ARVISCOPE

Georeferenced Visualization of Dynamic Construction Processes in Three-Dimensional Outdoor Augmented Reality

by

Amir H. Behzadan

A Dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Civil Engineering)
in The University of Michigan
2008

Doctoral Committee:

Assistant Professor Vineet R. Kamat, Chair
Professor Photios G. Ioannou
Lecturer John G. Everett
Research Scientist Klaus-Peter Beier

To My Lovely Family,
That if it were not for their help and support,
I could not make my way towards my every success.

To Sara,
Who held my hands all along the way,
Who never left me alone.

Acknowledgments

The present document is my final Ph.D. dissertation. This dissertation describes my graduate research which began in September 2004 and concluded in April 2008.

During this period, I had the privilege to work with Professor Vineet R. Kamat who served as my research and graduate advisor in the Department of Civil and Environmental Engineering at the University of Michigan. During this period, he has been extremely diligent and willingly helpful. I would like to take this opportunity to specially thank him for his sincere and resolute support. He has always been an excellent teacher and an incredible advisor and I really appreciate his every effort and help to achieve the objectives of the presented work. I would also like to appreciate the support of the other members of my Ph.D. committee, Professors Photios Ioannou, Klaus-Peter Beier, and John Everett, for their contributions to the presented work, and for providing me with invaluable advice and deep insight and suggestions to improve the quality of my work. I am also grateful to Professors Sherif El-Tawil and Jerome Lynch at the University of Michigan, Professor Kincho Law at Stanford University (Stanford, CA), and Mr. Parham Khoshkbari at Kleinfelder Construction Company (San Jose, CA) for their confidence in and support of my research. Their deep insight and advice improved the quality of this research and provided me with the necessary boost that helped accomplish the work in a timely manner.

I would also like to thank my parents, Mohammad Ali Behzadan and Mehri Daliri Farahani, my brother Afshin, and my sister Nazanin, who have always been a major source of motivation and support for me in every minute of my life. My special thanks go to Sara Jabbarizadeh, my wonderful friend, who was a constant inspiration and emotional support for me during my Ph.D. years and encouraged me a lot to accomplish my goals. I

strongly believe if it were not for their encouragements and persevering endeavor, I could not achieve any progress in my life, education, and career.

I am very thankful to all my friends, especially Hiam Khoury, Mustafa Saadi, Rita Awwad, and Chachrist Srisuwanrat for their sincere friendship and help during the past years.

Amir H. Behzadan

Table of Contents

List of Figures

List of Tables

List of Appendices

List of Abbreviations

2D: Two Dimensional

3D: Three Dimensional

3DOF: Three Degree of Freedom

6DOF: Six Degree of Freedom

4D: Four Dimensional

ACA: American Chiropractic Association

ACD: Activity Cycle Diagram

API: Application Programming Interface

AR: Augmented Reality

ARCHEOGUIDE: Augmented Reality Based Cultural Heritage On-Site Guide

ARION: Augmented Reality for Intra-Operative Navigation

ARVISCOPE: Augmented Reality Visualization of Simulated Construction Operations

ASCII: American Standard Code for Information Interchange

BARS: Battlefield Augmented Reality System

CAD: Computer-Aided Design

CAVE: Computer-Aided Virtual Environment

COTS: Commercial-off-the-Shelf

CPU: Central Processing Unit

CRC: Cyclic Redundancy Check

CT: Computed Tomography

DES: Discrete Event Simulation

DGPS: Differential Global Positioning System

DLL: Dynamic Link Library

EOL: End of Line

GPS: Global Positioning System

HFOV: Horizontal Field of View

HMD: Head Mounted Display

HTML: Hyper Text Markup Language

IT: Information Technology

LADAR: Laser Detection and Ranging

LO: Local Origin

MRI: Magnetic Resonance Imaging

MSL: Mean Sea Level

MTM: Modified Transformation Method

NMEA: National Marine Electronics Association

OOD: Object Oriented Design

P&P: Plug and Play

RGB: Red-Green-Blue

RGP: Rear Guide Point

RTK: Real Time Kinematics

SVGA: Super Video Graphics Array

STROBOSCOPE: State and Resource Based Simulation of Construction Processes

URL: Uniform (Universal) Resource Locator

USB: Universal Serial Bus

VFOV: Vertical Field of View

VGA: Video Graphics Array

VITASCOPE: Extensible and Scalable 3D Visualization of Simulated Construction Operations

VR: Virtual Reality

VRML: Virtual Reality Modeling Language

WAAS: Wide Area Augmentation System

X3D: Extensible 3D

XML: Extensible Markup Language

Abstract

Construction processes can be conceived as systems of discrete, interdependent activities. Discrete Event Simulation (DES) has thus evolved as an effective tool to model operations that compete over available resources (personnel, material, and equipment). A DES model has to be verified and validated to ensure that it reflects a modeler's intentions, and faithfully represents a real operation. 3D visualization is an effective means of achieving this, and facilitating the process of communicating and accrediting simulation results.

Visualization of simulated operations has traditionally been achieved in Virtual Reality (VR). In order to create convincing VR animations, detailed information about an operation and the environment has to be obtained. The data must describe the simulated processes, and provide 3D CAD models of project resources, the facility under construction, and the surrounding terrain (Model Engineering). As the size and complexity of an operation increase, such data collection becomes an arduous, impractical, and often impossible task. This directly translates into loss of financial and human resources that could otherwise be productively used. In an effort to remedy this situation, this dissertation proposes an alternate approach of visualizing simulated operations using Augmented Reality (AR) to create mixed views of real existing jobsite facilities and virtual CAD models of construction resources. The application of AR in animating simulated operations has significant potential in reducing the aforementioned Model Engineering and data collection tasks, and at the same time can help in creating visually convincing output that can be effectively communicated.

This dissertation presents the design, methodology, and development of ARVISCOPE, a general purpose AR animation authoring language, and ROVER, a mobile computing

hardware framework. When used together, ARVISCOPE and ROVER can create three-dimensional AR animations of any length and complexity from the results of running DES models of engineering operations. ARVISCOPE takes advantage of advanced Global Positioning System (GPS) and orientation tracking technologies to accurately track a user's spatial context, and georeferences superimposed 3D graphics in an augmented environment. In achieving the research objectives, major technical challenges such as accurate registration, automated occlusion handling, and dynamic scene construction and manipulation have been successfully identified and addressed.

Chapter 1

Introduction

The objective of this research was to find methods of accurately creating dynamic animations of simulated construction operations in outdoor Augmented Reality (AR). The dynamics of construction projects and the uncertainty involved in the corresponding activities of a typical construction operation are major incentives for decision makers to introduce and adapt Information Technology (IT) based solutions that can conveniently supplement traditional methods of operations planning, site layout planning, resource management, crew selection, and work progress monitoring [1,2,3]. Simulation has gained a lot of credibility over the past few years as a powerful tool to study the complexities of an operation and perform sensitivity analyses that highlight the interdependencies between the involved entities and resources [4,5,6,7]. Depending on the complexity of a simulated operation, the results of the simulation model have to be carefully reviewed, interpreted, and communicated. In order for a simulation model to be practically used in real life decision making, it is extremely important that the results of the model be understood and accepted in order to satisfy two main criteria: 1) the simulation model has been built right, and 2) the right simulation model has been built. The first condition is addressed by verification in which the computer representation is compared to the conceptual model to make sure that the parameters and logical structure used are correctly representing a modeler's understanding of the operation. The second condition is fulfilled by validation of the simulation model, which determines if the computer model is an accurate representation of the real world operation [8].

There are several approaches to verification and validation of a simulation model [9]. Among them, 3D visualization is an effective approach as it facilitates the process of accrediting and communicating the simulation results by providing an easily interpretable and understandable visual interface [8,10,11,12,13,14]. 3D visualization of simulated operations has traditionally been performed in Virtual Reality (VR). In order to create realistic VR displays of a simulated process, detailed data about the process as well as the environment in which it takes place has to be obtained. Such data must be able to describe the simulation, and 3D CAD models of the resources, the facility under construction, and terrain topography (Model Engineering) [15]. As the size and complexity of an operation increases, data collection also becomes time and resource consuming. This directly translates into loss of project financial and human resources that could otherwise be saved and used more productively. In an effort to remedy this situation, an AR-based approach to create dynamic animations of simulated operations has been developed in this research and is described in this dissertation. Following this approach, AR is used to create mixed views of real existing facilities on the jobsite and virtual CAD objects of resources involved in the construction. The application of AR in animating simulated construction operations has great potential in reducing the Model Engineering and data collection tasks, and at the same time leads to visually convincing animation results [16].

The primary research question that this dissertation addressed was how to create smooth, dynamic, and accurate animations of simulated construction operations in an AR environment based only on communicated discrete events occurring within Discrete Event Simulation (DES) models, while at the same time providing mobile users with the ability to study the animated operations in real time inside a mixed environment of real and virtual objects. This can significantly facilitate the process of user-oriented verification and validation of simulated operations due to the interaction between a user and the scene where the user can study the model from different perspectives and under different simulated scenarios.

The end result of this effort is ARVISCOPE, an acronym for Augmented Reality Visualization of Simulated Construction Operations. ARVISCOPE is an animation authoring language capable of describing a dynamic simulated construction operation in AR using simple yet expressive statements that are meant to be written either manually or by end-user programmable software such as DES tools. Instructions written in this language are ordered chronologically to describe the temporal and spatial characteristics of each activity within a simulated operation. The language and its implementation are also capable of animating the interactions between the user and the mixed environment as well as virtual-real object interactions.

1.1. Importance of the Research Activity

Operations planning is a critical component of managing and controlling the different aspects of an ongoing construction project. A comprehensive operational level work plan and corresponding site layout that can provide easy accessibility to different locations of the jobsite is a key to significant savings in project time and costs, and at the same time results in reduction of several unwanted resource and physical space conflicts. Working in a well organized operational environment with minimum amount of time spent on resolving conflicts and tasks not directly related to the scope of the project is an important factor in meeting the project schedule and budget.

This is a major incentive for using computer applications to model, simulate, and visualize operations beforehand in order to identify any potential project related events that may cause unexpected delays or conflicts during the course of the real operations, and plan ahead of time to avoid such situations during actual construction. Amongst several approaches of modeling construction operations, DES has gained significant credibility since almost every construction operation can be effectively broken down and modeled as a system of discrete activities that each consumes resources (personnel, material, and equipment) in order to be completed [6,17]. DES models provide an effective means to establish logical relationships between activities within a project which compete over and make use of available and often scarce resources.

For a relatively small operation, reviewing, interpreting, verifying, and validating the results of a simulation model can be done manually using statistical data, flowcharts, flow diagrams, and other numerical tools [9]. However, as the size of the operation increases and with the introduction of more resources and activities within the operation, automating the process of communicating the results of a simulation model for verification and validation purposes turns into a crucial need that has to be addressed in a timely and effective manner. One of the effective methods of verifying and validating the results of a simulation model is to communicate the flow of activities in a chronologically and spatially accurate manner. 3D visualization is an effective means to achieve this objective and facilitate the process of understanding and accrediting the simulation results [7,8]. The application of VR in visualizing the results of simulated processes has been investigated by several researchers. In contrast, there have been very few studies conducted in AR primarily due to the fact that the successful implementation of an AR environment requires addressing several critical challenges and problems in order to produce an acceptable degree of reliability and credibility in the visual results.

AR is a rapidly advancing technology that offers high potential for significant improvement in many scientific and engineering domains such as construction, industrial and mechanical engineering, medicine, aviation, and manufacturing. For example, using AR in construction can allow walking through an actual site and experiencing a virtual facility as it may be built in the future, or looking into the ground and “seeing” utility lines as they exist based on as-built CAD data. AR can also be applied to assembly lines, with the option of presenting the individual work steps to the assembler in an augmented visualization environment during training or actual assembly phases. AR visualizations can also help product manufacturers in providing required training to production line staff about assembling different components to enhance productivity. In aviation, aircraft companies such as Boeing have recently carried out trials using AR technology in the assembly field [18]. Flow and inventory control of raw material and final products in a warehouse can also be conveniently animated using AR to minimize order lead times, product delivery times, and maintenance request waiting times. The applications and potential of AR are thus vast and applicable to several engineering domains.

1.2. Prior State of Knowledge

AR related research has been conducted in a growing number of scientific and engineering disciplines. Integration of AR in CAD/CAM systems helps manufacturing companies (e.g. automotive, airlines, etc.) to model mechanical designs, visualize stresses or flows calculated from previous simulations, test for interferences through digital preassembly, and study the manufacturability and maintainability of subsystems [19].

