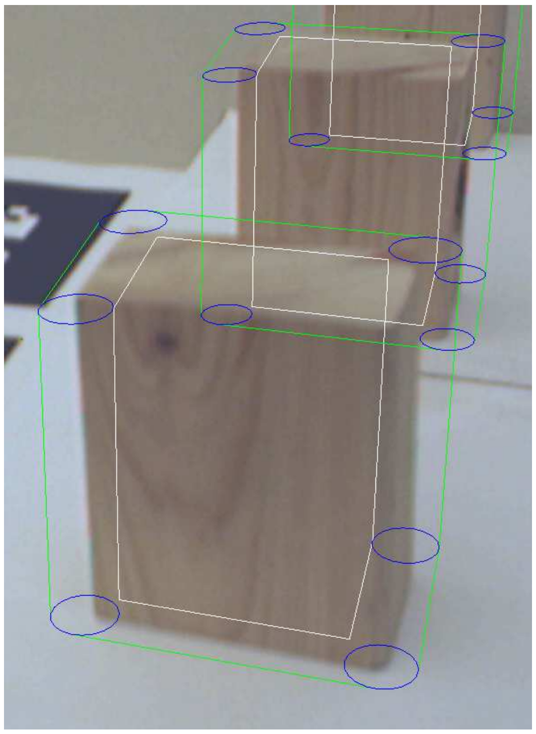


Figure 12: Error Regions

floating point value for the object. This corresponds to the second step necessary to compute the registration error assessment for the whole object. As an aggregator function, one can consider the closest vertex, the average of all vertices, etc.

5.4 Region Class

The Region subclass stores a polygonal representation of the following regions:

error ellipses (displayed in blue) – the ellipses that correspond to the projection of the error estimate for each vertex on the convex hull of the object,

outer region (displayed in green) – the region that the object *might* intersect, and

inner region (displayed in white) – represents the region the object *should* intersect .

The *error ellipse* corresponds to applying step one of the computation of the error estimation, while both the *inner* and *outer* regions correspond to step two, when the vertex related estimates are aggregated to generate on object estimate. Figure 12 shows two physical cubes with all regions displayed.

To perform the computation of these regions [8], we define the operations of *expansion*

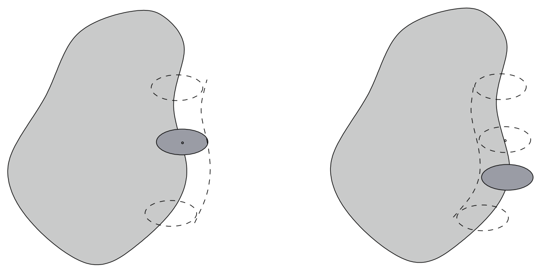


Figure 13: Expansion adds all neighboring points at distance smaller then the radius of the kernel, in a given direction. Contraction removes all points closer then the radius of the kernel.

and *contraction*. These operations are inspired by the morphological operators dilation and erosion [46, 35].

5.5 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 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 13). 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:

5.5.1 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 illustrated in Figure 14). Using polygonal approximations to the error ellipses, the algorithm computes the convex hull of the union of the vertices of these polygons.

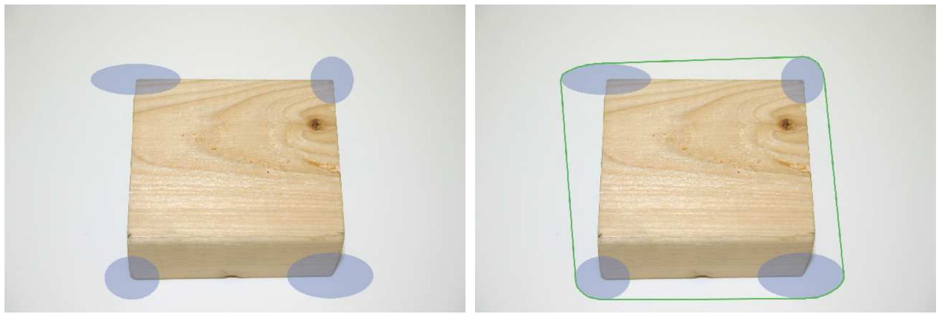

(a) (b)

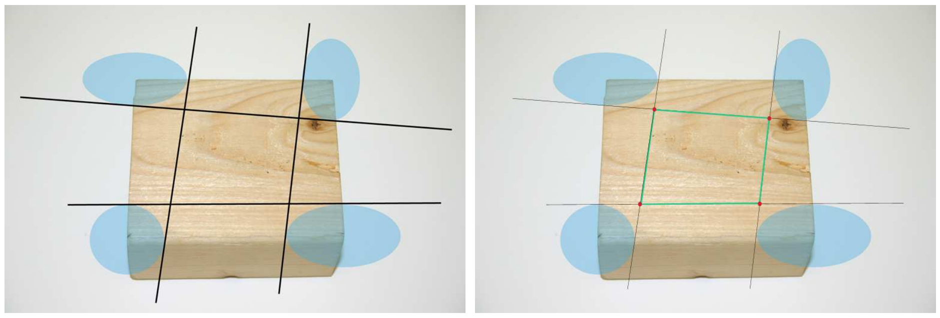
Figure 14: Expansion

Figure 14 shows an illustration of the process. In 14(a), the ellipses correspond to the projection on the screen of the estimated position of each vertex. The convex hull of these ellipses are computed (figure 14(b)) and this defines the expanded convex hull.

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.

5.5.2 Contraction

To contract $CH(V)$, we first determine the interior tangent lines between each pair of adjacent error ellipses (figure 15(a)). Then, for each ellipse, the intersection of these two tangent lines is computed (figure 15(b)). The solution to the contraction operator is the convex hull $CH(VC)$, where VC is the set of these intersection points.



(a) (b)

Figure 15: Contraction

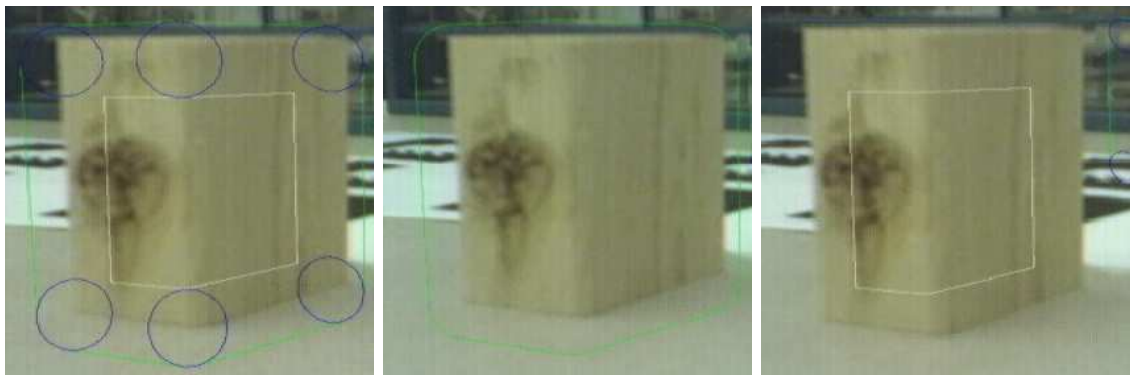
The tangent line between any two error ellipses can be computed straightforwardly. First, the joint convex hull of the polygon approximations of the two ellipses is computed. By maintaining vertex order information, we can immediately find the edge of the convex hull that lies along the tangent line. 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.

5.6 Specialized Group Subclasses

To demonstrate the capabilities of the classes previously introduced and to provide the application developer examples of how to extend the basic *group* class to support the development of adaptive Augmented Reality applications, we have implemented one subclass of Assessment – LOE – and two subclasses of Region – Bounding Regions and Label Placer.

5.6.1 Bounding Regions

Bounding regions are used to display graphical representations of the 2D convex hull. This class can display the vertex ellipses, the inner region, the outer region, or any combination. Figure 16 shows individual examples of each polygonal region computed for a physical cube. It can be used for prototyping, debugging or as a crude highlight for the region an object is expected to occupy.



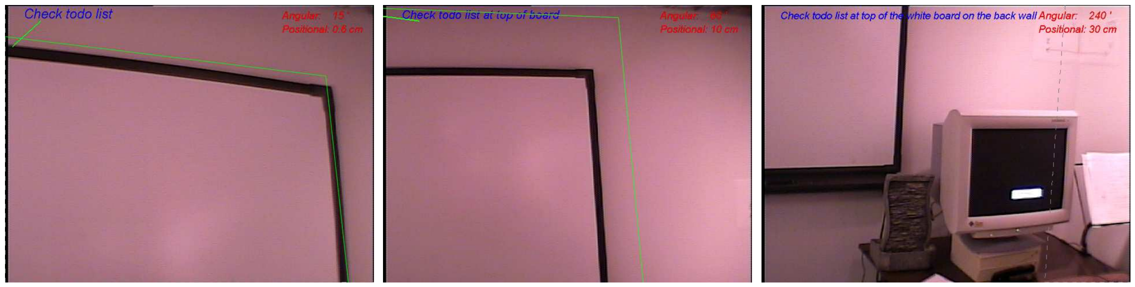
(a) ellipses (b) outer bounds (c) inner bounds

Figure 16: The *error ellipses* are shown in blue (a), the *outer bounds* are shown in green (b), and the *inner bounds* in white (c).

5.6.2 Level of Error (LOE) Switches

The LOE [61] automatically chooses between different children of a group node, allowing the application developer to specify different augmentations corresponding to the same real object. The most appropriate augmentation will be chosen accordingly, depending on the registration error estimate computed in real time.

Figure 17 shows an example where an LOE is used with three different augmentations. When the registration error is below a threshold determined by the application developer, the augmentation is superimposed to the physical object (figure 17(a)). If the registration error estimate is larger and would not allow a unambiguous direct augmentation, some clues related to surrounding area can be show together with the augmentation (figure 17(b)). In the worst case the application would try to display as much information as possible, but



(a) small error (b) medium error (c) large error

Figure 17: Level of Error

rely on the user being able to understand the clues (figure 17(c)) and not try to augment the physical environment, since any attempt by the system would not be correct and the user would be presented with ambiguous or misleading information.

5.6.3 Label Placer

The Label Placer computes where to position the labels for a given object based on the computed inner and outer regions of the model. One has the option to keep the label always inside the object or guarantee that the label will never block the object. This class computes where to place the label to follow one of those constraints. The label placer uses a callback that specifies how to position the labels, for which we have implemented two very simple examples. The first callback positions the label where there is the most space available, computed from the sides of the object to the limits of the screen. The second callback always tries to position the label in this order: right, top, bottom and then left. If the label does not fit on the right side, then it tries to position it at the top, and so on.

In figure 18, the application augments a tool box by displaying labels on top of the drawers. The system tries to display the labels inside the area the drawer is supposed to occupy on the screen (figure 18(a)). If the estimated inner region (section 5.4) of the drawer is not big enough to allow the augmentation to be displayed unambiguously (figure 18(b)), then the labels are displayed on the top left corner of the screen and lead lines are drawn linking each label with the corresponding drawer (figure 18(c)).