Visualization of medical information projected onto a patient's body is also an established application of AR technology. Traditional Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) images provide physicians with information on a totally detached display from a patient. Using AR displays allow MRI and CT images to be superimposed over the patient's anatomy which can assist in tasks such as planning of surgical procedures [19]. In the Division of Medical and Biological Informatics at ZGDV in Germany, work has been done on a project known as ARION (Augmented Reality for Intra-Operative Navigation), using AR for image-guided surgery [20].

The Computer Graphics Center, also at ZGDV in Germany, in partnership with several European businesses, has been involved in the ARCHEOGUIDE Project (Augmented Reality Based Cultural Heritage On-site Guide). This project seeks to create a virtual tour guide for individuals visiting great historical and cultural sites. ARCHEOGUIDE has been tested near the remaining foundation of Hera's temple in Olympia, Greece, where it created an augmented view of the temple as it would have appeared in ancient Greece [21]. A similar real time touring application has been developed by a team at Columbia University. By providing wireless internet access to the mobile AR backpack, along with a link to the University's campus information server, the user has their display augmented with information about buildings currently in their view [22].

AR has also been studied for implementation in the military sector. The Battlefield Augmented Reality System (BARS) was developed and tested at the Naval Research Laboratory as a mobile AR system to improve situational awareness between remote

users in the field. The purpose of this project was to analyze interaction and data distribution among networked users as they inspect an area [23,24]. AR is also being investigated in realms beyond science and engineering including the creation of first-person, real time video game simulations. The ARQuake system, developed by researchers at the University of South Australia, applies AR technology to the first-person shooter genre, allowing the user to physically roam the terrain, with the video game graphics and opponents projected as virtual images onto the background [25,26,27].

The application of visualization techniques for planning, analysis, and design in construction and civil engineering is relatively new compared to the sizeable amount of AR related research conducted for diverse applications in fields such as manufacturing, medical operations, military, and gaming. During recent years, visualization has gained an increasing credibility among researchers in construction and has been noted as one of the four main IT domains in construction [28]. Previous studies have explored the application of AR as a state-of-the-art visualization technique for a limited number of architecture and construction applications.

For example, Webster et al. [29] presented a system that shows locations of columns behind finished walls, and rebar inside columns. They also presented an AR system to guide workers through the assembly of a space frame. Roberts et al. [30] used AR to overlay locations of subsurface utility lines onto real world views. These applications have been designed to demonstrate the potential of AR in helping maintenance workers avoid buried infrastructure and structural elements. Hammad et al. [31] augmented contextual information on real views of bridges to help inspectors conduct inspections more effectively. Kamat and El-Tawil [32] used AR to study the extent of horizontal displacements sustained by structural elements due to extreme loading conditions. Wang and Dunston [33] explored the potential of AR as an assistant viewer for computer-aided drawing. Behzadan and Kamat [16] studied the applicability of 4D CAD visualization in AR for construction [34] by performing several experiments using time tagged CAD models of building structures.

Each of the abovementioned research projects, however, was focused on developing a domain specific AR platform that was intended for a very specific purpose, thereby limiting its widespread applicability. For example, the methods developed for and incorporated in such platforms are not available in a form that allows them to be readily extended or reused for other AR applications. In addition, prior research in AR:

- Does not incorporate arbitrary operations level complexity into AR visualizations.
- Does not consider dynamic objects evolving in an AR scene based on the results of discrete-event simulation.
- Does not afford unrestricted mobility to the observer of a dynamic AR scene.
- Does not consider the interaction between real and virtual objects in an AR scene.
- Does not consider problems associated with enabling AR visualization in outdoor unprepared environments.

1.3. Research Objectives

In order to address the limitations in the existing state of knowledge, the overall objective of this research was to investigate the requirements and develop the ability to use AR-based visualization for animating dynamic operations modeled and simulated using DES. The specific objectives of this research were identified as follows:

- Study the required parameters that can identify an observer's global spatial context, and design methods to obtain and retrieve this information in real time in order to update the graphical contents of a running AR animation.
- Investigate feasible methods to communicate with a running DES model in order to extract all necessary information that can describe the environment and the underlying logic of a dynamic operation. Data about the simulation resources (i.e. equipment, machinery, personnel, material) such as their quantity, configuration, interaction, and dynamic motion are examples of such required information.
- Develop an expressive yet simple animation authoring language capable of describing discrete activities of a DES model in a smooth, continuous manner within a georeferenced, outdoor, 3D AR environment.

- Identify and effectively address issues such as incorrect visual occlusion that significantly affect the visual impact and credibility of an AR animation. Such issues arise from the fact that in an AR animation, two groups of objects (i.e. virtual and real) have to appear to seamlessly coexist in a single scene

This dissertation presents the developed methodology and the design of AR-based visualization techniques, position and orientation tracking algorithms, and a powerful animation authoring language together with their practical implementation inside a mobile AR visualization tool. The designed animation language provides a convenient method to automatically author animations of any length and complexity at the operations level of detail using an external software process such as a running DES model. The results of the research have been validated by animating several simulated construction operations in outdoor AR.

1.4. Research Methodology

In order to effectively achieve the objectives of this research, the identified problems and challenges were clustered into several groups for investigation. The following is an abridged list of the major steps followed in this research to fulfill the requirements of a functional, reliable, and practical AR visualization methodology and tool that can be conveniently used in construction or any other engineering environment:

- Studied the details of involved parameters in tracking an observer's 3D position and head orientation in a global space without dependence on any type of preinstalled tracking infrastructure that might possibly limit the observer's freedom of movement.
- Designed and successfully implemented a stand-alone tracking method capable of communicating with positional and head tracking devices in real time and sending appropriate data to the AR application at each animation frame.
- Created a basic AR visualization application to evaluate the reliability of the tracking data in updating the graphical context of the observer's view based on

the animation scenario and the observer's latest position and head orientation inside the augmented environment.

- Studied how the existing visualization application and DES tools work to understand the underlying simulation data transferring algorithms that can enable a visualization application to create a dynamic animated scene of the corresponding simulated operations.
- Designed and implemented the core ARVISCOPE language in a mobile, AR environment that receives, interprets, and creates visual representations of the communicated simulation data from a DES model using 3D CAD models of the entities involved in the operation.
- Successfully tested the developed AR visualization application in several outdoor environments using various construction project scenarios and CAD models while providing full maneuvering ability to the scene observer allowing inspection of the animation from different perspectives.
- Studied the requirements of and designed dynamic scene graph based methods that support geometric instancing to effectively construct, manage, and robustly manipulate large dynamic graphical databases of CAD objects inside an augmented environment.
- Conducted an extensive study on the advantages and limitations of available methods to correctly handle visual occlusion effects in AR in order to explore techniques to design a self-contained automated method to correctly resolve occlusion cases in real time.
- Developed and implemented a state-of-the-art method to effectively resolve visual occlusion cases in AR and successfully validated the designed algorithms in miniature indoor construction environments.
- Successfully validated the designed algorithms, methods, and components of the developed AR platform in several experiments including both outdoor and indoor tests of simulated construction operations in augmented environments.

1.5. Dissertation Outline

During the course of this research, the author produced and published several scientific manuscripts in the form of journal papers and conference articles as each identified challenge or research problem was successfully addressed. Each Chapter in this dissertation is primarily composed of material from its corresponding publications. As a result, each chapter serves as a stand-alone document that describes the details of individual scientific questions successfully addressed, major challenges involved, and algorithms and methods studied, adopted, developed, and implemented in achieving a particular research objective.

Each Chapter of this dissertation has been compiled and written so that it is easily understandable and interpretable to a wide technically literate audience ranging from individuals with prior experience in computer simulation, modeling, and graphics, to those with only basic knowledge about construction operations and projects. Each Chapter concludes with an extensive list of references (books, papers, and internet URLs) that direct the reader to further resources related to the main topic of the chapter.
In Chapter 2, the final product of this research, an AR animation authoring language called ARVISCOPE, is introduced and described. The details of the language and the mechanisms that enable dynamic AR animations of DES models are presented. Chapter 3 describes the spatial tracking mechanism designed in this research and its integration into an existing AR application (also developed by the author) to provide real time access to an observer's positional and head orientation data. In Chapter 4, an animated operation is studied from the perspective of its individual entities (i.e. CAD models) focusing on their assembly into a single scene using a hierarchical approach to create a dynamic scene graph of animated objects. Chapter 5 discusses the details of the occlusion handling mechanism developed and implemented in this research. Finally, Chapter 6 documents results from several validation experiments that have been conducted to evaluate the functionality and reliability of the individual components of the developed AR methodology, each of which have been individually described in Chapters 2 through 5.

The information presented in the Chapters of this dissertation is supplemented by additional implementation details provided in several Appendices. In Appendix A, the details of the ARVISCOPE animation authoring language statements and their syntax are documented. Each individual statement is discussed with the help of an example describing the different arguments and parameters that statement requires to perform its designed functionality. In Appendix B, the steps for creating a global reference point (introduced in Chapter 2) in order to facilitate the construction of different elements in an AR scene (e.g. points, routes) are discussed with the help of an example. Appendix C describes the process of authoring an ARVISCOPE animation trace file by instrumenting an external software process such as a DES model. The example presented in this Appendix also serves as a guide on how to automatically create an accurate ARVISCOPE animation trace file with a high level of detail and with minimal effort. In Appendix D, the implementation details of ARVISCOPE are presented in the form of a flowchart and several pieces of pseudo code. While the contents of this Appendix are not intended to be software development aids, they can be used by interested readers to gain a better understanding of the internal processes and data flows within the developed AR platform, thereby facilitating its possible reuse and/or modification for use in other application domains.

1.6. References

[1] Osman, H. M., Georgy, M. E., and Ibrahim, M. E. (2003), “A Hybrid CAD-Based Construction Site Layout Planning System Using Genetic Algorithms”, Journal of Automation in Construction, 12(6), Elsevier Science, New York, NY, 749-764.

[2] Tam, C. M., Tong, T. K. L, and Chan, W. K. W. (2001), "Genetic Algorithms for Optimizing Supply Locations Around Tower Crane”, Journal of Construction Engineering and Management, 127(4), American Society of Civil Engineers (ASCE), Reston, VA, 315-321.

[3] Cheng, M. Y., O’Connor, J. T. (1996), "ArcSite: Enhanced GIS for Construction Site Layout”, Journal of Construction Engineering and Management, 122(4), American Society of Civil Engineers (ASCE), Reston, VA, 329-336.

[4] Martinez, J. C. (2001), “EZStrobe – General-Purpose Simulation System Based on Activity Cycle Diagrams”, In Proceedings of the Winter Simulation Conference (WSC), IEEE, Arlington, VA, 1556-1564.

[5] Shi, J. J., and Zhang, H. (1999), “Iconic Animation of Construction Simulation”, In Proceedings of the Winter Simulation Conference (WSC), IEEE, Phoenix, AZ, 992-997.

[6] Martinez, J. C., and Ioannou, P. G. (1999), “General-Purpose Systems for Effective Construction Simulation”, Journal of Construction engineering and Management, 125(4), American Society of Civil Engineers (ASCE), Reston, VA, 265-276.

[7] Kamat, V. R., and Martinez, J. C. (2001), “Visualizing Simulated Construction Operations in 3D”, Journal of Computing in Civil Engineering, 15(4), American Society of Civil Engineers (ASCE), Reston, VA, 329-337.

[8] Kamat, V. R. (2003), “VITASCOPE: Extensible and Scalable 3D Visualization of Simulated Construction Operations”, PhD Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[9] Sargent, R. G. (2003), “Verification and Validation of Simulation Models”, In Proceedings of the Winter Simulation Conference (WSC), New Orleans, LA, 37-48.

[10] Op den Bosch, A. (1994), “Design/Construction Process Simulation in Real-Time Object-Oriented Environments”, PhD Dissertation, Georgia Institute of Technology, Atlanta, GA.

[11] Barnes, M. R. (1997), “An Introduction to QUEST”, In Proceedings of Winter Simulation Conference (WSC), IEEE, Atlanta, GA, 619-623.

[12] Bishop, J. L., and Balci, O. (1990), “General Purpose Visual Simulation System: A Functional Description”, In Proceedings of the Winter Simulation Conference (WSC), IEEE, New Orleans, LA, 504-512.

[13] Rohrer, M. W. (2000), “Seeing Is Believing: The Importance of Visualization in Manufacturing Simulation”, In Proceedings of the Winter Simulation Conference (WSC), IEEE, Orlando, FL, 1211-1216.

[14] Rohrer, M. W., and McGregor, I. W. (2002), “Simulating Reality Using AUTOMOD”, In Proceedings of the Winter Simulation Conference (WSC), IEEE, San Diego, CA, 173-181.

[15] Brooks, Jr., F. P. (1999), “What’s Real About Virtual Reality?”, Journal of Computer Graphics and Applications, 16(6), IEEE, Piscataway, NJ, 16-27.

[16] Behzadan, A. H., and Kamat, V. R. (2005), “Visualization of Construction Graphics in Outdoor Augmented Reality”, In Proceedings of the Winter Simulation Conference (WSC), IEEE, Orlando, FL, 1914-1920.

[17] Martinez, J. C. (1996), “STROBOSCOPE: State and Resource Based Simulation of Construction Operations”, PhD Dissertation, University of Michigan, Ann Arbor, MI.

[18] Azuma, R. (1997), “A Survey of Augmented Reality.” Teleoperators and Virtual Environments, 6(4), MIT Press, Cambridge, MA, 355–385.

[19] Barfield, W., and Caudell, T. (2001), Fundamentals of Wearable Computers and Augmented Reality, Lawrence Erlbaum Associates, Philadelphia, PA.

[20] Suthau, T., Vetter, M., Hassenpflug, P., Meinzer, H., and Hellwich, O. (2002), A Concept Work for Augmented Reality Visualization Based on a Medical Application in Liver Surgery, Technical University Berlin, Commission V, WG V/3.

[21] Gleue, T., and Dähne, P. (2001), “Design and Implementation of a Mobile Device for Outdoor Augmented Reality in the Archeoguide Project”, In Proceedings of the 2001 Conference on Virtual Reality, Archeology, and Cultural Heritage, ACM, Glyfada, Greece, 161-168.

[22] Feiner, S., MacIntyre, B., Hollerer, T., Webster, A. (1997), ”A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment”, In Proceedings of the International Symposium on Wearable Computing (ISWC ’97), Cambridge, MA, 74-81.

[23] Brown, D., Julier, S., Baillot, Y., and Livingston, M. (2003), “An Event-Based Data Distribution Mechanism for Collaborative Mobile Augmented Reality and Virtual Environments”, In Proceedings of the Virtual Reality Conference, IEEE, Los Angeles, CA, 23-29.

[24] Livingston, M., Rosenblum, L., Julier, S., Brown, D., Baillot, Y. (2002), “An Augmented Reality System for Military Operations in Urban Terrain”, In Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC ‘02), Orlando, Florida, 1-8.

[25] Piekarski, W., Smith, R., and Thomas, B. (2004), “Designing Backpacks for High Fidelity Mobile Outdoor Augmented Reality”, In Proceedings of the 3rd International Symposium on Mixed and Augmented Reality (ISMAR04), IEEE & ACM, Arlington, Virginia.

[26] Piekarski, W., and Thomas, B. H. (2003), “ARQuake - Modifications and Hardware for Outdoor Augmented Reality Gaming”, In Proceedings of the 4th Australian Linux Conference, Perth, Australia.

[27] Thomas, B., Close, B., Donoghue, J., Squires, J., Bondi, P., Morris, M., and Piekarski, W. (2000), “ARQuake: An Outdoor/Indoor First Person Augmented Reality Application”, In Proceedings of the 4th International Symposium on Wearable Computers (ISWC), IEEE, Atlanta, GA, 149-146.

[28] Turk, Z. (2006), “Construction Informatics: Definition and Ontology”, Journal of Advanced Engineering Informatics, 20(2), Elsevier Science, New York, NY, 187-199.

[29] Webster, A., Feiner, S., MacIntyre, B., Massie, W., and Krueger, T. (1996), ”Augmented Reality in Architectural Construction, Inspection and Renovation”, In Proceedings of the 3rd Congress on Computing in Civil Engineering, American Society of Civil Engineers (ASCE), Anaheim, CA, 913-919.

[30] Roberts, G. W., Evans, A., Dodson, A., Denby, B., Cooper, S., and Hollands, R. (2002), “The Use of Augmented Reality, GPS, and INS for Subsurface Data Visualization”, In Proceedings of the FIG XXII International Congress, Washington, D.C.

[31] Hammad, A., Garrett, J. H., and Karimi, H. (2004). “Location-based computing for infrastructure field tasks.” Telegeoinformatics: Location-based computing and services, CRC Press, 287-314.

[32] Kamat, V. R., and El-Tawil, S. (2007), “Evaluation of Augmented Reality for Rapid Assessment of Earthquake-Induced Building Damage”, Journal of Computing in Civil Engineering, 21(5), American Society of Civil Engineers (ASCE), Reston, VA, 303-310.

[33] Wang, X., and Dunston, P. S. (2006), “Potential of Augmented Reality as an Assistant Viewer for Computer-Aided Drawing”, Journal of Computing in Civil Engineering, 20(4), American Society of Civil Engineers (ASCE), Reston, VA, 437-441.

[34] McKinney, K., Kim, J., Fischer, M., and Howard, C. (1996), “Interactive 4D-CAD”, In Proceedings of the 3rd Congress on Computing in Civil Engineering, American Society of Civil Engineers (ASCE), New York, 383–389.

Chapter 2

Automated Generation of Simulation-Based Augmented Reality Animations

2.1. Introduction

Operations planning is a critical component of managing and controlling the different aspects of an ongoing construction project. A comprehensive operational level work plan and corresponding site layout that provides easy accessibility to different locations of interest is key to significant savings in project time and costs, and at the same time results in reduction of several unwanted resource and physical space conflicts [1]. Working in a well organized operational environment with minimum amount of time spent on resolving conflicts and doing tasks not directly related to the scope of the project is an important factor in meeting the project schedule. This is a major incentive for using computer applications to model, simulate, and visualize operations beforehand in order to identify any potential project related events that may cause unexpected delays or conflicts during the course of the real operations, and plan ahead of time to avoid such scenarios during actual construction [2].

For a relatively small operation, reviewing, interpreting, verifying, and validating the results of a simulation model can be done manually using statistical data, flowcharts, flow diagrams, and other numerical tools [1]. However, as the size and complexity of the operation increases and with the introduction of more resources and activities within the

operation, communicating the results of a simulation model for verification and validation purposes turns into a crucial need that has to be done in a timely and efficient manner. One of the effective methods of verifying and validating the results of a simulation model is to visualize the flow of activities in a chronologically and spatially accurate manner. 3D visualization of simulated operations has traditionally been performed in Virtual Reality (VR). In order to create convincing VR animations, detailed data about the process and the environment has to be obtained. Such data must be able to describe the simulation, 3D CAD models, facility under construction, and terrain topography (Model Engineering). As the size and complexity of the operation increases, data collection becomes a time consuming task. This directly translates into loss of project financial and human resources which could otherwise be saved and used more productively. In an effort to remedy this situation, an alternate approach based on Augmented Reality (AR) was designed and implemented in this research to create mixed views of real existing facilities on the jobsite and virtual CAD objects involved in construction. The application of AR in animating simulated construction operations has great potential in reducing the Model Engineering and data collection tasks, and at the same time helps in creating visually convincing output [3].

Several researchers have recently focused on the applications of VR in verifying and validating simulated operations [4, 5, 6, 7, 8, 9]. Although VR provides an environment in which computer generated representation of a simulated operation can be viewed and studied, there are a number of disadvantages to its application. The significant amount of time and effort invested in CAD Model Engineering (i.e. creating, rendering, and managing 3D CAD models of the simulation entities) is a major challenge in VR, especially as the size and complexity of the operation grows [10]. Furthermore, since there is no notion of the real world in a VR-based visualization, the observer of the VR scene has a relatively low level of interaction with and involvement in the scene that are usually defined by a limited maneuvering area (e.g. immersive VR laboratory rooms) or hand motions (e.g. using a pinch glove) [2,7,9].

Unlike VR, the application of AR as a general purpose visualization technique in many scientific and engineering fields is very promising as it can potentially provide a mixed environment in which the observer of the scene can completely interact with both real and virtual objects. At the same time, the introduction of the real existing environment into the visualization can potentially lead to more thoughtful insights. It can also lead to a reduction of time and effort otherwise invested in CAD Model Engineering. However, AR-based visualization applications are still in their early stages of development. Although some disciplines (e.g. medicine, automotive industry, and military) have been using AR as a frontline technology to overcome visualization challenges in their domain [11, 12, 13, 14, 15, 16, 17, 18], work still needs to be done to develop functional and reliable AR-based visualization tools that can be effectively used in areas such as construction and other engineering disciplines.

The presented research investigates and introduces the applicability of AR-based visualization for animating dynamic operations modeled and simulated using Discrete Event Simulation (DES). This Chapter presents the methodology and design of AR-based algorithms, and a powerful animation authoring language together with their practical implementation inside a mobile AR visualization tool. The designed animation language provides a convenient method to automatically author animations of any length and complexity at the operations level of detail using an external software process such as a running DES model. The results of the research have been validated (Chapter 6) by animating several simulated construction operations in outdoor AR.

2.2. Prior Work in Visualization of Simulated Construction Operations

Schematic visualization techniques such as those provided in EZStrobe [19] and ABC [20] can produce useful results in terms of highlighting important events that occur within a simulation. However, they provide no sense of realistic resource motions, nor can they adequately represent the processes occurring within activities. There is also no

notion of physical space and therefore cases dealing with spatial conflicts or congestion cannot be studied in such visualizations. As a result, many researchers have discussed the potential of 3D visualization in validating and verifying the simulation models [9, 21]. Smooth and continuous animation can provide more accurate representation of a simulated operation. It can also help overcome the inadequacies of traditional schematic visualization techniques. Most of these studies [4, 5, 6, 7, 8, 9], however, have been focused on the application of VR and virtual interactive environments. Limited research has been conducted to investigate the applicability of AR for such purpose.

The application of visualization techniques for planning, analysis, and design in construction and civil engineering is relatively new compared to the sizeable amount of AR related research conducted for diverse applications in fields such as manufacturing, medical operations, military, and gaming. For example, Webster et al. [22] presented a system that shows locations of columns behind finished walls, and rebar inside columns. They also presented an AR system to guide workers through the assembly of a space frame. Roberts et al. [23] used AR to overlay locations of subsurface utility lines onto real world views. These applications have been designed to demonstrate the potential of AR in helping maintenance workers avoid buried infrastructure and structural elements. Hammad et al. [24] augmented contextual information on real views of bridges to help inspectors conduct inspections more effectively. Kamat and El-Tawil [25] used AR to study the extent of horizontal displacements sustained by structural elements due to extreme loading conditions. Figure 2.1 shows sample snapshots of previous work in AR animation in construction as presented in [22,23,25].

(a) **(b)** **(c)**

Figure 2.1 – Previous Work in Application of AR Animation in Construction [22,23,25]

Wang and Dunston [26] have studied the potential of AR as an assistant viewer for computer-aided drawing. Behzadan and Kamat [21] studied the applicability of 4D CAD visualization in AR for construction [27] by performing several experiments using time tagged CAD models of building structures.

A common characteristic of the existing AR related research is that, in all cases, the work has led to a final product designed for a specific purpose (i.e. application). Several of these products are only capable of handling static views or snapshots of an augmented scene manually created to serve as the basis for further visual analysis. A key limitation in existing knowledge has been the lack of an automated method to generate dynamic visualizations of simulated operations of arbitrary length and complexity. To the author's best knowledge, none of the conducted AR related research has hitherto focused on the use of an external software process such as a DES model to automatically create and animate simulated operations in 3D outdoor AR.

2.3. Main Contributions

Construction operations usually include a large number of tasks involving different resources (i.e. equipment, labor, material) that interact in complex ways. As a result, illustrating the complex dynamics of a typical construction site solely based on the discrete information obtained from a DES model is a very challenging task. This was the main motivating factor in this research for designing an animation authoring language that would enable a running simulation model to automatically generate a syntactically accurate trace file representing the operations being simulated. The generated trace file consists of chronologically ordered tasks performed during the simulation and is used as the underlying script to create the AR-based animation of the operation. ***The primary contribution of the research presented in this Chapter is an AR animation authoring language that enables modelers to automatically create augmented reality animations of construction operations of arbitrary length and complexity simulated in a DES tool.***

In order to visualize a construction operation, the required components include but are not limited to the facility under construction, equipment, personnel, materials, and temporary structures as well as their possible movements, transformations and interactions. All these components have to be accurately depicted within the augmented scene. In order to depict smooth motion, visual elements must be shown at the right position and orientation several times per second. Due to the amount of detail and precision involved, accurate visualization of construction activities at this level has always been a challenging prospect [28].