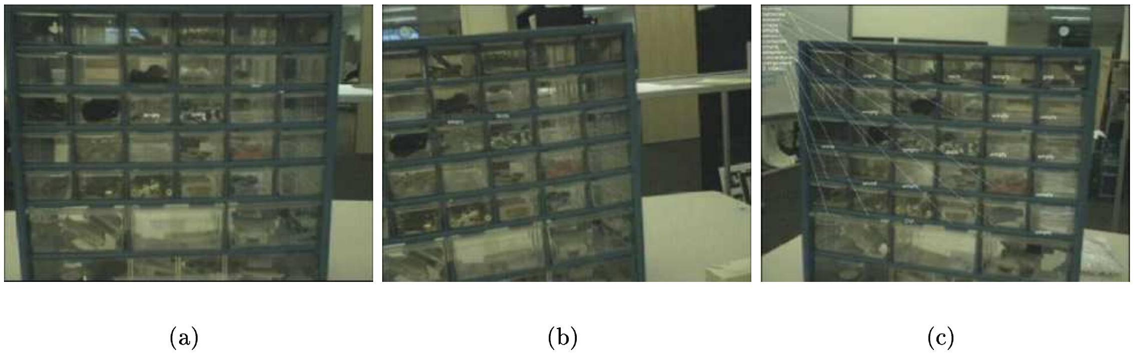

(a) (b) (c)

Figure 18: Label Placer

5.6.4 Implementation Details

Labels that are not superimposed by the *Label Placer* are handled by the *Space Manager*. The *Label Placer* marks labels as either *anchored* or not. Anchored labels have their position specified by the *Label Placer* and are drawn by the *Space Manager* at their previously determined position. Unanchored labels have their position specified by the *Space Manager* itself. The *Space Manager* positions the labels on the left corner of the screen, from top to bottom.

5.7 Conclusion

This chapter demonstrated how to created adaptable output by using the per-object registration error estimates computed by OSGAR, as described in the previous chapter. The three widgets described here were intended as a demonstration on how to make use of OSGAR's capabilities to support adaptive AR, not as a complete set of augmentations that will cover all the needs of adaptive AR applications. Proper techniques and augmentations need to be developed and are subject of research. We claim that this work facilitates the study of which augmentation and technique to be used on each case.

CHAPTER VI

SUPPORT FOR USER INTERACTION

6.1 Introduction

Handling interaction in Augmented Reality (AR) systems is a challenging problem. Many compelling AR systems in military, industrial and entertainment domains envision users operating in mobile 3D environments where they interact with the world, such as by pointing at physical objects to obtain information about them. While AR interaction shares many traits with interaction in other 3D environments, such as virtual reality, it differs in one important way: much of what the user sees and interacts with are physical objects about which the system has imperfect information, especially regarding their geometry and pose (position and orientation) relative to the user.

Errors in tracking the pose and modeling the geometry of objects in the world most obviously manifest themselves to the user in the form of *registration error* between the object and any virtual augmentation being rendered in relation to that object (e.g., a label on a piece of equipment, or a virtual lamp sitting on a physical table). These errors also introduce ambiguity into user interaction, especially in *selection* tasks. Furthermore, registration error is rarely constant, so it can be difficult to adapt to; while every tracking system reports its best-guess approximation of the pose of the object being tracked, the error in the pose estimates is context-dependent and time-varying.

Most research into interaction with AR systems does not address the problem of tracking errors. Piekarski, for example, introduced a paradigm called *working planes* to permit action and construction at a distance [72]. However, although he acknowledges the affects of tracking errors on his system, he does not attempt to mitigate them. Others [26, 56] often avoid the problem by having the user control a virtual on-screen pointer and interact with virtual content that is "attached" to the physical objects. This approach is problematic, as using a screen-fixed pointer to interact with world-fixed objects (which are constantly moving in screen-space) is an extremely difficult task.

As already mentioned throughout this work, the reason AR systems ignore error, or use crude *ad hoc* approximations, is that calculating the impact of uncertainty in an AR system is extremely difficult. The relationship between the graphical display and an object in the world follows a chain of poses from the user's eye out to the world and then back to the object; each pose could be uncertain, but aggregating uncertainty along even simple chains is non-trivial.

In Chapter 4 we presented an AR framework called OSGAR that is capable of propagating error statistics through an arbitrarily complex scene graph, and presenting the programmer with estimates of registration error between points in the physical world and the corresponding locations in the scene graph. Chapter 5 described how OSGAR can be used to dynamically change the state of the display [14]. In this chapter we show how to leverage registration error estimates, such as those provided by OSGAR, to calculate selection probabilities for objects in AR systems [15]. This approach provides a simple yet robust foundation for creating interaction techniques in AR environments, and can feed directly into the multi-modal approaches described by other authors [69, 52].

6.2 Motivating Example

Part of our research interest is in supporting AR applications in a industrial environment [13]. In a production line, workers want to be able to "add" some graphical annotations to the moving assembly line to communicate with other employees information about the tasks they are performing, and to interact with machines and robots

In this scenario, some areas on the factory are instrumented with high accuracy trackers, while other areas only have low accuracy trackers. And the application will have to be robust enough to work properly in either area and transition between them seamlessly as the worker moves freely back and forth. In a routine maintenance task, the crew finds out that one of the pieces of the machine needs to be replaced. They tell the manager about it and augment the machine with information about the task that needs to be performed as well as a forecast about how long it will take until the machine is back into production.

The manager decides to change the schedule of other machines to compensate for the repair time so that it does not impact the total month production quota. She starts to verify the work load of other machines and change the annotations.

This scenario raises interesting questions: How can one guarantee that another user who views the information, possibly from a different view point, will see the augmentation where the first person intended to? If there are moving parts, can the user interact with them all the time? What happens when the user cannot access a part she wanted to augment because it is behind another object? How can the system guide the user to a position where interaction is possible? How to figure out if interaction is possible?

6.3 Classification

We classify the possible interaction techniques to be used based on the following factors:

- **Relative location.** The target of the interaction can be defined either in world space, body space or in screen space. The interaction tactics and the effects on user perception are different in each case. People can more easily point to objects that are stationary in relation to either the world or their body than to objects that have a fixed position on the screen. Even worse are objects that move around in screen space. This may happen, for instance, as a consequence of using a space manager that tries to guarantee that physical objects are not occluded by the computer augmentations.

- **Pointer stability.** The temporal behavior of the representation used to give the user feedback about the selection location can influence how easy it is for the user to select objects. In AR, due to tracker inaccuracy it is very likely that the actual position of the selection pointer differs from the intended position. Depending on the characteristics of the device used, this difference can be constant and the user may be able to compensate for such difference. Or it may be chaotic and vary from frame to frame. In this case, even with feedback, the user will not be able to compensate for the erratic behavior of the tracking device.

- **Pointer area.** Depending on the accuracy of the device used to track the location of the selection apparatus, the corresponding area on the viewing plane can vary from a couple of pixels to dozens of pixels. The size of the selection pointer has an influence on the ability of the user selecting only one object. If the area is too large and there are many objects to be selected, it is possible that wherever the user positions the pointing device there will be more than one object that can be selected, creating ambiguity.

We have interest in supporting interaction in the presence of chaotic, large area pointers in both screen and world space, for two reasons: (1) these are the situations that will arise in real environments, and (2) these are the most challenging scenarios.

6.4 Uncertainty and Selection in AR

As in many others, our selection technique is based on the detection of collision between the selection pointer and objects in the scene. Given the registration error estimate regions in figure 16 (page 64), the basic selection algorithm is straightforward:

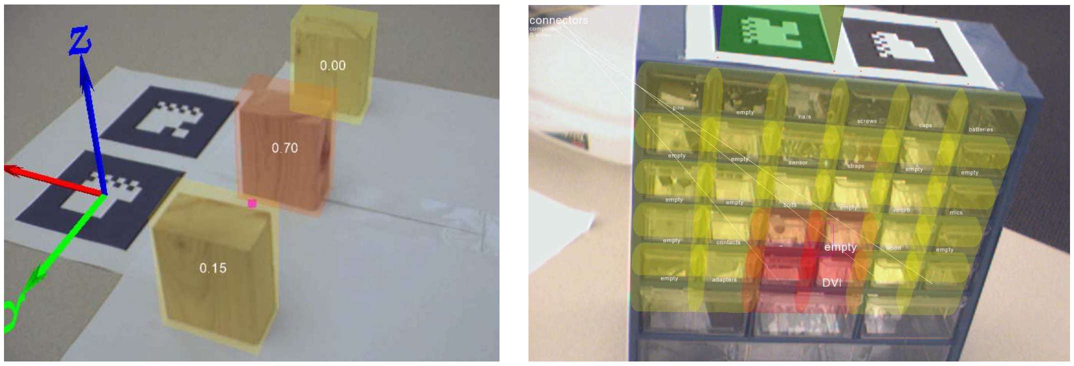
1. **Intersect** the (magenta) region around the pointer with the regions around objects (green hulls) or points (blue ellipsoids) of interest, and
2. **Return** a list of all hits with additional useful information about each hit target, including the relative depth and likelihood of this being the target.

The obvious metric for the "likelihood" is the area of intersection between the error estimation regions, but (as we discuss below) we give programmers the flexibility to define their own evaluation callbacks.

6.5 Selection Implementation

The central component of the selection implementation in OSGAR is the computation of the likelihood that an object is the selection target. The algorithm is implemented in the OSGAR *space manager* (described in 4.6.4) in the following way:

- each augmentation (*Entity*) stores a probability of it being the selection.



(a) (b)

Figure 19: As the user moves the selection pointer (a hollow red box), the computed likelihood of an object being selected changes. Reddish objects are possible selections.

- a *Picker* object is added to represent the 2D projection of instances of a 3D *Pointer* in the scene graph.
- a subclass of SpaceManager, called SpaceManagerInput, handles the computation of the selection probability of each instance of *Entity*.

Figure 19 shows two examples of the computation of the likelihood that an object has been selected. In both examples, the error regions (the green hulls from figure 12, on page 60) are displayed as translucent filled regions whose colors vary from yellow (likelihood=0, not selected) to red (likelihood=1). In 19(a), the likelihood values (computing using *SpaceManagerInputDistance* below, rather than intersection area) are displayed. In 19(b), we use a toolbox with many drawers because it has a regular structure where the presence of uncertainty in the system may cause the error regions around the drawers to overlap from multiple viewpoints.

In order to demonstrate how such an architecture might be extended by an application developer, we will discuss two subclasses of *SpaceManagerInput*. Both assume that the entity is larger than the picker. Both of these algorithms operate in a manner analogous to OpenGL picking — they will return the probability of selection for all objects under the picker, whether they are visible or not. However, the space manager also calculates the depth to the selection point and this is used to determine order relationships.

6.5.1 Based on Area

This class computes the likelihood based on the area of the intersection of the polygonal representations of both the pointer and the boundary associated with each *Entity* that exists in the SpaceManager. This measure is equivalent to calculating the probability that a pixel in the picker intersects with a pixel in the entity, assuming uniform distributions within the areas.

The computation of the intersection between the pointer's area and the *Entity* is done by using Toussaint's algorithm [85] to compute the intersection of convex polygons in linear time.

$$likelihood = \frac{area(picker_{boundary} \cap entity_{boundary})}{area(picker_{boundary})}.$$

6.5.2 Based on Distance

Another method for computing the likelihood is to estimate the distance between the center of the picker and the center of each *Entity* instance.