Furthermore, features enabling user interactivity within the augmented scene are key factors that have to be integrated into an AR application. In a mobile visualization application, the user should be given the ability to walk through an animation in real time, adjust, modify, or completely change the underlying operational logic of a set of related tasks, properties of a certain resource, or even the overall simulation scenario in order to study the effects of different decisions and outcomes in one experiment. What distinguishes the developed methods in this research from previous work is that in the presented methodology, the user's position inside the 3D augmented world determines what parts of the animation should be displayed. This is discussed in more detail in Chapters 3 and 4 of this dissertation.

The constantly updating augmented space displays only what is visible to a user at a particular location. This means that there are no previously defined viewpoints that restrict the location from where an animation can be observed. Instead, the user can freely change position, and the view direction (i.e. head orientation), while the position and/or orientation of the virtual contents inside the augmented space are continually updated. The procedure of constantly adjusting the position and orientation of the virtual objects overlaid on top of the real world is often referred to as registration, and has been one of the major challenges overcome in this research. To achieve precise registration of virtual objects and create augmented scenes of simulated operations, a fully functional Global Positioning System (GPS) based tracking mechanism has been developed. The details of this mechanism have been described in [29] and in Chapter 3 of this dissertation.

2.4. Technical Approach to Create AR Animations

The application of DES tools to simulate construction processes has been widely investigated [30, 31]. DES is a powerful objective function evaluator that is well suited for the design of construction operations. It runs on the concept of separate activities linked together based on their resource needs and precedence logic. DES as applied to construction operations planning and analysis entails the creation of models that represent how construction operations will be performed. These models consider the different resources that are required to carry out the construction operations, the rules under which the different tasks that compose the operations are performed, the managerial decisions made during the operations and the stochastic nature of events. Once the models are created, the modeled operations can be simulated in the computer and the statistical measures of performance for the operations can be studied [32].

In order to create smooth animations from the results of a DES model, events that mark the beginning and end of discrete activities have to be interpreted and communicated to the visualization tool in a continuous chronological order. To achieve this objective, using an authoring language that automatically creates time stamped events in the form of sequential statements written to an animation trace file is a very viable approach [9]. The generated animation trace file can be then fed into the visualization application to link each simulation entity to a CAD object, and dynamically manipulate CAD objects based on the contents of the trace file.

In addition, to cause objects under the control of such simulations to be aware of, and react to the user motions in the augmented scene, the augmented environment implementation must communicate bi-directionally, on a continuous basis, and at a high speed with location tracking devices. This communication is very critical in obtaining real time position and orientation as the user walks inside the animation to observe the scene from different perspectives. The data acquired from tracking devices are vital parts of the application in order to display the final animated output. The underlying infrastructure established and implemented in this research to communicate with and

extract data from tracking devices is discussed in more details in Chapter 3. Figure 2.2 shows the approach taken in the presented research to use the result of a running DES model in order to generate the animation trace file of an AR animation, upload the required CAD objects inside the user's augmented view, and update the contents of the AR scene based on the user's latest position and orientation obtained from the tracking devices.

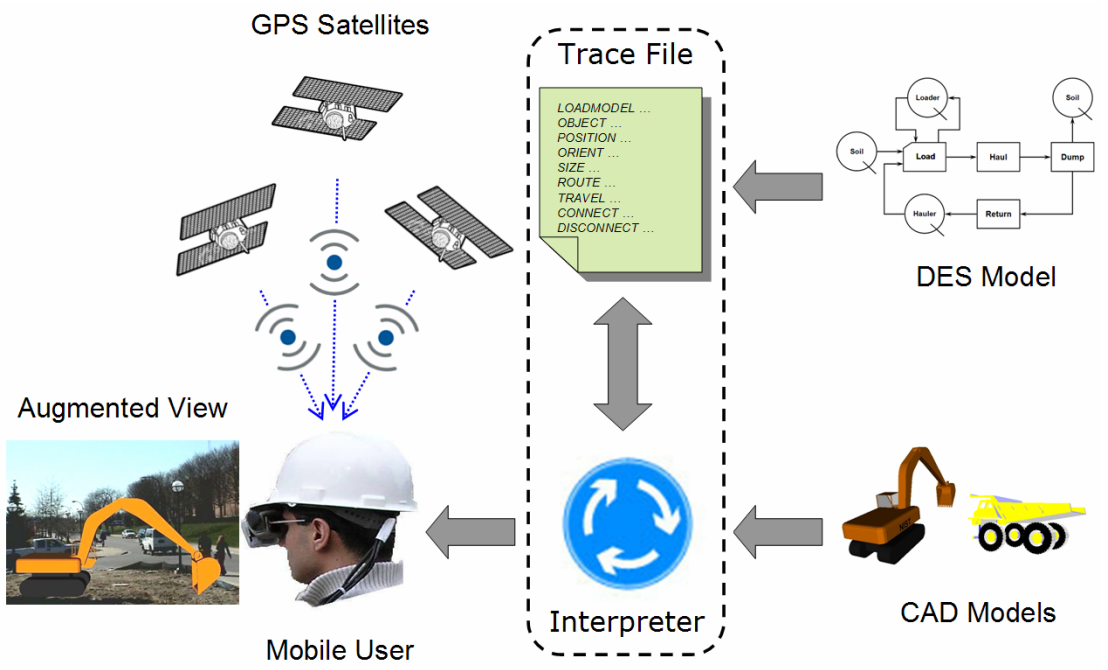


Figure 2.2 – Relation between the DES, CAD and GPS Data, and AR Animation

As shown in the schema presented in Figure 2.2, the resulting AR animation trace file is sequentially interpreted line by line and appropriate CAD objects are uploaded and/or transformed as each line of the trace file is executed. In addition, the positional data coming through the tracking devices connected to the user are simultaneously extracted, integrated, and used to generate an augmented viewing frustum with the user's eye at the center of the projection. Inside this frustum, the contents of the virtual world (i.e. CAD objects of construction entities) are superimposed on top of live video streams of the surrounding world as captured by a video camera installed in front of the user's eye. The result is a dynamic augmented view of the ongoing construction operation which is fully responsive to position and head orientation changes of the user in 3D space [21].

The animation is shown to the user through a Head Mounted Display (HMD). The video camera, head orientation tracking device, and HMD are connected to the user's hard hat. The user can walk freely on the site with minimum physical constraints and observe the animated scenes from different positions. The heart of the system is a laptop computer which is installed and secured inside a backpack. Other devices included in the backpack are a GPS receiver unit, and an external battery pack. A miniature keyboard and a touch pad are also connected to the laptop and carried by the user to provide full interaction capability during the course of the animation when there is no physical access to the laptop computer [29]. Figure 2.3 shows a profile of the user wearing the mobile backpack equipped with all necessary components.

Figure 2.3 – Profile of the User with Mobile Backpack and Registration Devices

2.5. ARVISCOPE Animation Authoring Language

As noted earlier, an authoring language is a critical component of the AR application to extract simulation results from a running DES model. In this research, a powerful, self-contained animation authoring language called ARVISCOPE (acronym for Augmented

Reality Visualization of Simulated Construction Operations) was designed. ARVISCOPE is a high-level 3D animation authoring language that can allow an external software process (e.g. a running DES model) to author a dynamic visualization in AR.

Sequential statements written in this language can describe a smooth and continuous operation of arbitrary length and complexity. The communicated statements (i.e. events) are interpreted by the visualization engine of the AR application. Appropriate data structures, algorithms, and routines are then invoked (Appendix D) to manipulate CAD models and other 3D geometric primitives to present a smooth, accurate representation of the operations. Despite the fact that the ARVISCOPE authoring language is powerful enough to describe the complexities involved in a typical construction operation, the syntax of the language is not very complex.

According to their functionality, ARVISCOPE language statements can be grouped into scene construction, dynamic, and control statements. These statements can be sequentially recorded into and interpreted from a text file referred to as the animation trace file in this dissertation. The animation trace file begins to be parsed as soon as the application starts, the individual statements are processed, and the graphical representation corresponding to the event in each line of the trace file is simultaneously created and depicted inside the augmented view. During this process, the user can freely move in the animated augmented space. Figure 2.4 shows the animation trace file interpretation cycle.

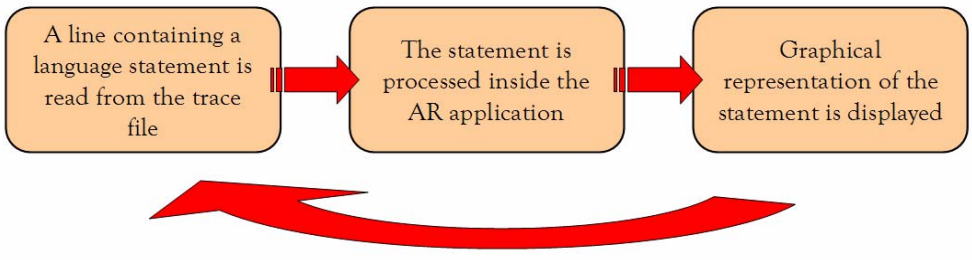


Figure 2.4 – ARVISCOPE Animation Trace File Interpretation Cycle

2.5.1. Scene Construction Statements

Scene construction statements are designed to set up the animation environment and manage the initial and dynamic creation and destruction of simulation entities. This is done by referencing CAD models of relevant resources (e.g. equipment) in different graphical formats, creating instances of specific CAD objects, creating complex CAD meta-objects by assembling simple CAD objects into logical geometric hierarchies, and specifying the initial position and orientation of objects in the desired state on the construction site. This group also contains statements that are used to define routes (i.e. 3D trajectories) that entities may travel on while performing operations. Table 2.1 lists the scene construction statements of ARVISCOPE together with a brief explanation of their functionality. More detailed discussion on the syntax and functionality of each statement together with several examples are presented in Appendix A.

Table 2.1 – List of ARVISCOPE Scene Construction Statements

Statement	Functionality
LOADMODEL	Assign a CAD file to a class of objects
ORIENTMODEL	Change a CAD file orientation
CHANGEMODEL	Change the CAD file assigned to an object
OBJECT	Create an instance of an object class
REMOVE	Remove an object from the scene
CONNECT	Create a child node to a parent node
STICK	Connect an object without changing the size
DISCONNECT	Disconnect a child from its parent node
ROUTE	Define a 3D trajectory for moving objects
POSITION	Place an object in the augmented view
ADJUST	Set properties of a group or an object

2.5.2. Dynamic Statements

Dynamic statements constitute the core of the ARVISCOPE language. This group consists of several statements that can be used to dynamically manipulate instantiated scene objects to depict the performance of a smooth and continuous operation. Statements in this group describe dynamic geometric transformations of scene objects. These transformations change the position, orientation, and scale (i.e. size) of objects in the 3D augmented space to depict the accurate motion objects undergo while performing operations. The most important statements of this group are those describing single elemental motions that a construction resource undergoes during a specific operation. Examples of such statements are `TRAVEL`, `ORIENT`, and `SIZE`. A time-stamped sequence of an arbitrary number of such elemental motions can effectively describe a smooth, continuous 3D rendition of the pertinent construction operation. Table 2.2 lists the scene construction statements of ARVISCOPE together with a brief explanation of their functionality. More detailed discussion on the syntax and functionality of each statement together with several examples are presented in Appendix A.

Table 2.2 – List of ARVISCOPE Dynamic Statements

Statement	Functionality
`HRZORIENT`	Change horizontal orientation
`VRTORIENT`	Change vertical orientation
`SIDEORIENT`	Change side orientation
`ORIENT`	Change orientation by a certain amount
`ORIENTTO`	Change orientation to a target value
`TRAVEL`	Move an object on a route for a certain duration
`TRANSFER`	Move an object on a route at a certain speed
`SHIFT`	Move an object by a certain amount
`SHIFTTO`	Move an object to a certain location
`SIZE`	Change the scale of an object
`SIZETO`	Change the scale to a target value

2.5.3. Control Statements

The primary control statement in ARVISCOPE is `SIMTIME`. This statement keeps track of the simulation clock while the animation is running. Every discrete event that is represented by a statement inside the trace file has a preceding `SIMTIME` statement that indicates the simulation time at which the event begins to take place.

Figure 2.5 shows how different ARVISCOPE statements can be used in order to construct a sample augmented scene and manipulate its contents.

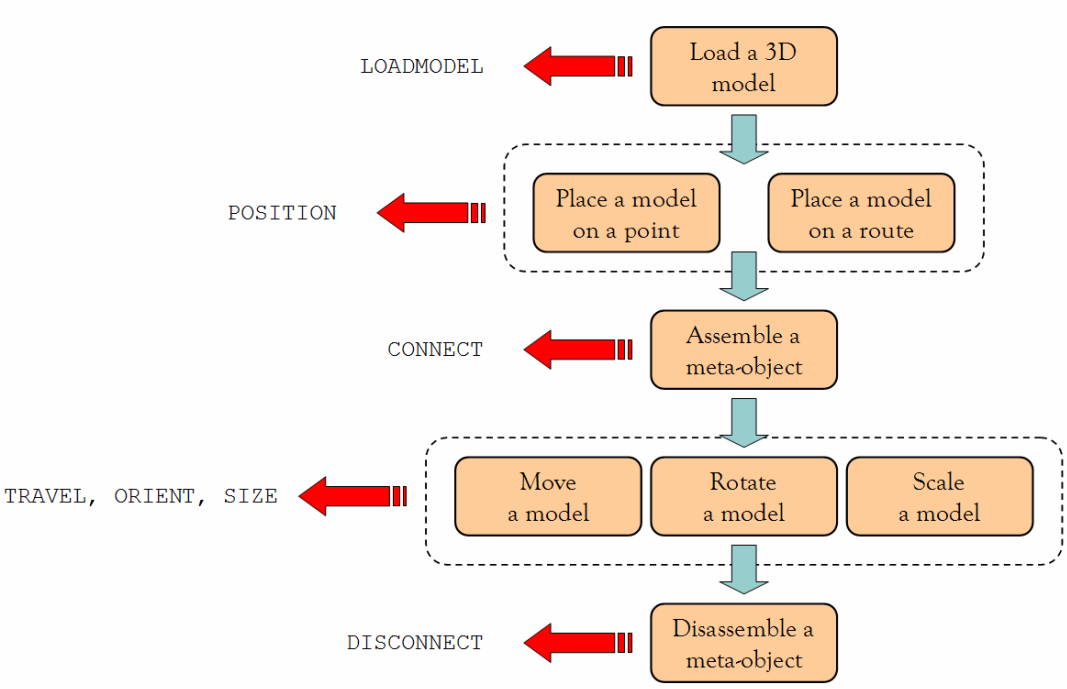


Figure 2.5 – Creating an Augmented Scene Using ARVISCOPE Statements

2.6. Creating an AR Animation Trace File from a DES Model

Figure 2.6 shows the Activity Cycle Diagram (ACD) of a simple earthmoving operation in STROBOSCOPE format. STROBOSCOPE [33] is a programmable and extensible simulation system designed for modeling complex construction operations in detail and

for the development of special purpose simulation tools [34]. The operation shown in Figure 2.6 consists of a simple load-haul-dump-return cycle within which different resources are either used (i.e. loader and hauler) or transferred (i.e. soil). Since the focus of this Chapter is to describe the process of visualizing modeled operations in AR and not the process of modeling earthmoving operations, issues such as volume of the work, productivity, and necessary equipment and steps to perform the operation itself are excluded from the discussion.

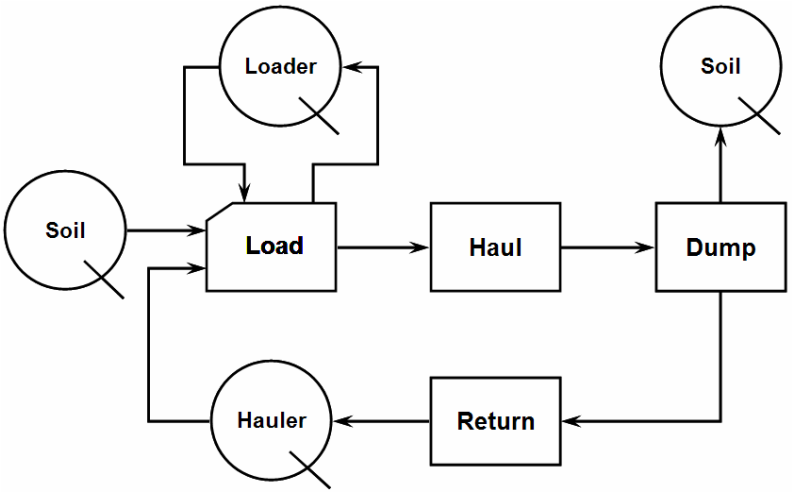


Figure 2.6 – ACD for an Earthmoving Operation in STROBOSCOPE

As noted earlier, an animation trace file is required to create augmented animations of simulated operations in ARVISCOPE. This file can be created either manually (for short animations) or automatically during a simulation run. Manual generation of an animation trace file is typically not practical except in the case of simple demonstrative examples of short animated duration. Automatic generation of a trace file is more recommended since it requires less time and produces more accurate results. Automatic generation of an ARVISCOPE animation trace file requires instrumentation of a simulation model (i.e. including additional code in a simulation model).

For example, Figure 2.7 shows how two new lines are created inside the ARVISCOPE animation trace file of the simple earthmoving operation (Figure 2.6) as a result of a statement added to the STROBOSCOPE model of the same operation. These two lines

will be written to the trace file numerous times with different arguments (e.g. time tag, duration, object name, route name) depending on the specific instance of the activity taking place. The completed trace file will contain other lines of text that will be written out when other parts of the modeled operation take place. Thus, the time ordered sequence of animation statements written out by all the activities in the model during a simulation run constitutes the trace file required to visualize the modeled operations in AR. A more detailed discussion on the process of automatic generation of an ARVISCOPE animation trace file from a running DES model is presented in Appendix C. The reader is recommended to refer to this Appendix to gain a better understanding of the instrumentation process of an animation trace file.

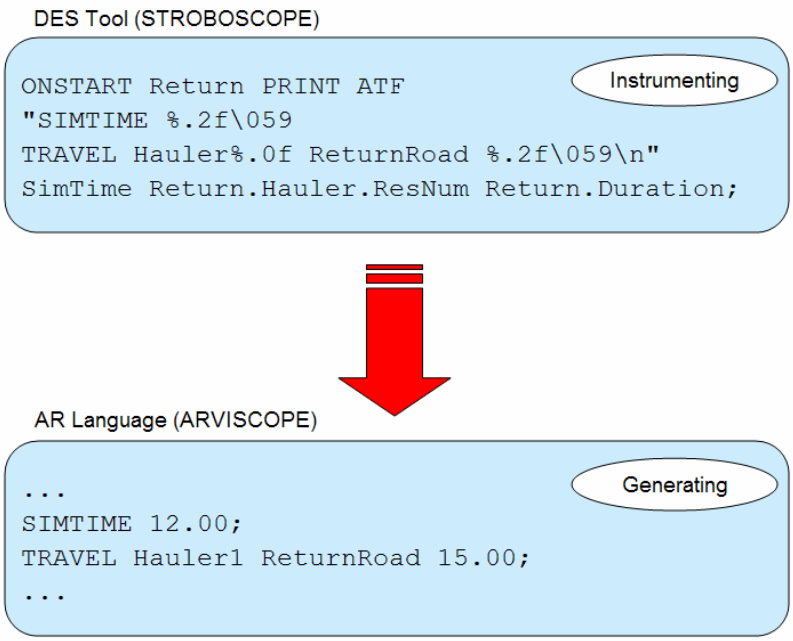


Figure 2.7 – Automated Generation of an ARVISCOPE Animation Trace File

Figure 2.8 shows the algorithm used in the designed AR application to read, extract, and interpret the contents of an animation trace file. As shown in this Figure, once the animation trace file is opened and the animation clock is started, the first available `SIMTIME` statement is read and the specified time argument is extracted. This argument is then compared to the continuously progressing animation time.

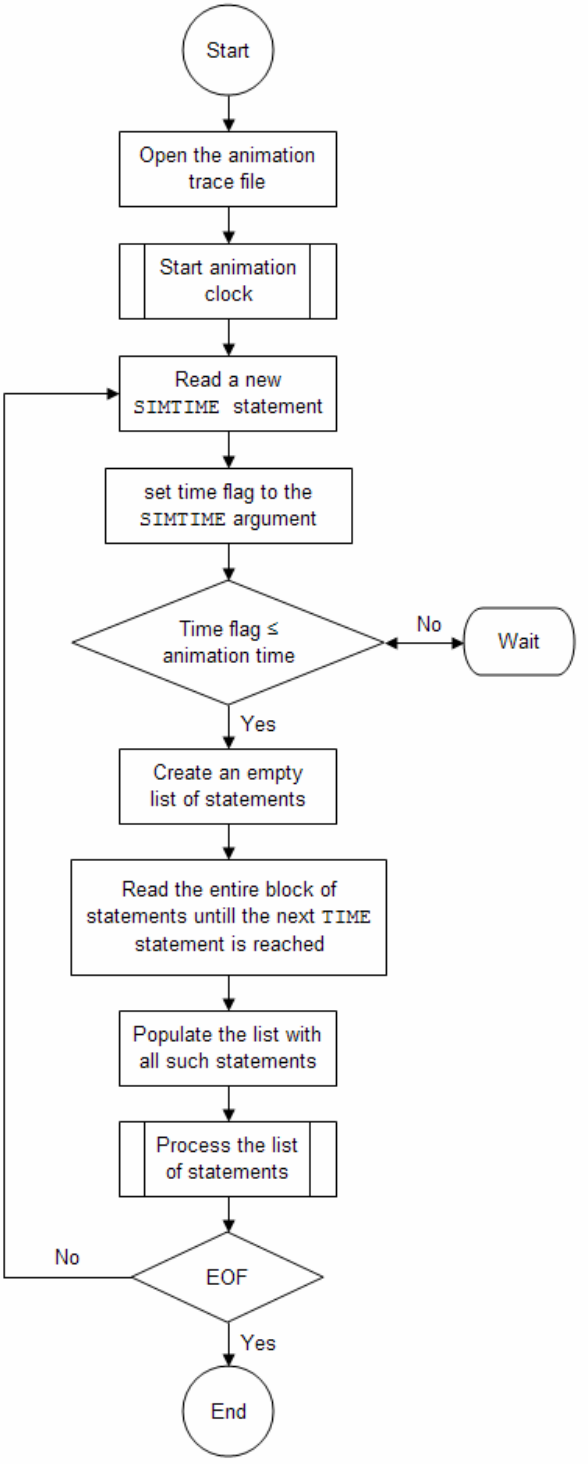


Figure 2.8 – Main Processing Loop of an Animation Trace File in ARVISCOPE

If the time argument is less than or equal to the current animation time, the entire block of statements between the current and next `SIMTIME` statements are read and stored in an

empty list. Otherwise, the application suspends the processing of the trace file until the above condition holds. At this point, each statement inside the list is separately processed and the contents of the augmented scene are accordingly updated. This process continues until the end of animation trace file is reached.

Figure 2.9 presents a small portion of the animation trace file of the earthmoving operation shown in Figure 2.6 in the ARVISCOPE language. By referring to Tables 2.1 and 2.2, the result of processing each statement in this Figure can be explained. First, a route called `ReturnRoad` is defined by specifying the beginning, ending, and two intermediate points in terms of global values of longitude, latitude, and altitude. The 3D models of a hauler and its bucket are then loaded. An instance of each of these models is then created and the bucket is attached to the hauler. This new meta-object, now called `Hauler1`, is then placed on the predefined route. Further processing of the trace file is suspended until simulation time 12 is reached. At this time, `Hauler1` will start moving on the route `ReturnRoad` and this trip requires 15 units of simulation time to complete. During the animation, the ratio of simulated time to viewing time (also known as viewing ratio) is maintained at a constant value specified by the user in the beginning of the animation. For example, if this ratio is 3, the animation will show `Hauler1` at the beginning of route `ReturnRoad` for 4 seconds and then the truck starts traveling on the route for 5 seconds.

```
ROUTE ReturnRoad   (-83.728180,42.287913,260.00)
                   (-83.727064,42.287469,260.00)
                   (-83.726099,42.288374,260.00)
                   (-83.726442,42.289152,260.00);

LOADMODEL Hauler Truck.lwo;
LOADMODEL Bucket Bucket.lwo;
OBJECT Hauler1 Hauler;
OBJECT Bucket1 Bucket;
CONNECT Bucket1 Hauler1 (-7,2.2,0);
POSITION Hauler1 ON ReturnRoad;

SIMTIME 12;
TRAVEL Hauler1 ReturnRoad 15;
```

Figure 2.9 – Portion of a Sample ARVISCOPE Animation Trace File

2.7. ARVISCOPE Language Design Issues

Several basic object handling issues in designing the syntax of the ARVISCOPE language had to be addressed to make it capable of describing complex construction scenes. Two major challenges had to be addressed during the design course of ARVISCOPE:

- Converting global coordinates of an object to user's local coordinate frame when updating the virtual contents of the augmented viewing frustum at each animation frame.
- Converting local coordinates of an object to global coordinate frame when a dependent virtual object is about to be placed independently in the augmented scene.