The probability that an object has been selected is given by the ratio of the Euclidean distance in screen space between the centers of the two regions, and the size of the error region of the entity. The bigger the distance, the less like likely an object is to be selected.

$$likelihood = \max\left[0, 1 - \left(\frac{distance}{entity_{radius}}\right)\right]$$

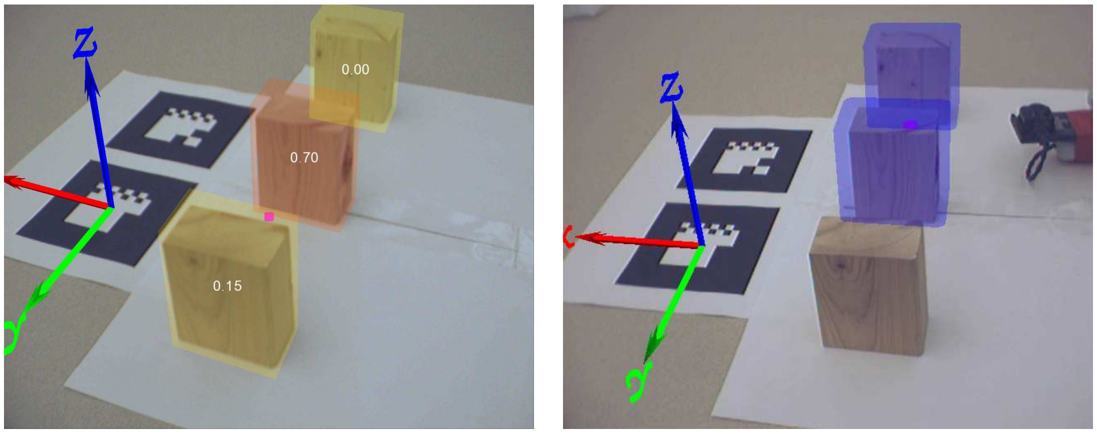
where $distance = picker_{center} - entity_{center}$.

6.6 *Interaction Example*

To demonstrate how our system can support interaction in the presence of dynamically changing environmental characteristics, we present three examples of more complex selection techniques. If the system detects that the user can unambiguously select the object, it could use the direct selection technique. In case there may be ambiguity, the indirect selection technique could be used. We also present a combined version of the two, called two-step selection. Each of these techniques will be described in more details in the following sections.

6.6.1 Direct selection

Direct selection is useful when individual targets can be selected unambiguously. Based on the infrastructure we presented before, this technique can be implemented by choosing the object with the highest likelihood or closest depth value. In figure 20, the highest likelihood is highlighted the darkest red. However, when there is too much overlap, such as in figure 20(b), direct selection becomes more difficult. In that case, other interaction techniques need to be employed.

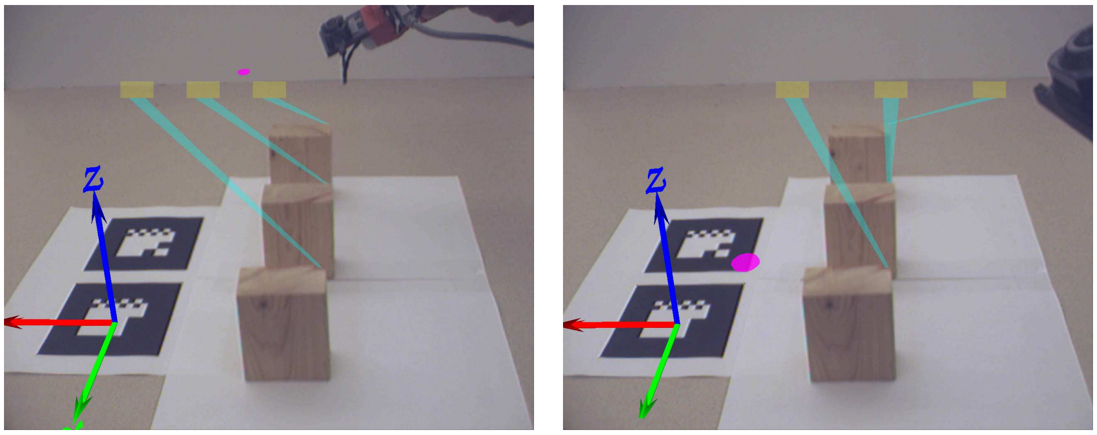


(a) likelihoods (b) selections

Figure 20: When direct selection is possible, the system reports the likelihood of each object been selected.

6.6.2 Indirect selection

Indirect selection can be used when the user cannot unambiguously select a single object. Our implementation of the indirect selection strategy is illustrated in figures 21(a) and 21(b). Whenever more than one object is selected a set of target labels are created, one for each possibly selected object. The targets are positioned automatically by the system (in this case, horizontally) depending on the magnitude of the error estimate of the pointer; the labels automatically spread themselves out further as the pointer error estimate increases. Assuming that the objects remain on the screen, the user can select a single object unambiguously, since this indirect selection mechanism guarantees that the location on the screen



(a) (b)

Figure 21: When the location of the selection targets can be controlled by the application, positioning the interaction elements in well spaced locations allows the user to interact with the system unambiguously.

of the targets will allow each target to be selected individually.

6.6.3 Two Step selection

Many AR demonstration system allow a user to select a single object and obtain more information about it. These applications are usually crafted so ambiguity is minimized. But in many cases, the selection of a single unambiguous object is not possible and if no support for techniques that allow selection in the presence of uncertainty are provided, the user will select the wrong object or will be prevented from selecting objects all together.

For instance, in figure 22(a), only one object is present. For the user, even under heavy uncertainty, it is very easy to select this object. Most AR systems demonstrated by research groups implement this scenario, since it prevents the application developer from having to deal with the annoying problem of ambiguity in the selection process. A more interesting situation is presented in figure 22(b) where the user is supposed to select one unique object from a set of many, very close objects.

For this case we propose the use of the two-step-selection technique:

1. As a first step, most of the objects are eliminated by using the direct selection mechanism.

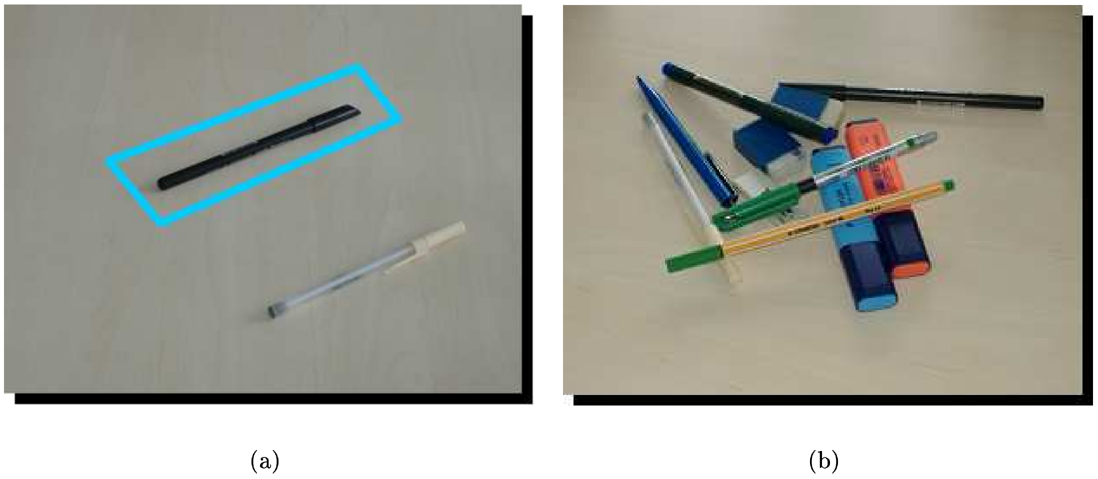


Figure 22: In the presence of uncertainty, selection is an easy task when only one object is present 22(a). But if multiple objects are available 22(b), the user may have a harder time making sure that the right object was selected.

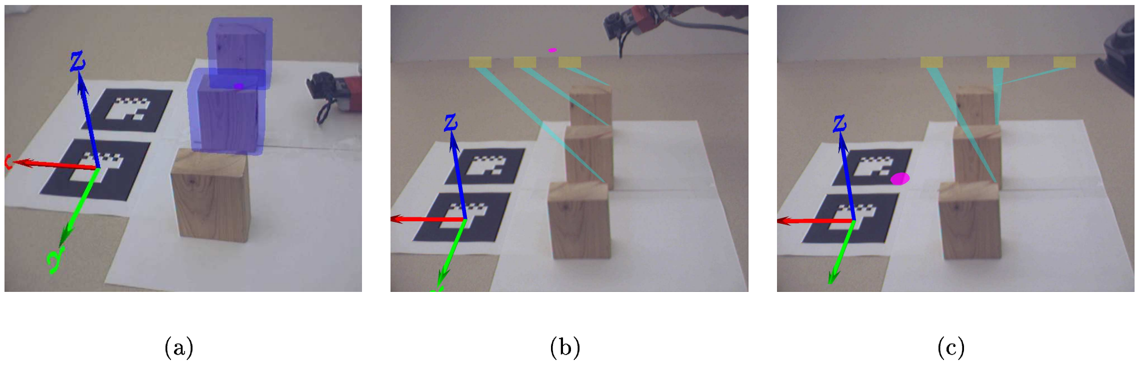


Figure 23: Two step selection is implemented using direct and indirect selection.

2. Then the indirect selection mechanism is used to create on-screen targets corresponding to the objects selected on step one. Now the user can unambiguously select one unique object.

Figure 23 shows an example of how this technique can be used.

6.7 Ease of Use

In the same way as the implementation of OSGAR (Chapter 4) and the adaptive augmentations (Chapter 3), careful consideration was given to make it as easy to use as possible. An application that wants to take advantage of the support for interaction we provide needs to specify a subclass of *Space Manager Input* and a pointer. In the example provided here (see table 8) a *pointer* is attached to a vrpn transformation representing a wand the user manipulates to select physical objects. The *checkIntersections()* method in the space manager detects which objects the pointer intersects with.

Table 8: Source code of an application that uses the proposed selection technique.

```
#include <osgEstimate/ErrorVisitor>
#include <osgAR/AugmentVisitor>
#include <osgView/SpaceManagerInputArea>
#include <osgInput/Pointer>
...
int main( int argc, char **argv )
{
  // add a pointer
  osgInput::Pointer* pointer = new osgInput::Pointer();
  pointer->setName("pointer");
  osg::ref_ptr<osg::Geode> pointerGeode = new osg::Geode();
  pointerGeode->addDrawable(createBlock(osg::Vec3( 0.3f, 0.2f, 0.0f),
                                        osg::Vec3( 0.002f, 0.002f, 0.002f)));
  pointer->addChild(pointerGeode.get());
  vrpnTracker->getTransform(0)->addChild( pointer);
  ...
  // creates the SpaceManager
  osg::ref_ptr<osgView::SpaceManagerInputArea> spaceManager =
           new osgView::SpaceManagerInputArea;
  spaceManager->createHUD(root);
  // creates an errorVisitor
  osg::ref_ptr<osgEstimate::ErrorVisitor> errorVisitor =
           new osgEstimate::ErrorVisitor();
  // creates an AugmentVisitor
  osg::ref_ptr<osgAR::AugmentVisitor> augmentVisitor =
           new osgAR::AugmentVisitor();
  augmentVisitor->setSpaceManager(spaceManager.get());
  ...
  while( !viewer.done() )
  {
    ...
    // handles user input
    spaceManager->checkIntersections();
    // display the objects on the HUD
    spaceManager->display(&viewer);
    ...
  }
}
```

6.8 Discussion

As with selection in other libraries (for example, OpenGL's picking mechanism), the programmer can use this basic information to implement specific interaction techniques. However, if a programmer is to implement reasonable techniques, they must carefully consider the probabilistic nature of the results. For example, the relationships between the reported poses of objects can change in substantial ways without the physical world changing significantly, as illustrated in figure 24. One physical configuration (solid magenta pointer and yellow physical object) can result in two very different values reported to the system (hollow magenta pointer and translucent yellow object with dashed outline). The true locations of the object and pointer fall within the error estimate (hollow blue circle and rounded green rectangle), but in (a) the physical pointer is far from the object and in (b) it is within the object. In both cases the error estimate regions intersect.

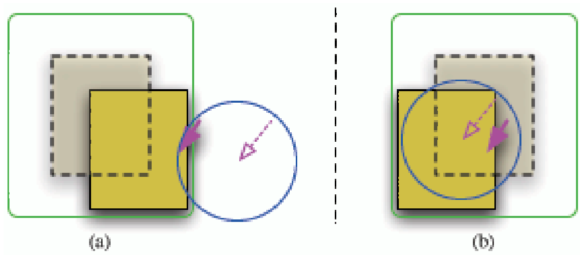


Figure 24: The relationship between the magenta pointer and the yellow object is the same on both sides of the figure, but relationship between the reported values is significantly different.

Furthermore, assuming likelihood is based on the area of intersection, no matter how small the intersection region is, there is a chance that this region is the one the user intended to select. Therefore, no hits should be dismissed based solely on area of intersection. Conversely, the larger the region used for picking, the smaller the probability of an intersection region of a given size being the intended target (since the total probability over the picking region is constant). Henceforth, it may be suitable to use different interaction metaphors as the uncertainty around the pointer increases.

The approach can be adjusted depending on the application and the hardware. For example, if the pointer is stable and the error changes slowly within its error region then an application could attach a virtual cursor to the tracker and use a small error region for hit testing, even if the error region around the pointer is large. In other cases, where the error might change with high frequency (such as if a hand tremor was being modeled to support a disabled user), the error region for the cursor would need to be larger.

For instance, suppose an application developer uses the direct selection method to allow a user to select one object among a set of five objects. Lets suppose that the uncertainty in the system causes the pointer to occupy a larger area on the screen than the space between the objects. It is very likely that more than one object will be returned when the user tries to select one of the objects. The näive approach would be to use the likelihood value provided by this function to order the selected objects and pick the one with the highest value. For the sake of argument, lets suppose that two objects were returned with values 0.64 and 0.59 respectively. Ignoring the latter is not the correct approach. More suitable techniques would be to use the two step selection process presented in the previous section or preselect the most likely, but provide a fast and easy way for the user to select another candidate.

6.9 Conclusion

This chapter introduced an approach to implementing selection in AR interfaces that is robust, predictable and flexible. We present these selection mechanisms as building blocks, analogous to OpenGL picking, not as full-fledged interaction techniques. The determination of which are the appropriate techniques for selection, manipulation and navigation (if it makes sense in an AR context) is object of further research. We claim that this work sets the foundations on top of which that research can take place.

CHAPTER VII

OSGAR-BASED PROJECTS

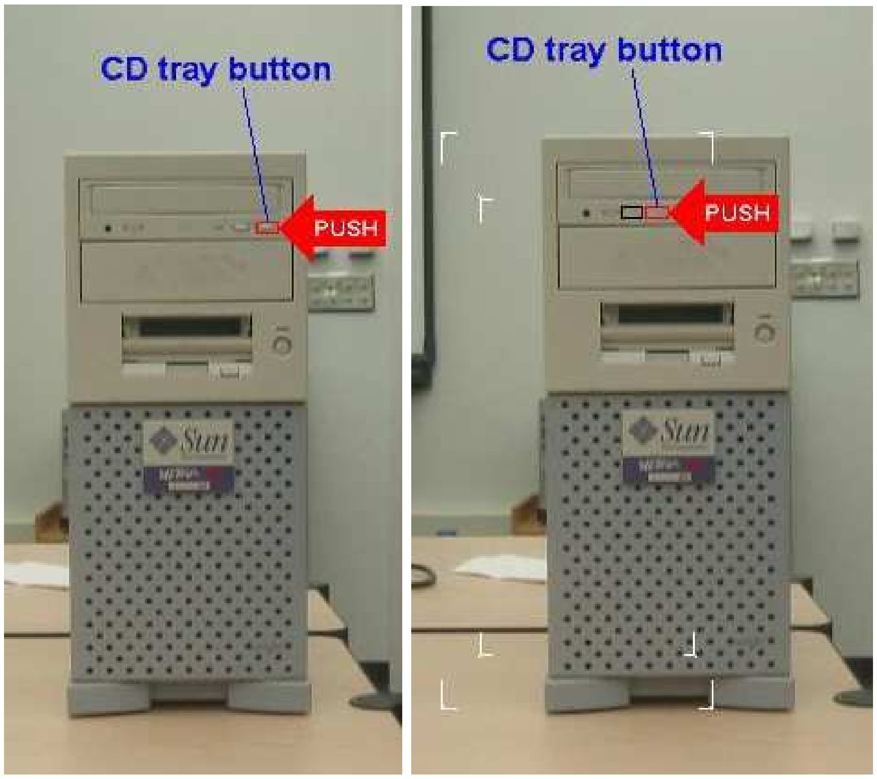
7.1 Introduction

This chapter reports on two other projects that are related to the work presented here. The first project has also been developed at the Augmented Environments Laboratory at the Georgia Institute of Technology and relies on the infrastructure provided by this work to function. The goal of that project is to determine how to best convey the intent of the augmentation to the user. The second project is a simple tracker implemented on a model train set. It demonstrates the ability of the toolkit to function with ever changing trackers and we hope that it can stimulate commercial tracker manufacturers to expose the tracker report accuracy values. It also demonstrates how the ideas introduced by this work can support the development of applications that work with very low accuracy trackers.

7.2 Intent Based Augmentations

The goal of AIBAS (an Adaptive Intent-Based Augmentation System), also developed in the Augmented Environments Lab at the Georgia Institute of Technology, is to understand how semantic knowledge of a scene can be leveraged to simplify the creation of AR applications that work well in real-world situations with 'good enough' tracking and registration. AIBAS depends on OSGAR to provide the programmer with a continuous estimate of registration error. It also relies on the system having sufficient knowledge of the intent of the augmentation and on the viewer's ability to make sense of ambiguous situations when provided with sufficient context. This project studies and develops new strategies for providing the correct amount and most appropriate displays of visual context.

Figure 25 presents an example of how such a system would work. The system augments a desktop computer. These images were created to show a fictional scenario that walks a novice computer repair person through a sequence of operations necessary to repair a



(a) High accuracy: only the intended object is augmented

(b) Low accuracy: contextual information is added to the augmentation

Figure 25: Adaptive Intent-Based Augmentation System

computer. These images show the part of the task where the repair person needs to insert a specific CD into the CD tray. In the case of negligible registration error (Figure 25(a)), the system would display the arrow with the instruction right on top of the model the system is using. The button would also be highlighted and a call-out line with proper labels would be displayed. In this situation it is clear which button needs to be pushed. But, as the registration error increases (Figure 25(b)), the scene becomes less and less clear. There are two buttons located close to each other and they are identical in shape and size. In order for the repair person to differentiate between the two buttons, more contextual information needs to be provided. In this case, the white lines that correspond to the edges of the computer and the surrounding button the may be confused with the target button are drawn.

AIBAS and OSGAR are very closely related. For spatially adaptive AR applications to work properly in real situations, application developers need to be able to carefully craft the augmentations to be used for each situation. AIBAS studies how to properly do it. The studies developed in the context of the AIBAS projects are made possible by the infrastructure provided by OSGAR. We claim that the real-time estimation of the registration error provided by OSGAR is necessary for the experiments and research performed as part of the AIBAS project.

7.3 Accuracy Reporting Tracker

The estimation of the registration error relies on having access to the accuracy of the measurements reported by the trackers, besides the values for the models and calibration. On most of our implementations, we used the value provided by the tracker manufacturers for the worst case as a measurement of the accuracy of the tracker's reports. Since we did not have access to real estimates of the accuracies, we hard coded a value at the time the application was initialized and did not change it during the time the application ran. For instance, for the Intersense's IS-600 tracker we used 1/4 of an inch for positional accuracy since that it the value the manual specifies [44].

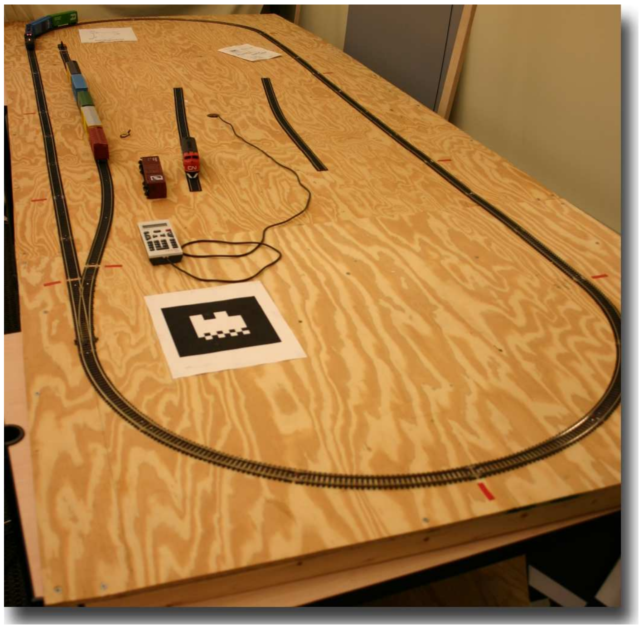
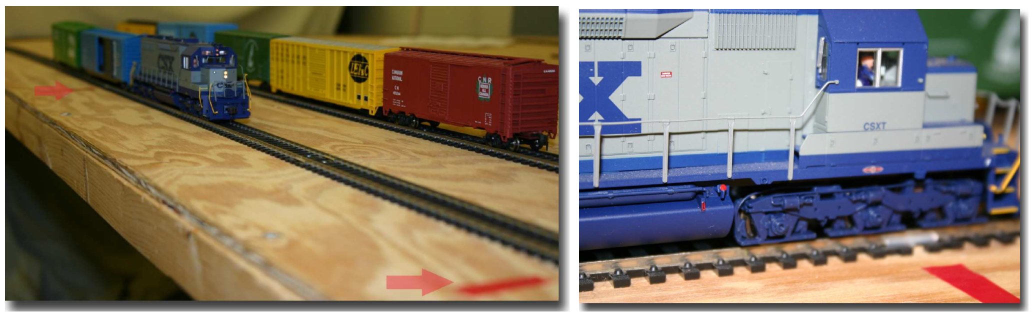


Figure 26: The model train set

For the system proposed in this work to reach its full potential, trackers need to report their inaccuracy in real time. To demonstrate the advantages of having this, we implemented a low accuracy tracker on a model train set. A picture of the model train set used in this experiment is shown in figure 26.

7.3.1 The Train Set

For this experiment we used the *Digital Plus by Lenz* [25] model-railway control system. In our set up, the train track is composed of ten separate sections. One of such sections is depicted in figure 27(a).

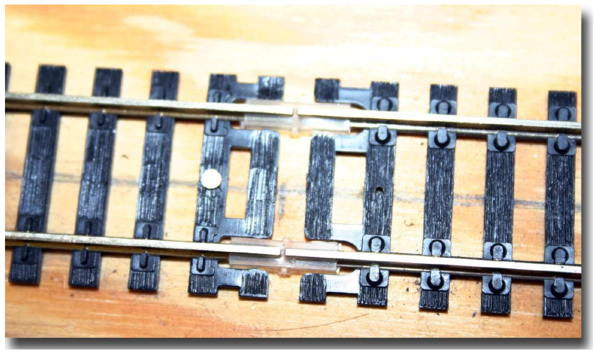


(a) Track section (b) Locomotive crosses sections

Figure 27: The train set is composed of ten sections

Each section is isolated from the others, as illustrated in figure 28(a). Occupancy sensors, shown in picture 28(b), are used to determined which track section the locomotive is drawing current from at a given moment. Using this set up, the system always knows which track section the locomotive is occupying, but it does not know exactly where on that section the locomotive is. There is only one moment when the system knows exactly where the locomotive is: at the connection of adjacent track sections (figure 27(b)).