In this research, the global coordinate system is defined by three major axes: X which is parallel to the equator with its positive direction pointing towards the geographical east, Y which is perpendicular to the earth surface with its positive location pointing upwards, and Z which is parallel to the prime meridian with its positive direction pointing towards the geographical south. X, Y, and Z readings in the global coordinate system are used to determine the longitude, altitude, and latitude of the user as well as CAD objects in the global space. At the same time, each CAD object has its own unique local coordinate system. The definition of the three major axes in a local coordinate system varies between CAD objects and is mainly specified when the CAD model is created in a graphical software such as AutoCAD or 3D Studio. However, to make sure that CAD objects are placed in the global coordinate system with guaranteed consistency, their local coordinate frame should be oriented in a way that it is exactly aligned with the global coordinate frame (e.g. local X axis is parallel to the global X axis).

While objects can be placed in the scene by defining their global coordinates (i.e. longitude, latitude, and altitude), all transformations (i.e. translations, and rotations) have to be done relative to the user. As a result, the application must be able to keep track of the latest local position and orientation of each CAD object inside the user's coordinate

frame. As shown in Figure 2.10, although the user and all CAD objects have their global position determined in the global coordinate system, CAD objects still have to be placed relative to the user's eyes. Therefore, for each CAD object, an additional step has to be executed in order to calculate the relative distance between the user and that object and use this distance to construct or update the transformation matrix corresponding to that CAD object. Only after this matrix is calculated, the CAD object can be correctly placed inside the user's viewing frustum.

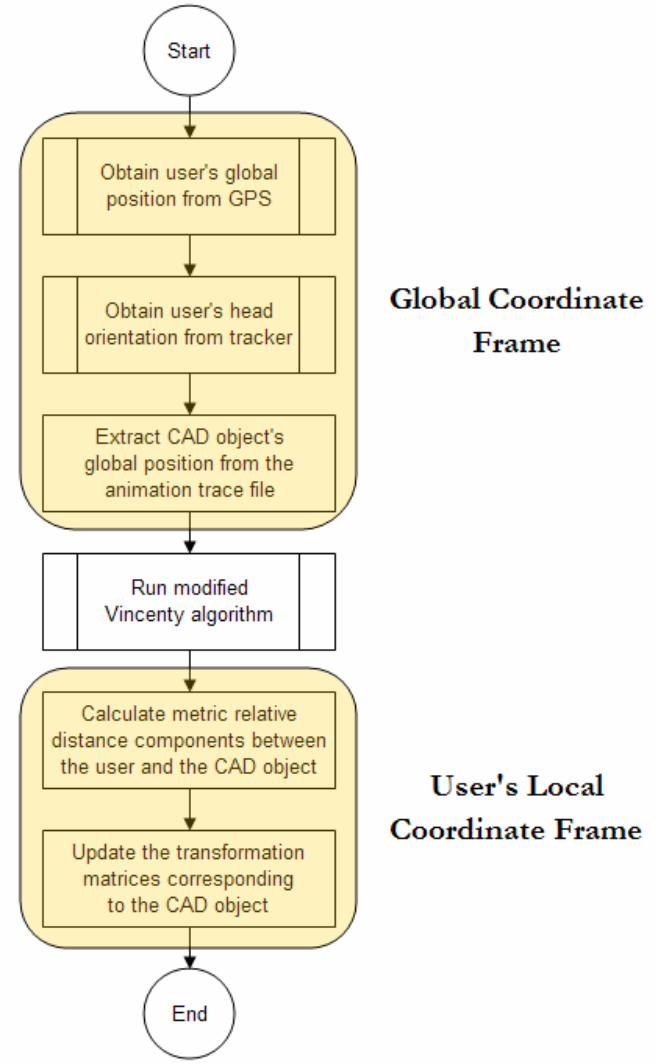


Figure 2.10 – Switching from Global to User's Local Coordinate Frame

Since the user is assumed to be moving, all transformation calculations must be done in real time with the latest global position of CAD objects and the one for the user being

important input parameters to such calculations. Switching from local to global coordinate frames and vice versa and adapting appropriate conversion methods to achieve the expected result have been key challenges in designing ARVISCOPE language statements. The language statements are designed in a way that they can accept and handle positional values in both global and local coordinate frames as arguments when necessary. The following Subsections describe the details and methods followed in this research to design three main ARVISCOPE language statements (i.e. `POSITION`, `ROUTE`, and `DISCONNECT`) that comply with such situations by automatically switching between global and local coordinate frames.

2.7.1. On-Site Positional Measurement Problems

When the number of CAD objects increases the time required to measure the longitude, latitude, and altitude for each point grows dramatically. For an animation to be a precise representation of a simulated system, it is necessary to have contingency in measuring longitude, latitude, and altitude values of different points in the scene. Sometimes, it is practically impossible to measure these values on the site. A good example is a point far beyond physical access (e.g. a point on a body of water) or a number of points with limited distance to be accurately measured with a GPS device (e.g. two CAD objects located a few centimeters from each other while the GPS accuracy is on the same order). Another situation is the case in which a large number of CAD objects are to be placed in a scene and the only known pieces of data are the relative distances between them as opposed to their absolute longitude, latitude, and altitude values. Figure 2.11 shows some of such situations in which providing the global coordinate values for all the individual points on the augmented scene becomes a time consuming and sometimes an impossible task.

As a common practice in drawing the site layout plans, local position of important objects are defined relative to the positions of a number of known surveying benchmarks. Sometimes the only points on the site with known longitude, latitude, and altitude values are benchmarks and the positional values of everything else is measured and calculated

relative to these points. While obtaining the global position values for every item on a construction site is a very time consuming and often impossible process, finding these values for only a few points and placing others relative to them seems to be a more reasonable approach.

Figure 2.11 – Sample Scenarios Involving Positional Measurement Problems

As described in Table 2.1, the `POSITION` statement is used to place a static or a moving object in the augmented scene by defining its global coordinates (e.g. longitude, latitude, and altitude values). To improve the initial design of the `POSITION` statement and make it more powerful and practical, it was further modified to also accept local point

coordinates relative to a reference point with known global coordinates. Hence, a numerical algorithm was developed and implemented in this research to calculate the global position of a point based on the same values of a reference point and the relative distance between the two points. This required establishing methods to convert local to global coordinate values. The initial methods to convert global coordinates to local values in the presented work were adapted from an algorithm originally introduced by Vincenty [35] in which the straight line distance between two points with known global coordinates is calculated [21]. The details of the original Vincenty algorithm are discussed in Chapter 3 of this dissertation. The calculated distance between two points on Earth is entirely a function of where they are located on the Earth's surface. This is due to the fact that the parallels (the rings around the Earth parallel to the Equator) become smaller as they approach the poles and the meridians (the rings around the Earth parallel to the Prime Meridian) converge at the poles. As a direct result, for every one degree change in longitude or latitude, the equivalent displacement in meters will be larger close to the Equator and smaller close to the two poles. These values range between 1 to 69 miles depending on where the location of the point is on the surface of the Earth. Although an average of 60 miles per degree is recommended for approximation, a more precise way to convert changes in degrees into displacements in meters had to be used in this research to be able to create exact graphical representation of a construction site.

Since the initial Vincenty algorithm uses numerical methods to minimize the distance error margin by trial and error, there is no inverse Vincenty method available to convert local values to global coordinates. As a result, a new method was designed in this research which internally uses parts of the original Vincenty algorithm in its calculations. Figure 2.12 shows the difference in terms of input and output parameters between the original Vincenty algorithm [35] and the developed method in this research. As shown in this Figure, the developed method calculates how many meters of displacement in x (z) direction correspond to a one degree change in longitude (latitude) value. As discussed earlier, the answer totally depends on where the point is located on Earth.

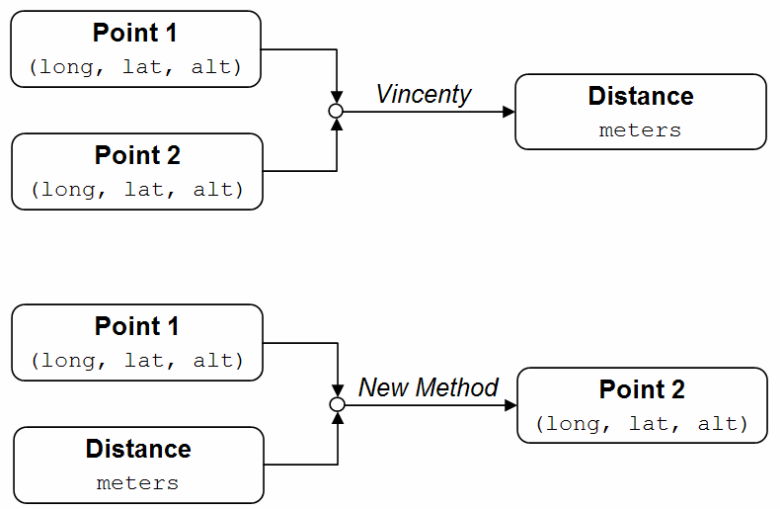
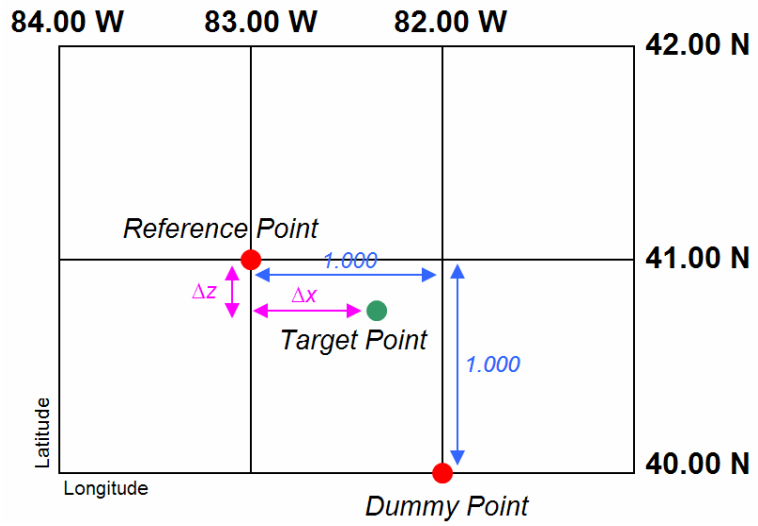


Figure 2.12 – Calculation of a Global Point Coordinates in ARVISCOPE

The points as defined and used in the calculations are shown in the planar view of Figure 2.13. In this Figure, the global position of the *Reference Point* and the relative distance between the *Target Point* and the reference point along the three major axes are known. The goal is to calculate the global position of the target point which can be a point on a route or a point on which a CAD object is to be placed. As shown in Figure 2.13, knowing the global coordinates of the reference point, a *Dummy Point* can be imagined which is exactly one unit (in longitudinal and latitudinal directions) away from the reference point. Based on the coordinates of the reference and dummy points, the developed method can calculate how many meters of displacement correspond to a one degree change in longitude (i.e. `MIn1DegLong`) as well as a one degree change in latitude (i.e. `MIn1DegLat`) by making a number of internal calls to the modified Vincenty algorithm [21]. With a good approximation (and assuming the reference point is close enough to the point of interest), the surface containing the reference and dummy points is assumed planar and linear interpolation (or extrapolation) is used to calculate longitude, latitude, and altitude values of the target point. These steps are presented as pseudo code in Figure 2.13. For sign compatibility, calculated values along X and Y axes are added to the coordinates of the reference point while the calculated value along Z axis is subtracted to obtain the coordinates of the object.




```
DummyPoint.long = ReferencePoint.longitude – 1.000000
DummyPoint.lat = ReferencePoint.latitude – 1.000000
DummyPoint.alt = ReferencePoint.altitude

TargetPoint.long = ReferencePoint.long + [Δx ÷ (MIn1DegLong)]
TargetPoint.lat = ReferencePoint.lat – [Δz ÷ (MIn1DegLat)]
TargetPoint.alt = ReferencePoint.alt + Δy
```

Figure 2.13 – Definition of Reference, Dummy, and Target Points in Planar View

Adding this new functionality to the original syntax of the `POSITION` statement, there are three distinctive ways to place an object in the augmented scene in ARVISCOPE:

1) To place it on a point with known global position values, or
2) To place it on a route with known point coordinates, or
3) To place it relative to a reference point with known global position values.