The model train set can be connected via serial port to a PC and the information about which track section the locomotive is occupying is sent to the computer. In our case, the locomotive would be occupying one of the ten sections. A software development kit (Kamind's Train Tools [53]) allows one to have access to the occupancy sensors and know which one is reporting that the track section is been occupied at a given moment.


(a) Track isolation

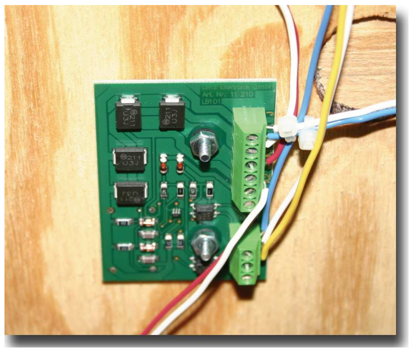

(b) Occupation sensor

Figure 28: Model train components

7.3.2 A Train Tracker

The train tracker was implemented as a VRPN [89] server. To the original VRPN pose messages, we added uncertainty messages that allow the tracker to report the accuracy of the pose reports. To the standard **vrpn_TRACKERCB** data structure we add a custom **vrpn_TRACKERCOVCB** data structure that contains the covariance information. In order to have access to this information, a client needs to register a callback in the same way done to receive the tracker reports.

Our simple tracker works by estimating the position of the train on the rail track based on the time elapsed since the locomotive crossed from one adjacent section to the other. The train only has one degree of freedom, since it moves on the track all the time. We take advantage of this fact and use a parametric representation to describe each individual track segment parameterized by t.

Simple physics equations were used to implement the dead reckoning mechanism briefly described above. The following equation was used for the computation of the train's position along the model railway track:

$$S = S_0 + V_0 t + \frac{\alpha}{2} t^2, \tag{12}$$

where S is the current position, S_0 is the beginning of the track segment, V_0 is the velocity when the locomotive crossed from the previous section to the current section. We simplify our implementation by considering that acceleration is 0, since we do not measure acceleration. In this case, equation 12 is simplified to:

$$S = S_0 + V_0 t \tag{13}$$

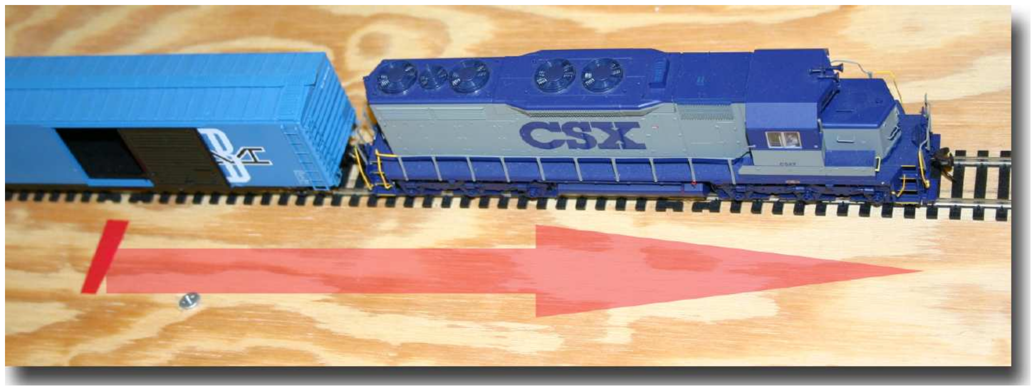
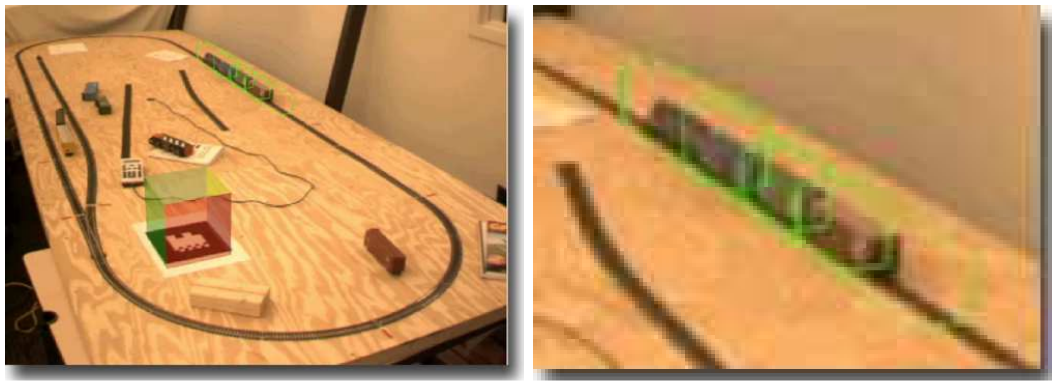


Figure 29: The uncertainty increases as the train progresses

Because we do not measure acceleration, instantaneous changes of speed are not detected. To take this fact into account, our estimate of the accuracy decreases as time passes by. At the very moment the locomotive crosses from one section to the other, we compute the average velocity at which the train traveled the previous section, and the uncertainty is set to 0. As the locomotive progresses towards the end of the current track section, the uncertainty increases (Figure 29). The value we use to increase the uncertainty is given by the difference between the maximum speed the train can travel and the locomotive's average speed during the previous section. The tracker treats the locomotive and each individual train car as individual sensors. An initialization script is used to specify how many cars a locomotive is pulling as well as their size and distance to the previous car.

7.3.3 Registration Error Estimates

We created an application that uses the error regions class introduced in section 5.6 to display the region where the train can be. Our tracker was configured to report the pose of the locomotive plus three train cars for a total of four sensors. One error region was created for each sensor, that is: four error regions were used – one for the locomotive and one for each of the three train cars.



(a) Track View (b) Detail View

Figure 30: The augmented model train set

Figure 30 shows the application that displays the error estimates using the *bounding regions* described in section 5.6 (page 63). The artificial marker is used solely to detect the pose of the table in relation to the camera. The location of the train in relation to the track is computed using the methods described here.

7.3.4 Low Accuracy Trackers

Besides demonstrating that accuracy reporting trackers are necessary for the further development of Uncertainty Aware AR, this project also demonstrates how other AR applications that could be easily deployed in environments with low accuracy tracking mechanisms. Many other environments can be modeled exactly as the train set presented here. That is: there is a location where it is possible to obtain high accuracy information about the object(s) and/or people the system is interested in tracking, but over the remaining space the system has to infer the the information about the objects and/or people, usually based on the last available piece of information. For instance, in a factory, a production line moves, usually at a constant speed, along a path known *a priori*, instrumentation could be deployed to allow AR applications to be developed without the need of other tracker technologies (magnetic, optical, ...) that could suffer interference in such environment. Also, the same model could be used for buildings, streets, etc. where there is one location where high accuracy knowledge exists and dead reckoning could be used to estimate the location on the remaining areas.

7.4 Conclusion

The development of an infrastructure that automatically computes estimates of the registration error gives rise to relevant questions that need to be addressed before one can deploy spatially adaptive AR applications. This chapter described two projects that could be pursued once the basic mechanisms to estimate the registration error were in place. The fist one (AIBAS) investigates what the proper augmentations are for different error estimate values. The second addresses the need for the availability of real time measurements of the accuracy of the tracker's reports in order for OSGAR to adapt to the current characteristics of trackers. It also demonstrates how low accuracy trackers could be created by instrumenting a location, assembly line, etc. The existence of these projects is a consequence of the availability of OSGAR and thus help validate the claim made by the thesis statement.

CHAPTER VIII

FUTURE WORK AND CONCLUSION

8.1 Future Work

The future work proposed here can be divided in two big groups: improvements the system itself and other necessary developments to allow spatially adaptive AR applications to be developed and deployed in real situation. The modifications to the system include different error estimation and propagation mechanisms (Section 8.1.1), separate data structures for the error estimates and the scene graph itself (Section 8.1.3), and other interfaces to expose error estimates (Section 8.1.2) of different models of uncertainty.

While the work presented here establishes the foundation for the creation of spatially adaptive AR systems, one still needs to know how the information provided as a result of the error estimates can be leveraged to develop the applications. Among them, one needs to determine what are the proper augmentations (Section 8.1.4) that should be used for each estimate value, as well what are the proper user interaction techniques (Section 8.1.5).

8.1.1 Uncertainty Model, Error Estimation and Propagation

For this work, we modeled the uncertainty as a Gaussian distribution and assumed that there was no correlation between the different positional and angular values. Although this is sufficient for a reasonable large number of applications, we realize that this model imposes some limitations and that it could be expanded to include other types of distributions that more closely represent the behavior of some tracking devices. Thus, the model presented here can be modified in two ways: (1) one can consider using a multi modal distribution and (2) some high level knowledge about the application can be used to specify the correlation. For instance, GPS trackers may present distributions that are multi-modal when it is close to a building and the radio signal bounces off the walls.

Since one of the goals of this work was to support a large class of applications, a conscious choice was made to use a generic representation. However, if one is developing a specific application, one can profit from high level knowledge about the type of uncertainty the system will be subject to in order to tailor made the representation of the uncertainty. If one decides to modify this model for an specific application, among other things, one would have to change the propagation equations (equation 2 presented on page 29).

8.1.2 Interface to the Estimates

If other systems model the uncertainty in a different way, as suggested above, there will be a need to expose the error estimates using a modified interface. For instance, in the case of multi-modal distributions, the result of the computation of the registration error regions on the screen would be a set of polygonal lines. This fact would create some challenges in relation to how the system exposed this result to the application developer. For instance, if two polygons A and B in the set intersect, what would be the appropriate behavior?

- return one polygon corresponding to the union $(A \cup B)$.
- return the two original polygons and ignore the fact that they intersect?
- return the union $(A \cap B)$ and the intersection $(A \cup B)$?

The same questions are relevant in the case of the *assessment* class 5.3 described on page 59: Should the toolkit return multiple values? Should it return the average? Perhaps the maximum? These are all interesting questions that beg for further investigation. Nonetheless, the fact is that a change in the models used to represent uncertainty will require the computed error estimates to be exposed in a format compatible with the chosen model.

8.1.3 Implementation

The current implementation integrates the computation of the registration error estimate with the scene graph. The rational behind this approach is that:

1. OSG is a well implemented scene graph, with a growing community that vigorously develops it. This makes OSG a very good choice to be used as a basis for the development of AR applications.

2. It provided for a compact, simpler implementation that does not require any data copy. This also made the implementation of the toolkit easier, faster and less error prone, since OSG has been tested by many developers, whereas a piece of code developed exclusively for this project would be initially tested by a handful of users.

Despite these valuable arguments, integrating the computation with the scene graph may hinder a more broad adoption of this project by other groups. The reason being that there is relevant resistance to the adoption of new code that are not complementary to the legacy systems developed in these groups.

As a future work, and for others considering re-implementing the system described in this work, I suggest the separation of the computation from any scene graph. This could be done, for instance by replicating the relevant nodes and data, and exposing an interface that would allow any scene graph to interact with this data structure. The algorithm would then work as follows:

- At the beginning of every frame, the computation data structure would have its values updated, usually based on tracker reports.
- The computation would be performed on this data structure, that is: the view independent and the view dependent traversals would be executed.
- The computed values would be copied back to the scene graph.

This way, anyone interested in using the mechanism for the estimation of the registration error would only have to understand the interface for copying data back and forth and implement the relevant methods associated with those.

8.1.4 Adaptive Augmentations

In order to create augmented reality applications that work in uncontrolled environments, one needs two things: (1) real-time estimates of the registration error, and (2) carefully designed augmentations that make use of such estimates. This work presents how one can use the uncertainty information to accomplish the first part. For the second part, user experiments need to be carried out to determine what the appropriate augmentations are. Among others, the following questions need to be answered: What are the necessary amount of context information that need to be provided for small, medium, and large amounts of error? How to strike a balance between between using as much computer generated information as necessary to convey the intent of the augmentations and preventing the graphics from blocking the physical world information? What are the relevant contextual information? What is the proper way to display graphics? Wire frames? Realistic rendering? Transparent objects? How to properly switch between the different levels of augmentations? For instance, how to transition between small and large error? Fade-in and Fade-out effects? Make use of animations? If so, what are the appropriate animations to be used? Could animation be used even if there is no transitions, solely to draw the user's attention to a relevant contextual information? As already mentioned in section 7.2, another project in the AEL lab (AIBAS) is investigating some of these relevant questions.

8.1.5 User Interaction

Chapter 6 starts to address the issue on how the real-time estimation of the registration error can be leveraged to support user interaction in the presence of uncertainty. We presented a implementation that supports selection, but only start to hint on how this could be used for the implementation of real techniques. Since the result of the algorithm presented in this work is a set of objects that could possibly be selected by the user, there is a need to develop proper techniques that allow the user to determine which object from the set is the one she is actually selecting.

In this direction, we proposed the two-step selection mechanism (Section 6.6 on page 72), where first all the objects that are clearly not candidates are eliminated, then virtual targets associated with the candidate objects are displayed on the screen. The separation between the virtual targets guarantee that there will be no ambiguity for the second step of the selection process. Another option would be for the system to pick the one it deems more likely for the user to select but also present alternate options. The user would be offered the possibility to easily switch between the object the system picked and the alternate objects.

Besides selection, further research is necessary on how to support manipulation and navigation (if applicable to AR). Such research would require the creation of new techniques and the set up of multiple user studies. There is a need to compare and contrast the effectiveness of the created interaction techniques in the presence of varying accuracy levels, since one technique that is effective in a high accuracy environment may not be as effective in the presence of greater levels of uncertainty.

8.2 Conclusion

This work question the often implicit assumption that perfect registration is necessary for Augmented Reality applications to function properly. Instead, we propose that carefully designed augmentations and interactions techniques be used to create AR systems that work under varying environment characteristics and are decoupled from the tracker technologies.

We believe that the reason no Augmented Reality interfaces currently deal correctly with uncertainty and registration error is that estimating the registration error is a very complex task. We have addressed this problem by creating a toolkit (OSGAR) that allows application developers to easily create applications that adapt in the presence of changing uncertainty, and we hope that this work will support the creation of new interfaces and interaction techniques for Augmented Reality systems.

It is our belief that the use of such estimates in the design of user interfaces for AR systems will result in the development of applications that adapt to the characteristics of the surrounding environment and are robust to changes in (and even failure of) the devices used to track the user and the objects.

8.2.1 Contributions

These are the main contributions of this work:

1. Introduce the notion of using Spatially Adaptive Augmented Reality to support a new class of AR applications.

2. Make use of registration error estimates as a basis for Spatially Adaptive Augmented Reality applications.

3. Establish a theoretical framework that supports the creation of Uncertainty Aware Augmented Reality applications.

4. Present a toolkit that computes a registration error estimate and makes it available to the application developer.

5. Demonstrate how the registration error estimates can be used to modify the augmentations in real time.

6. Demonstrate how the registration error estimates can be used in real time to support user interaction.

8.2.2 System Sustainability

One very interesting consequence of using the technology presented here is that it allows new trackers and equipments to be added to systems without any changes to the software architecture, thus providing long term sustainability to the system as a whole, since it can be improved by adding small components. This characteristic make this approach economically attractive for a large set of applications.

For instance, in the context of a car repair shop that uses AR to support maintenance, it is reasonable to expect that newer cars will provide improved ways to display information and more accurate tracking mechanisms. Since the toolkit proposed here supports the adaptation to different conditions, it would allow cars produced at different years to be maintained by a mechanic using the same system.

In extreme case such as undersea exploration or outer space missions where the cost of repairing or replacing the actual trackers may be almost prohibitive, the architecture presented here would naturally allow the incremental deployment of new technologies as they become available. The ability to expand the current system or replace individual components that have failed, without having to discard the whole physical infrastructure, would help make such enterprises economically viable.

APPENDIX A

STATISTICAL ERROR ESTIMATION

A.1 Introduction

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. This work assumes that the position of $\mathbf{p}(k)$ is known perfectly. If there is some uncertainty, due to modeling errors or the fact that $\mathbf{p}(k)$ might be a tracked object, the model can be easily extended to include these additional terms. 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 $\mathbf{M}(k)$,

$$\mathbf{p}'(k) = \mathbf{M}(k)\,\mathbf{p}(k)\,. \tag{14}$$

$\mathbf{M}(k)$ 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. Furthermore, the tracker value used is likely to be predicted from raw measurements using an implicit motion model and an assumed time step length. Both the model and the time step length can be inaccurate.

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 mobile augmented reality systems where 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:

$$\mathbf{P}(k) = \begin{bmatrix} \frac{f}{a} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} & \frac{2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{15}$$

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

$$\mathbf{y}'(k) = \mathbf{P}(k)\,\mathbf{p}'(k). \tag{16}$$

Therefore, the pixel coordinate on the viewport is

$$\mathbf{y}(k) = \mathbf{y}_0(k) + \frac{1}{2y'_4}\begin{bmatrix} w(y'_1 + y'_4) \\ h(y'_2 + y'_4) \end{bmatrix} \tag{17}$$

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 14, 16 and 17, the projection operator can be written as

$$\mathbf{y}(k) = \mathbf{h}[\mathbf{p}(k), \mathbf{x}(k), \mathbf{u}(k)] \tag{18}$$

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 $\mathbf{X}(k)$.

It should be noted that it is *not* necessary to assume that $\mathbf{x}(k)$ is zero-mean, uncorrelated, or Gaussian distributed. These assumptions can be made because no attempt is being made to fuse data over multiple time steps. If a recursive estimator such as a Kalman filter was used, the zero-mean and uncorrelated assumptions are critical. However, the often-heralded assumption of Gaussianity is entirely superfluous and is a consequence of the Bayesian interpretation of the Kalman filter. Rather, we require the weaker condition that $[\hat{\mathbf{x}}(k), \mathbf{X}(k)]$ is a *conservative* estimate of $\mathbf{x}(k)$. In other words:

$$\mathbf{X}(k) - \mathrm{E}\left[(\mathbf{x}(k) - \hat{\mathbf{x}}(k))(\mathbf{x}(k) - \hat{\mathbf{x}}(k))^T\right] \geq \mathbf{0} \tag{19}$$

where $\geq \mathbf{0}$ means that the left hand side is positive semidefinite (none of its eigenvalues are negative). Intuitively, this condition means that we do not underestimate the mean squared error in any direction. 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 $\mathbf{y}(k)$, $[\hat{\mathbf{y}}(k), \mathbf{Y}(k)]$.

A.2 Limitations of Linearization

Because Equations 16 and 17 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

$$\hat{\mathbf{y}}(k) = \mathbf{h}\left[\mathbf{p}(k), \hat{\mathbf{x}}(k), \mathbf{u}(k)\right] \tag{20}$$

$$\mathbf{Y}(k) = \boldsymbol{\nabla}_x \mathbf{h}\, \mathbf{X}(k)\, \boldsymbol{\nabla}_x^T \mathbf{h} \tag{21}$$

where $\boldsymbol{\nabla}_x\mathbf{h}$ and $\boldsymbol{\nabla}_v\mathbf{h}$ 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 [60], 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.*

A.3 The Unscented Transformation

The unscented transformation (UT) works on the principle that *it is easier to approximate a Gaussian distribution than it is to approximate an arbitrary nonlinear function or transformation* [88, 50]. 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 computed 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:

$$\begin{aligned} \boldsymbol{\mathcal{X}}_0 &= \hat{\mathbf{x}}(k) & W_0 &= \kappa/(n+\kappa) \\ \boldsymbol{\mathcal{X}}_i &= \hat{\mathbf{x}}(k) + \left(\sqrt{(n+\kappa)\mathbf{X}(k)}\right)_i & W_i &= 1/2(n+\kappa) \\ \boldsymbol{\mathcal{X}}_{i+n} &= \hat{\mathbf{x}}(k) - \left(\sqrt{(n+\kappa)\mathbf{X}(k)}\right)_i & W_{i+n} &= 1/2(n+\kappa) \end{aligned} \tag{22}$$

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,

$$\hat{\mathbf{y}}(k) = \sum_{i=0}^{2n} W_i \boldsymbol{\mathcal{Y}}_i. \tag{23}$$

3. The covariance is the weighted outer product of the transformed points,

$$\mathbf{Y}(k) = \sum_{i=0}^{2n} W_i \left\{\boldsymbol{\mathcal{Y}}_i - \hat{\mathbf{y}}(k)\right\} \left\{\boldsymbol{\mathcal{Y}}_i - \hat{\mathbf{y}}(k)\right\}^T. \tag{24}$$

In this application, the optimal choice of κ is $\kappa = 3 - n$ [50].

A.4 Advantages of The Unscented Transformation

The unscented transform has three important advantages:

1. 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 [87] 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.

2. It can be proved that this algorithm is more accurate than linearization [50]. 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.

3. By changing the point set, it is possible to extend or change the statistical information which is propagated. In [47] and [49], the author showed that the skew and kurtosis (third and fourth order moments) can be propagated through a simple change of points. This is potentially important because transformations using spherical Cartesian coordinates are sufficiently nonlinear that fourth order statistics must be utilized to calculate the mean and covariance for errors of only a few degrees [49]. Recent work has also proved that, for n dimensions, only $n+2$ sigma points are required.

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. For example, the parameters that make up $[\hat{\mathbf{x}}(k), \mathbf{X}(k)]$ for a given coordinate system in a typical scenegraph can be cached, as can many of the intermediate matrices needed to build the complete composite transformation matrix. In addition, notice that the values for the $2n+1$ weighted n-dimensional points $\boldsymbol{\mathcal{X}}_i$ given by Equation 22 are computed by adding $\hat{\mathbf{x}}(k)$ (the means of the current values of the trackers) to a set of values determined from the covariance $\mathbf{X}(k)$ of the current tracker readings. In the general case, the covariance may change every frame; for example, the "black box" for one or more of the parameters might be based on a complex model of a tracker that computes a tight error estimation based on factors such as the location of the sensor. However, if simple models are used, or if some of the values and their associated errors are static (as might happen, for example, when a building is coarsely positioned on a map), the covariance may never change, or change infrequently. In this case, the values used to compute the $\boldsymbol{\mathcal{X}}_i$ do not need to be recomputed.

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.

APPENDIX B

RECURSIVE ERROR PROPAGATION

Uncertainty in OSGAR is represented by adding a covariance matrix to the transformation nodes in the scene graph. Any transformation matrix is considered to be the mean of a probability distribution function (PDF) for that transformation, instead of just a single discrete transformation. If no uncertainty information is specified for a transformation, it is assumed to be exact. The mean of the PDF (i.e., the original transformation) is used for culling, rendering, and so on, as before.