Figure 2.14 shows different syntax of `POSITION` and `ROUTE` statements in ARVISCOPE. As shown in this Figure, in order to distinguish between the three types of `POSITION` statement, three different keywords are used: `AT`, `ON`, `REL`. In addition, the two different types of `ROUTE` statements can be distinguished using the `REL` keyword.

```
POSITION objectname AT (longitude, latitude, altitude);
POSITION objectname ON RouteName;
POSITION objectname REL ReferencePoint (Δx, Δy, Δz);

ROUTE RouteName (Long1,Lat1,Alt1)…(Longn,Latn,Altn);
ROUTE RouteName REL ReferencePoint (Δx1,Δy1,Δz1)…(Δxn,Δyn,Δzn);
```

Figure 2.14 – Different Syntax of the `POSITION` and `ROUTE` Statements

According to the definition of major axes in ARVISCOPE, `Δx` shows the relative distance between the object and the reference point along the longitudinal axis, `Δy` shows the relative distance between the object and the reference point along the altitudinal axis, and `Δz` shows the relative distance between the object and the reference point along the negative latitudinal axis. Recalling the animation trace file shown in Figure 2.9, the route `HaulRoad` can now be expressed in terms of its end and mid points' coordinates defined relative to the coordinates of a known point (i.e. `BM`). The new definition is shown in Figure 2.15.

```
LOADMODEL Reference BM.ac;
OBJECT BM Reference;
POSITION BM AT (-83.707000,42.29560,265.00);

ROUTE HaulRoad REL BM (20.000,0,-5.500)
                      (112.125,0,44.100)
                      (191.675,0,-56.450)
                      (163.400,0,-142.900);

. . .
```

Figure 2.15 – Defining a Route Using Relative Coordinate Values

A more detailed discussion on the process of automatic generation of an ARVISCOPE animation trace file from a running DES model is presented in Appendix C. The reader is recommended to refer to this Appendix to gain a better understanding of the instrumentation process of an animation trace file.

2.7.2. Disassembling a Virtual Construction Meta-Object

Another major challenge in designing ARVISCOPE statements capable of independently handling the CAD objects in the augmented scene was to design methods to convert local to global coordinate values. As described in Chapter 4, a virtual CAD object in ARVISCOPE is represented by a graphical node in the augmented scene graph. This node itself can be a parent node to some other child node(s). This hierarchical approach of defining children and parent nodes provides a multi-level structure to the augmented scene in which the highest level objects are those who have been initially placed in the scene by defining their global coordinates (i.e. longitude, latitude, and altitude) and each of the lower level objects has their coordinates defined in its parent's local coordinate system. More details on how a hierarchical structure of CAD objects is created, maintained, and updated during the course of the AR animation are presented in Chapter 4. Figure 2.16 shows how the coordinate values of CAD objects in different layers of such a structure are defined.

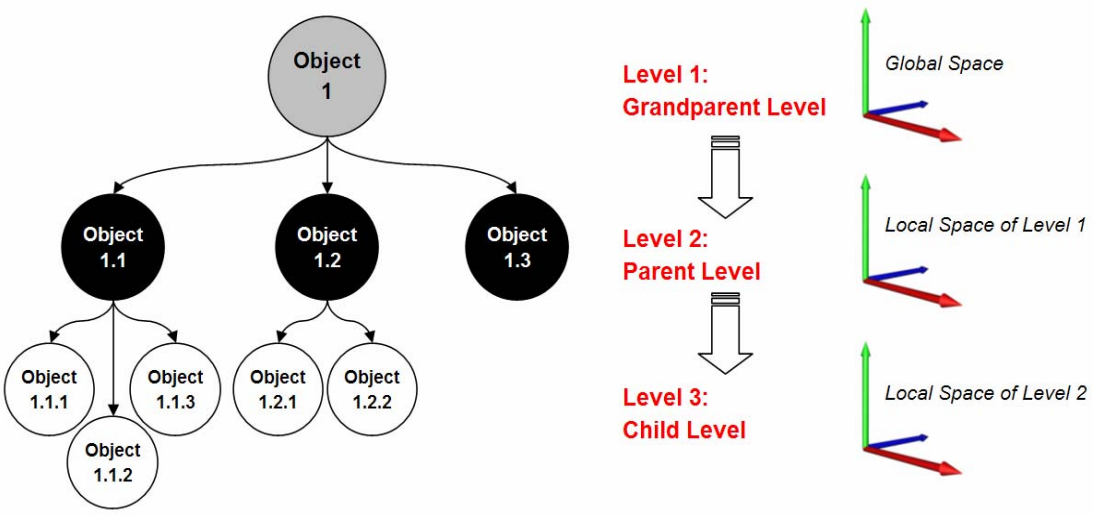


Figure 2.16 – Relation between Coordinate Frames in an AR Scene Hierarchy

The objects at the highest level of the hierarchy shown in Figure 2.16 are placed in the scene using the `PLACE` statements. They can be either placed on an absolute coordinate (in terms of longitude, latitude, and altitude values) or relative to a predefined benchmark point in which case their absolute coordinate values are calculated immediately using the

method described in Subsection 2.7.1. They can also be placed on a route with known endpoint global coordinates. Other objects in the hierarchy are placed relative to their immediate parents using the `CONNECT` statement which takes coordinate values (in terms of x, y, and z) in the local coordinate system of the parent node. All changes in the parent's node position and orientation will affect the corresponding values of its child objects simultaneously. The animation only keeps track of the child nodes' local coordinate values as opposed to their global position. As a result, just before a child node has to be separated from its parent node, one additional step has to be executed in order to calculate the global coordinate values of the object represented by that node so that it can be further manipulated independently inside the augmented scene and relative to the user. Figure 2.17 shows an example of how a child node is moved inside the animation by changing the transformation values of its parent node.

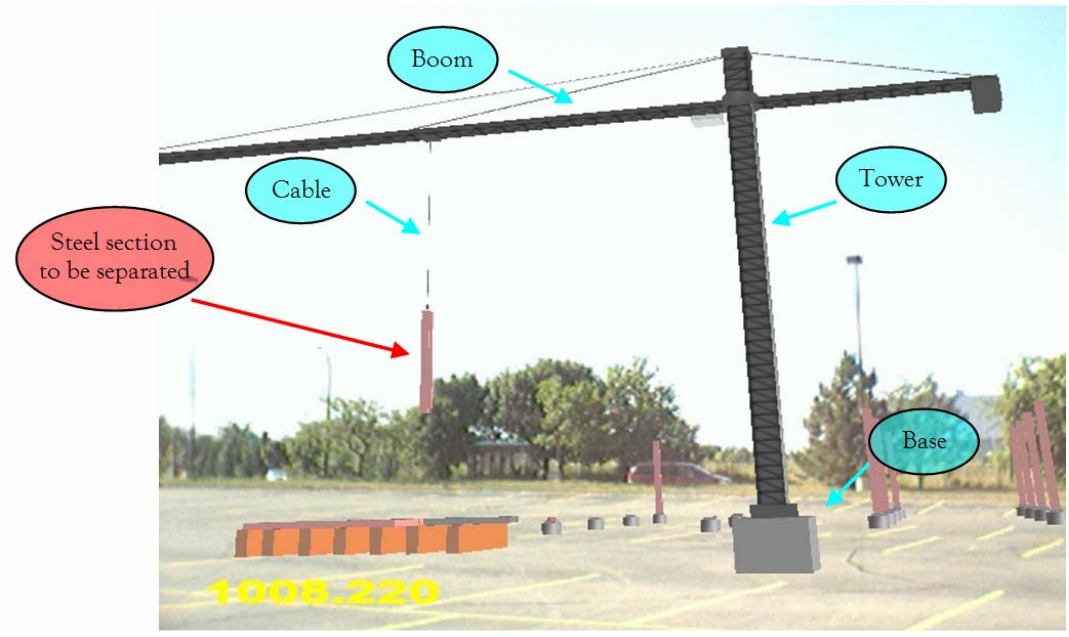


Figure 2.17 – Relation between Coordinates Frames of Different CAD Objects

In Figure 2.17, a CAD meta-object of a tower crane is shown in which the highest level parent node is the *Base* node. The *Tower*, *Boom*, and *Cable* nodes are connected in the scene graph to form the complete tower crane object. This is achieved by first placing the *Base* node globally in the scene. The position of the *Tower* is defined inside the local

coordinate frame of the *Base*. The *Boom* is placed locally inside the *Tower* coordinate frame. Also, the *Cable* node is a child of the *Boom* node. As shown in this Figure, a *Steel Section* has been defined as a child object to the *Cable* and hence is transformed (i.e. picked up, rotated, and installed) in the augmented scene by manipulating the transformational values of its immediate parent node (i.e. *Cable* node). As soon as the steel section is in its final position and has to be separated from the cable and installed on the foundation, its global coordinate values have to be calculated and used to find its relative transformation to the user for subsequent animation frames. The problem of conversion from local to global coordinate values has been addressed in this research using the concept of "transformation chain" which is shown in Figure 2.18.

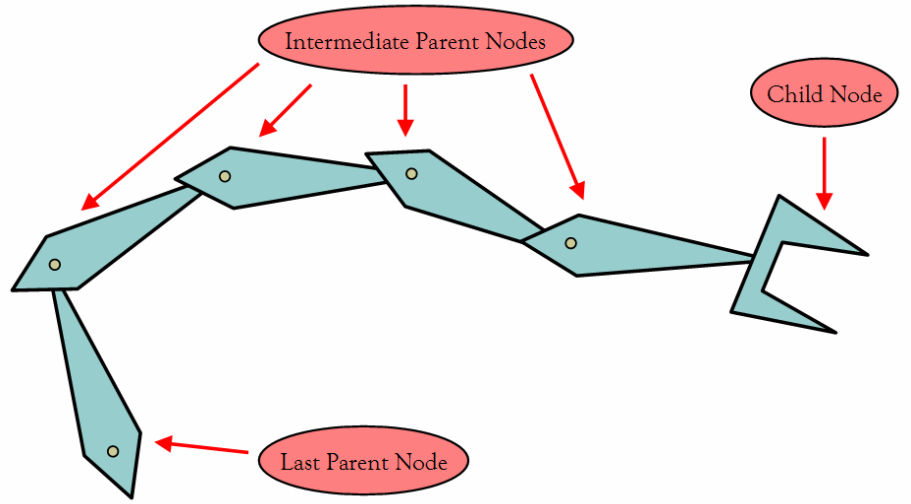
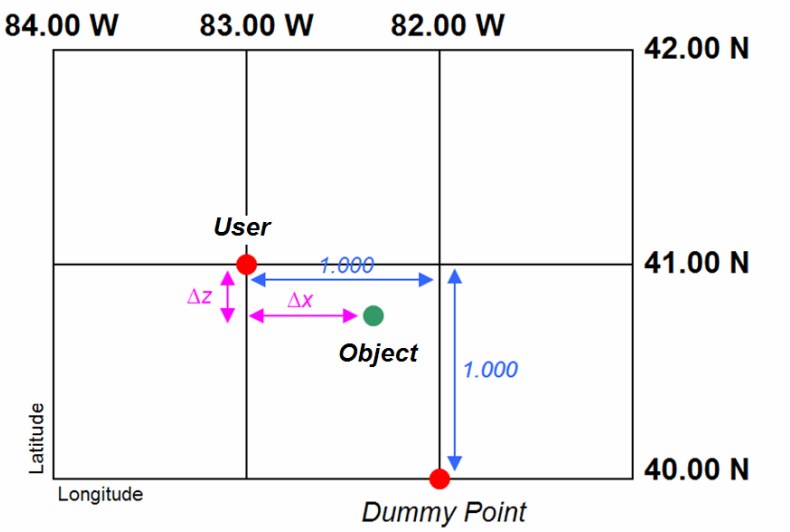


Figure 2.18 – Transformation Chain between the Lowest and Highest Nodes

In this Figure, to find the transformation values of the lowest level node (i.e. end gripper of the robotic arm) inside the coordinate frame of the highest level node (which is the global coordinate frame), and considering the fact that the definition of different axes in different levels of the hierarchy is consistent, all the transformation matrices connected to nodes at different layers (starting from the lowest level node to the highest level node) are multiplied together. The result of this matrix operation is a new transformation matrix describing the transformation of the child object inside the global coordinate frame. Since the position of the user in terms of longitude, latitude, and altitude values are also known in the same coordinate frame, the method previously described in Subsection 2.7.1 can be

applied to find the global position of the child object as well. Following this method, the *Reference Point* represents the user while the *Target Point* describes the position of the newly disconnected child node. This is shown in Figure 2.19.




```
Object.long = User.long + [Δx ÷ (MIn1DegLong)]
Object.lat = User.lat − [Δz ÷ (MIn1DegLat)]
Object.alt = User.alt + Δy
```

Figure 2.19 – Calculating the Position of a Newly Disassembled Child Node

Figure 2.20 shows the algorithm designed and implemented in this research to find the final transformation matrix of a child node inside the global coordinate frame. Following this algorithm and recalling the example presented in Figure 2.17, the global position of the steel section can be calculated as,

$$T_{steel}^{user} = T_{base}^{user} \times T_{tower}^{base} \times T_{boom}^{tower} \times T_{cable}^{boom} \times T_{steel}^{cable}$$

In which T_C^P denotes the transformation matrix of the C (child) node inside the local coordinate frame of the P (parent) node.