No restrictions are placed on the form of the transformation matrices; the scene graph is assumed to be composed of arbitrary nodes with arbitrary affine transformations between them. Specifically, let $\mathbf{M}_i^j$ be the true relative transformation from node i to node j ,

$$\mathbf{M}_i^j = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}$$

Each element can take arbitrary values. It is not even possible to assume that $m_{44} = 1$. To parameterize such a general matrix, a number of authors have developed methods to decompose an arbitrary matrix into a set of primitive operations including rotation, translation and scale [78, 38]. However, these decompositions are constructed by applying potentially expensive nonlinear operations (such as single value decomposition). Because the graph can be extremely large, a significant number of these decomposition operations might be performed leading to significant computational costs. Therefore, to simplify the implementation, all errors are expressed directly in terms of the elements of the transformation matrix. In other words, the uncertainty is a 16-dimensional state which corresponds to each element in the transformation matrix. This is a straightforward generalization of the approach of Bar-Itzhack for direction cosine matrices [**?**].

$\mathbf{M}_r^n$ is the cumulative transformation matrix from the root node $\mathbf{R}$ to an arbitrary node $\mathbf{N}$. This transformation is given by:

$$\mathbf{M}_r^n = \prod_{\forall i \in P} \mathbf{M}_{i-1}^i, \tag{25}$$

where p is the path from the root node $\mathbf{R}$ to an arbitrary node $\mathbf{N}$ and i are nodes in this path.

However, the system does not have access to these true values. Rather, it only has access to the estimated relative transformation $\hat{\mathbf{M}}_i^j$. The difference between the two is due to the sources of uncertainty outlined in Section 3.7. As a result, the cumulative transformation calculated in the graph is:

$$\hat{\mathbf{M}}_r^n = \prod_{\forall i \in P} \hat{\mathbf{M}}_{i-1}^i \tag{26}$$

Therefore, the problem is to estimate the statistics of $\hat{\mathbf{M}}_r^n$ given that error can be introduced at any transformation in the tree.

The error introduced at a node is assumed to be an additive matrix,

$$\hat{\mathbf{M}}_i^j = \mathbf{M}_i^j + \delta \mathbf{M}_i^j. \tag{27}$$

Therefore, the error propagation equation is

$$\begin{aligned} \mathbf{M}_r^i + \delta\mathbf{M}_r^i &= \left(\mathbf{M}_{i-1}^i + \delta\mathbf{M}_{i-1}^i\right)\left(\mathbf{M}_r^{i-1} + \delta\mathbf{M}_r^{i-1}\right) \\ &= \mathbf{M}_{i-1}^i\mathbf{M}_r^{i-1} + \mathbf{M}_{i-1}^i\delta\mathbf{M}_r^{i-1} \\ &\quad + \delta\mathbf{M}_{i-1}^i\mathbf{M}_r^{i-1} + \delta\mathbf{M}_{i-1}^i\delta\mathbf{M}_r^{i-1} \end{aligned} \tag{28}$$

Assuming that the error introduced at a node is independent of the error introduced at preceding nodes, the expected value of the last term will always evaluate to $\mathbf{0}$ and thus can be neglected. Therefore, the equation which propagates the error down the scene graph is as follows:

$$\begin{aligned} \hat{\mathbf{M}}_r^i &= \hat{\mathbf{M}}_{i-1}^i\hat{\mathbf{M}}_r^{i-1} \\ \delta\mathbf{M}_r^i &= \mathbf{M}_{i-1}^i\delta\mathbf{M}_r^{i-1} + \delta\mathbf{M}_{i-1}^i\mathbf{M}_r^{i-1} \end{aligned} \tag{29}$$

REFERENCES

[1] Athans, M., Wishner, R. P., and Bertolini, A., "Suboptimal State Estimation For Continuous-Time Nonlinear Systems From Discrete Noisy Measurements," *IEEE Transactions on Automatic Control*, vol. TAC-13, pp. 504–518, October 1968.

[2] Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., and MacIntyre, B., "Recent advances in augmented reality," *IEEE Computer Graphics and Applications*, vol. 21, pp. 34–47, /2001.

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

[4] Bajura, M., Fuchs, M., and Ohbuchi, R., "Merging virtual objects with the real world: Seeing ultrasound imagery within the patient," in *Proceedings SIGGRAPH 1992*, pp. 203–210, July 26-31 1992.

[5] Bascle, B., Navab, N., Loser, M., Geiger, B., and Taylor, R., "Needle placement under x-ray fluoroscopy using perspective invariants,"

[6] Bauer, M., Bruegge, B., Klinker, G., MacWilliams, A., Reicher, T., Riss, S., Sandor, C., and Wagner, M., "Design of a component-based augmented reality framework," in *Proceedings of The Second IEEE and ACM International Symposium on Augmented Reality (ISAR 2001)*, 2001.

[7] Bell, B., Feiner, S., and Höllerer, T., "View management for virtual and augmented reality," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2001)*, November 11-14 2001.

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

[9] Calvin R. Maurer, J., Sauerd, F., Huc, B., Bascled, B., Geigerd, B., Wenzeld, F., Recchib, F., Rohlnga, T., Brownc, C. M., Bakosa, R. S., Maciunasa, R. J., and Bani-Hashemid, A., "Augmented reality visualization of brain structures with stereo and kinetic depth cues: System description and initial evaluation with head phantom," in *Medical Imaging 2001*, pp. 445–456, 2001.

[10] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R., "Reconstruction and representation of 3D objects with radial basis functions," in *SIGGRAPH 2001, Computer Graphics Proceedings* (Fiume, E., ed.), pp. 67–76, ACM Press / ACM SIGGRAPH, 2001.

[11] Cascia, M., Sclaroff, S., and Athitsos, V., "Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models," in *IEEE IEEE PAMI '00)*, vol. 21:6, June 1999.

[12] Coelho, E. M. and MacIntyre, B., "High-level tracker abstractions for augmented reality system design," in *The Int'l Workshop on Software Technology for AR Systems (STARS 2003)*, October 7-10 2003.

[13] Coelho, E. M. and MacIntyre, B., "Augmenting real environments," in *The Third Young Investigator's Forum in Virtual Reality (YVR 2005)*, February 24-25 2005.

[14] Coelho, E. M., MacIntyre, B., and Julier, S., "osgAR: A scenegraph with uncertain transformations," in *Procedings of the Third International Symposium on Mixed and Augmented Reality (ISMAR 2004)*, pp. 6–15, November 2-5 2004.

[15] Coelho, E. M., MacIntyre, B., and Julier, S., "Supporting interaction in augmented reality in the presence of uncertain spatial knowledge," in *the Eighteenth Annual ACM Symposium on User Interface Software and Technology (UIST 2005)*, October 23-26 2005.

[16] Cohen, P., Johnston, M., McGee, D., Oviatt, S., and others, "Quickset: multimodal interaction for distributed applications," in *Proc. of Intl. Multimedia Conference*, (New York, NY), pp. 31–40, 1997.

[17] Cortadellas, J., Bellaire, G., and Graschew, G., "New concepts for intraoperative navigation: Calibration of a 3-d laparoscope," in *Bildverarbeitung fur die Medizin*, pp. 158–162, 2000.

[18] Curtis, D., Mizell, D., Gruenbaum, P., and Janin, A., "Several devils in the details: Making an ar application work in the airplane factory," in *Proceedings of the First International Workshop on Augmented Reality (IWAR '98*, pp. 47–60, 1998.

[19] Davison, A., "Real-time simultaneous localisation and mapping with a single camera," in *Proc. International Conference on Computer Vision, Nice*, October 2003.

[20] Davison, A. and Kita, N., "3d simultaneous localisation and map-building using active vision for a robot moving on undulating terrain," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition, Kauai*, IEEE Computer Society Press, 2001.

[21] Davison, A. J. and Murray, D. W., "Simultaneous localisation and map-building using active vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, July 2002.

[22] Dellaert, F., Fox, D., Burgard, W., and Thrun, S., "Monte carlo localization for mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA99)*, May 1999.

[23] DeSouza, G. N. and Kak, A. C., "Vision for mobile robot navigation: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 237–267, 2002.

[24] Devernay, F., Mourgues, F., and ere, E., "Towards endoscopic augmented reality for robotically assisted minimally invasive cardiac surgery," in *Proceedings of Medical Imaging and Augmented Reality*, 2001.

[25] Digital Plus by Lenz *http://www.lenz.com.*

[26] Feiner, S., MacIntyre, B., Höllerer, T., and Webster, A., "A Touring Machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment," *Personal Technologies*, vol. 1, no. 4, pp. 208–217, 1997.

[27] Feiner, S., MacIntyre, B., and Seligmann, D., "Knowledge-based augmented reality," *Communications ACM*, vol. 36, pp. 52–63, July 1993.

[28] Fogarty, J. and Hudson, S. E., "Gadget: a toolkit for optimization-based approaches to interface and display generation," in *Proceedings of the 16th annual ACM symposium on User interface software and technology*, pp. 125–134, ACM Press, 2003.

[29] Foxlin, E., *Motion Tracking Requirements and Technologies*, ch. 7. Lawrence Erlbaum Associates, Inc., 2002.

[30] Foxlin, E., Harrington, M., and Pfeifer, G., "Constellation: A wide-range wireless motion-tracking system for augmented reality and virtual set application," in *Proceedings of SIGGRAPH 98* (Cohen, M., ed.), pp. 371–378, Addison Wesley, 1998.

[31] Furmanski, C., Azuma, R., and Daily, M., "Augmented-reality visualizations guided by cognition: Perceptual heuristics for combining visible and obscured information," in *Proceedings of International Symposium on Mixed and Augmented Reality 2002 (ISMAR '02)*, pp. 215–225, September 30 – October 1st 2002.

[32] Gajos, K. and Weld, D. S., "Supple: automatically generating user interfaces," in *Proceedings of the 9th international conference on Intelligent user interface*, pp. 93–100, ACM Press, 2004.

[33] Gleicher, M., "A graphics toolkit based on differential constraints," in *Proceedings of the 6th annual ACM symposium on User interface software and technology*, pp. 109–120, ACM Press, 1993.

[34] Gomes, J. and Velho, L., "Abstraction paradigms for computer graphics," pp. 227–239, August 1998.

[35] Gonzalez, R. and Woods, R., *Digital Image Processing.* Addison-Wesley Publishing Company, 1992.

[36] Grossman, T., Wigdor, D., and Balakrishnan, R., "Multi-finger gestural interaction with 3d volumetric displays," in *Proceedings of the Seventeenth ACM Symposium on User Interface Software and Technology (UIST 2004)*, pp. 61–70, October 24-27 2004.

[37] Haralick, R. M., "Propagating covariance in computer vision," in *Proceedings of the 12th IAPR International Conference on Computer Vision & Image Processing*, pp. 493 – 498, 1994.

[38] Hartley, R. and Zisserman, A., *Multiple View Geometry.* Cambridge University Press, 2 ed., 2003.

[39] Hoff, W., "Fusion of data from head-mounted and fixed sensors," in *Proceedings of the First International Workshop on Augmented Reality (IWAR '98*, pp. 167–182, 1998.

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

[41] Holloway, R. L., "Registration Error Analysis for Augmented Reality," *Presence: Teleoperators and Virtual Environments*, vol. 6, pp. 413–432, August 1997.

[42] Holloway, R., *Registration errors in augmented reality systems.* PhD thesis, University of North Carolina at Chapel Hill, 1995.

[43] Huertas, A., Kim, Z., and Nevatia, R., "Multisensor integration for building modeling," in *Int. Conf. On Computer Vision and Pattern Recognition (CVPR'2000)*, pp. 203–210, June 2000.

[44] Intersense *IS-600 User's Manual.*

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

[46] Jain, A., *Fundamentals of Digital Image Processing.* Prentice Hall, 1986.

[47] Julier, S. J., "A Skewed Approach to Filtering," in *The Proceedings of AeroSense: The 12th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Orlando, Florida*, vol. 3373, pp. 54–65, SPIE, 1998. Signal and Data Processing of Small Targets.

[48] Julier, S. J., Lanzagorta, M., Baillot, Y., and Brown, D., "Information filtering for mobile augmented reality," in *IEEE and ACM International Symposium on Augmented Reality 2000 (ISAR '00)*, pp. 3–11, October 5-6 2000.

[49] Julier, S. J. and Uhlmann, J. K., "A consistent, debiased method for converting between polar and cartesian coordinate systems," in *The Proceedings of AeroSense: Acquisition, Tracking and Pointing XI*, vol. 3086, pp. 110–121, SPIE, 1997.

[50] Julier, S. J., Uhlmann, J. K., and Durrant-Whyte, H. F., "A new approach for the nonlinear transformation of means and covariances in linear filters," *IEEE Transactions on Automatic Control*, vol. 5, pp. 477–482, March 2000.

[51] Julier, S. J. and Uhlmann, J. K., "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, pp. 401–422, March 2004.

[52] Kaiser, E., Olwal, A., McGee, D., Benko, H., Corradini, A., Li, X., Feiner, S., and Cohen, P., "Mutual dissambiguation of 3d multimodal interaction in augmented and virtual reality," in *Proceedings of The Fifth International Conference on Multimodal Interfaces (ICMI 2003)*, pp. 12–19, November 5-7 2003.

[53] KAM Industries *http://www.trainpriority.com.*

[54] Kanbara, M., Okuma, T., Takemura, H., and Yokoya, N., "A stereoscopic video see-through augmented reality system based on real-time vision-based registration," in *Proceedings of IEEE Virtual Reality 2000 (VR 2000*, pp. 858–868, March 18–22 2000.

[55] Kelly, A., “Precision dilution in mobile robot position estimation,” in *Intelligent Autonomous Systems*, 2003.

[56] Klinker, G., Stricker, D., and Reiners, D., *Augmented reality for exterior construction applications.* Lawrence Erlbaum Press, 2001.

[57] Klinker, G., Creighton, O., Dutoit, A. H., Kobylinski, R., Vilsmeier, C., and Brgge, B., “Augmented maintenance of powerplants: a prototyping case study of a mobile ar system,” in *Proceedings of the International Symposium on Augmented Reality 2001 (ISAR ’01)*, pp. 124–133, October 29–30 2001.

[58] Klinker, G., Reicher, T., and Brugge, B., “Distributed user tracking concepts for augmented reality applications,” in *Proceedings of the International Symposium on Augmented Reality (ISAR 2000)*, October 5-6 2000.

[59] Leonard, J. J. and Durrant-Whyte, H. F., “Mobile robot localization by tracking geometric beacons,” *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, 1991.

[60] Lerro and Y. K. Bar-Shalom, D., “Tracking with Debiased Consistent Converted Measurements vs. EKF,” *IEEE Transactions on Aerospace and Electronics Systems*, vol. AES-29, pp. 1015–1022, July 1993.

[61] MacIntyre, B. and Coelho, E. M., “Adapting to dynamic registration errors using level of error (loe) filtering,” in *Proceedings of the International Symposium on Augmented Reality (ISAR 2000)*, October 5-6 2000.

[62] MacIntyre, B., Coelho, E. M., and Julier, S., “Estimating and adapting to registration errors in augmented reality systems,” in *Proceedings of the IEEE Virtual Reality (VR 2002)*, March 2002.

[63] Mitschke, M., Bani-Hashemi, A., and Navab, N., “Interventions under video-augmented x-ray guidance: Application to needle placement,” in *Proceedings of the Third International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2000)* (Delp, S. L., DiGioia, A. M., and Jaramaz, B., eds.), vol. 1935 of *Lecture Notes in Computer Science*, Springer, 2000.

[64] Navab, N., Bascle, B., Loser, M., Geiger, B., and Taylor, R., “Visual servoing for automatic and uncalibrated needle placement for percutaneous procedures,” in *Int. Conf. On Computer Vision and Pattern Recognition (CVPR’2000)*, pp. 327–334, June 2000.

[65] Navab, N., Bani-Hashemi, A., and Mitschke, M., “Merging visible and invisible: two camera-augmented mobile c-arm (CAMC) applications,” in *Proc. 2nd International Workshop on Augmented Reality (IWAR’99)*, pp. 134–141, October 1999.

[66] Navab, N., Bascle, B., Appel, M., and Cubillo, E., “Scene augmentation via the fusion of industrial drawings and uncalibrated images with a view to marker-less calibration,” in *Proc. 2nd International Workshop on Augmented Reality (IWAR’99)*, October 1999.

[67] Nevatia, R. and Price, K., “Automatic and interactive modeling of buildings in urban environments from aerial images,” in *IEEE ICIP*.

[68] Newman, J., Ingram, D., and Hopper, A., "Augmented reality in a wide area sentient environment," in *Proceedings of the International Symposium on Augmented Reality (ISAR 2001)*, October 29–30 2001.

[69] Olwal, A., Benko, H., and Feiner, S., "Senseshapes: Using statistical geometry for object selection in a multimodal augmented reality system," in *Proceedings of The Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2003)*, pp. 300–301, October 7-10 2003.

[70] Open Scene Graph (OSG) *http://www.openscenegraph.org.*

[71] Piekarski, W. and Thomas, B., "Tinmith-evo5 - an architecture for supporting mobile augmented reality environments," in *Proceedings of the International Symposium on Augmented Reality (ISAR 2001)*, pp. 177–178, October 29–30 2001.

[72] Piekarski, W. and Thomas, B., "Augmented reality working planes: A foundation for action and construction at a distance," in *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR 2004)*, November 2004.

[73] Pito, R., *Automated surface acquisition using range cameras.* PhD thesis, University of Pennsylvania, 1996.

[74] Reed, M., *Solid Model Acquisition from Range Imagery.* PhD thesis, Columbia University, 1998.

[75] Reiners, D., Stricker, D., Klinker, G., and Muller, S., "Augmented reality for construction tasks: Doorlock assembly," in *Proceedings of the First International Workshop on Augmented Reality (IWAR '98*, 1998.

[76] Robertson, C. and MacIntyre, B., *Adapting to Registration Error in an Intent-Based Augmentation System.* Springer Verlag, 2003.

[77] Russakoff, D. B. and Herman, M., "Head tracking using stereo," *Machine Vision and Applications*, vol. 13, pp. 164–173, 2002.

[78] Shoemake, K. and Duff, T., "Matrix animation and polar decomposition," in *Proceedings of Graphics Interface '92* (MacKenzie and James Stewart Davis, S., ed.), pp. 258–264, May 11–15 1992.

[79] Skubic, M., Bailey, C., and Chronis, G., "A sketch interface for mobile robots," in *Procedings of the 2003 IEEE International Conference on Systems, Man & Cybernetics (SMC 2003)*, pp. 918–924, October 2003.

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

[81] Starner, T., Schiele, B., Rhodes, B. J., Jebara, T., Oliver, N., Weaver, J., and Pentland, A., "Augmented realities integrating user and physical models," in *Proceedings of the First International Workshop on Augmented Reality (IWAR '98*, 1998.

[82] State, A., Hirota, G., Chen, D. T., Garrett, W. F., and Livingston, M. A., "Superior augmented reality registration by integrating landmark tracking and magnetic tracking," *Computer Graphics*, vol. 30, no. Annual Conference Series, pp. 429–438, 1996.

[83] State, A., Livingston, M. A., Hirota, G., Garrett, W. F., Whitton, M. C., Fuchs, H., and Pisano, E. D., "Technologies for augmented-reality systems: realizing ultrasound-guided needle biopsies," in *Proceedings of SIGGRAPH 96*, (New Orleans, LA), pp. 439–446, August 4–9 1996.

[84] Stevens, M. P., Zeleznik, R. C., and Hughes, J. F., "An architecture for an extensible 3d interface toolkit," in *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pp. 59–67, ACM Press, 1994.

[85] Toussaint, G. T., "A simple linear algorithm for intersecting convex polygons," *The Visual Computer*, vol. 1, pp. 118–123, 1985.

[86] Traub, J., Feuerstein, M., Bauer, M., Schirmbeck, E. U., Najafi, H., Bauernschmitt, R., and Klinker, G., "Augmented reality for port placement and navigation in robotically assisted minimally invasive cardiovascular surgery," in *Proceedings of Computer Assisted Radiology and Surgery (CARS 2004)*, (Chicago, USA), pp. 735–740, June 2004.

[87] Tuceryan, M. and Navab, N., "Single point active alignment method (spaam) for optical see-through hmd calibration for ar," in *Proceedings of the International Symposium on Augmented Reality (ISAR 2000)*, October 5-6 2000.

[88] Uhlmann, J. K., "Simultaneous map building and localization for real time applications," tech. rep., University of Oxford, 1994. Transfer thesis.

[89] Virtual Reality Peripheral Network *http://www.cs.unc.edu/Research/vrpn/*.

[90] Vogt, S., Khamene, A., Sauer, F., and Niemann, H., "Single camera tracking of marker clusters: multiparameter cluster optimization and experimental verification," in *Proceedings of International Symposium on Mixed and Augmented Reality 2002 (ISMAR '02)*, pp. 127–136, September 30 – October 1st 2002.

[91] Wagner, M., MacWilliams, A., Bauer, M., Klinker, G., Newman, J., Pintaric, T., and Schmalstieg, D., "Fundamentals of ubiquitous tracking," in *Second International Conference on Pervasive Computing, Hot Spots section*, (Vienna, Austria), April 2004.

[92] Wang, J., Azuma, R., Bishop, G., Chi, V., Eyles, J., and Fuchs, H., "Tracking a head-mounted display in a room-sized environment with head-mounted cameras," in *Proc. of Helmet-Mounted Displays II*, vol. 1290, pp. 47–57, April 19–20 1990.

[93] Ward, M., Azuma, R., Bennett, R., Gottschalk, S., and Fuchs, H., "A demonstrated optical tracker with scalable work area for head-hounted display systems," in *Symposium on Interactive 3D Graphics*, pp. 43–52, 1992.

[94] Welch, G. and Foxlin, E., "Motion tracking: No silver bullet, but a respectable arsenal," *IEEE Computer Graphics and Applications*, vol. 22, no. 6, pp. 24–38, 2002.

[95] You, S. and Neumann, U., "Fusion of vision and gyro tracking for robust augmented reality registration," in *IEEE Virtual Reality*, pp. 71–78, March 2001.

[96] You, S., Neumann, U., and Azuma, R., "Orientation tracking for outdoor augmented reality registration," *IEEE Computer Graphics and Applications*, vol. 19, pp. 36–42, 1999.

INDEX

VITA

Enylton Machado Coelho was born on the 10th of October, 1970 in the city of Rio de Janeiro, Brazil. Enylton received a Bachelor of Science in Computer Science from the Federal University of Rio de Janeiro (UFRJ) and a Master of Science from the Pontifical Catholic University in Rio de Janeiro (PUC-Rio). Enylton has been involved with computer graphics since 1994 when he joined the computer graphics group (Visgraf) at the National Institute of Pure and Applied Mathematics (IMPA). For about five years, before moving to Atlanta, Enylton dedicated his weekends to windsurfing and became reasonably proficient at it. Since joining the Georgia Institute of Technology, Enylton has been working with Dr. Blair MacIntyre at the Augmented Environments Lab. He worked as a intern at the Naval Research Lab. in Washington, DC and at the Siemens Corporate Research, in Princeton, NJ. He was also a visiting scholar at the Technical University Graz, in Graz, Austria.