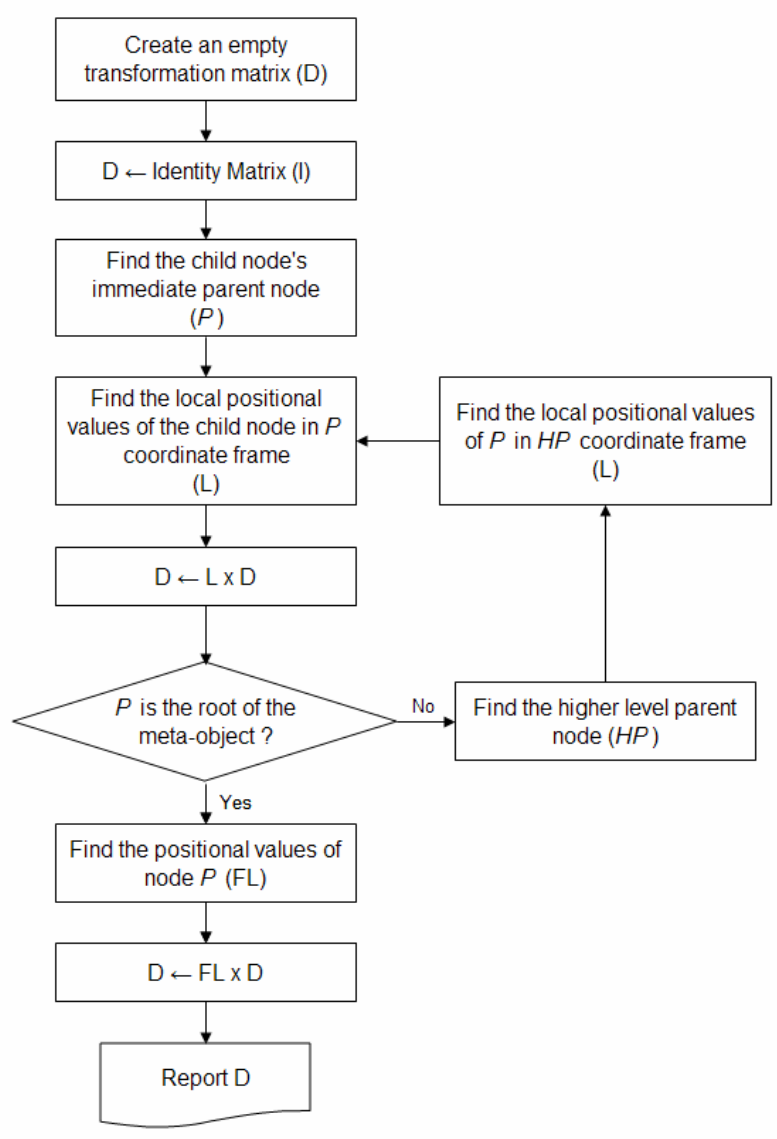


Figure 2.20 – Designed Transformation Chain Algorithm

2.8. Summary and Conclusions

Creating smooth and chronologically accurate animated scenes of a simulated operation modeled in a DES tool requires careful assignment of time tags to discrete events occurring during a simulation run. These time tags can then be used to generate an animation scenario which can be interpreted by an AR visualization application. Several steps need to be taken to create a logical link between a running DES model and an AR animation. First, an animation trace file has to be generated. The generated trace file has

to be interpreted by the AR visualization application and required graphical representations (i.e. CAD objects) have to be loaded and displayed in their appropriate position and orientation inside the user's viewing frustum. Since the user of the AR application is given full mobility inside the animation, the contents of this frustum have to be completely sensitive to changes in user's position and head orientation. This requires the AR application to be constantly communicating with registration devices (e.g. GPS and head orientation tracker) and using their output to calculate the coordinates and update the contents of the viewing frustum in real time.

The author has successfully designed and implemented methods and algorithms to address the above challenges in creating AR animations of simulated construction operations. In this Chapter, the details of an animation authoring language called ARVISCOPE designed to validate the functionality and effectiveness of the methods described above were presented. Although the main focus of this Chapter was on construction processes, most of the findings of this research are generic and widely applicable to other fields of science and engineering where the need to animate and communicate simulated operations is as important as that in construction.

2.9. References

[1] Halpin, D. W., and Riggs, L. S. (1992), Planning and Analysis of Construction Operations, John Wiley & Sons Inc., New York, NY.

[2] Rojas, E. M. (2000). "Virtual Environments for Construction Engineering and Management Education", In Proceedings of the Construction Congress VI, Reston, VA, 263-270.

[3] Behzadan, A. H., and Kamat, V. R. (2005), "Visualization of Construction Graphics in Outdoor Augmented Reality", In Proceedings of the Winter Simulation Conference (WSC), IEEE, Orlando, FL, 1914-1920.

[4] Op den Bosch, A. (1994), "Design/Construction Process Simulation in Real-Time Object-Oriented Environments", PhD Dissertation, Georgia Institute of Technology, Atlanta, GA.

[5] Barnes, M. R. (1997), "An Introduction to QUEST", In Proceedings of Winter Simulation Conference (WSC), IEEE, Atlanta, GA, 619-623.

[6] Bishop, J. L., and Balci, O. (1990), "General Purpose Visual Simulation System: A Functional Description", In Proceedings of the Winter Simulation Conference (WSC), IEEE, New Orleans, LA, 504-512.

[7] Rohrer, M. W. (2000), "Seeing Is Believing: The Importance of Visualization in Manufacturing Simulation", In Proceedings of the Winter Simulation Conference (WSC), IEEE, Orlando, FL, 1211-1216.

[8] Rohrer, M. W., and McGregor, I. W. (2002), "Simulating Reality Using AUTOMOD", In Proceedings of the Winter Simulation Conference (WSC), IEEE, San Diego, CA, 173-181.

[9] Kamat, V. R. (2003), “VITASCOPE: Extensible and Scalable 3D Visualization of Simulated Construction Operations”, PhD Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[10] Brooks, Jr., F. P. (1999), “What’s Real About Virtual Reality?”, Journal of Computer Graphics and Applications, 16(6), IEEE, Piscataway, NJ, 16-27.

[11] Feiner, S., MacIntyre, B., Hollerer, T., Webster, A. (1997), ”A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment”, In Proceedings of the International Symposium on Wearable Computing (ISWC ’97), Cambridge, MA, 74-81.

[12] Thomas, B., Close, B., Donoghue, J., Squires, J., Bondi, P., Morris, M., and Piekarski, W. (2000), “ARQuake: An Outdoor/Indoor First Person Augmented Reality Application”, In Proceedings of the 4th International Symposium on Wearable Computers (ISWC), IEEE, Atlanta, GA, 149-146.

[13] Barfield, W., and Caudell, T. (2001), Fundamentals of Wearable Computers and Augmented Reality, Lawrence Erlbaum Associates, London, UK.

[14] Gleue, T., and Dähne, P. (2001), “Design and Implementation of a Mobile Device for Outdoor Augmented Reality in the Archeoguide Project”, In Proceedings of the 2001 Conference on Virtual Reality, Archeology, and Cultural Heritage, ACM, Glyfada, Greece, 161-168.

[15] Livingston, M., Rosenblum, L., Julier, S., Brown, D., Baillot, Y. (2002), “An Augmented Reality System for Military Operations in Urban Terrain”, In Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC ‘02), Orlando, Florida, 1-8.

[16] Suthau, T., Vetter, M., Hassenpflug, P., Meinzer, H., and Hellwich, O. (2002), A Concept Work for Augmented Reality Visualization Based on a Medical Application in Liver Surgery, Technical University Berlin, Commission V, WG V/3.

[17] Brown, D., Julier, S., Baillot, Y., and Livingston, M. (2003), "An Event-Based Data Distribution Mechanism for Collaborative Mobile Augmented Reality and Virtual Environments", In Proceedings of the Virtual Reality Conference, IEEE, Los Angeles, CA, 23-29.

[18] Piekarski, W., and Thomas, B. H. (2003), "ARQuake - Modifications and Hardware for Outdoor Augmented Reality Gaming", In Proceedings of the 4th Australian Linux Conference, Perth, Australia.

[19] Martinez, J. C. (2001), "EZStrobe – General-Purpose Simulation System Based on Activity Cycle Diagrams", In Proceedings of the Winter Simulation Conference (WSC), IEEE, Arlington, VA, 1556-1564.

[20] Shi, J. J., and Zhang, H. (1999), "Iconic Animation of Construction Simulation", In Proceedings of the Winter Simulation Conference (WSC), IEEE, Phoenix, AZ, 992-997.

[21] Behzadan, A. H., and Kamat, V. R. (2007), "Georeferenced Registration of Construction Graphics in Mobile Outdoor Augmented Reality", Journal of Computing in Civil Engineering, 21(4), American Society of Civil Engineers (ASCE), Reston, VA, 247-258.

[22] Webster, A., Feiner, S., MacIntyre, B., Massie, W., and Krueger, T. (1996), "Augmented Reality in Architectural Construction, Inspection and Renovation", In Proceedings of the 3rd Congress on Computing in Civil Engineering, American Society of Civil Engineers (ASCE), Anaheim, CA, 913-919.

[23] Roberts, G. W., Evans, A., Dodson, A., Denby, B., Cooper, S., and Hollands, R. (2002), "The Use of Augmented Reality, GPS, and INS for Subsurface Data Visualization", In Proceedings of the FIG XXII International Congress, Washington, D.C.

[24] Hammad, A., Garrett, J. H., and Karimi, H. (2004). "Location-Based Computing for Infrastructure Field Tasks." Telegeoinformatics: Location-Based Computing and Services, CRC Press, 287-314.

[25] Kamat, V. R., and El-Tawil, S. (2007), "Evaluation of Augmented Reality for Rapid Assessment of Earthquake-Induced Building Damage", Journal of Computing in Civil Engineering, 21(5), American Society of Civil Engineers (ASCE), Reston, VA, 303-310.

[26] Wang, X., and Dunston, P. S. (2006), "Potential of Augmented Reality as an Assistant Viewer for Computer-Aided Drawing", Journal of Computing in Civil Engineering, 20(4), American Society of Civil Engineers (ASCE), Reston, VA, 437-441.

[27] McKinney, K., Kim, J., Fischer, M., and Howard, C. (1996), "Interactive 4D-CAD", In Proceedings of the 3rd Congress on Computing in Civil Engineering, American Society of Civil Engineers (ASCE), New York, 383–389.

[28] Kamat, V. R., and Martinez, J. C. (2003), "Automated Generation of Dynamic, Operations Level Virtual Construction Scenarios", Electronic Journal of Information Technology in Construction (ITcon), 8, Special Issue on Virtual Reality Technology in Architecture and Construction, Royal Institute of Technology, Stockholm, Sweden, 65-84.

[29] Behzadan, A. H., Timm, B. W., and Kamat, V. R. (2008), "General Purpose Modular Hardware and Software Framework for Mobile Outdoor Augmented Reality Applications in Engineering", Journal of Advanced Engineering Informatics, 22(1), Elsevier Science, New York, NY, 90-105.

[30] Tucker, S. N., Lawrence, P. J., and Rahilly, M. (1998), “Discrete-Event Simulation in Analysis of Construction Processes”, CIDAC Simulation Paper, Melbourne, Australia, 1-14.

[31] Martinez, J. C., and Ioannou, P. G. (1999), “General-Purpose Systems for Effective Construction Simulation”, Journal of Construction engineering and Management, 125(4), American Society of Civil Engineers (ASCE), Reston, VA, 265-276.

[32] Kamat, V. R., and Martinez, J. C. (2002), “Scene Graph and Frame Update Algorithms for Smooth and Scalable 3D Visualization of Simulated Construction Operations”, Journal of Computer-Aided Civil and Infrastructure Engineering, 17(4), Blackwell Publishers, Boston, MA, 228-245.

[33] Martinez, J. C. (1996), “STROBOSCOPE: State and Resource Based Simulation of Construction Operations”, PhD Dissertation, University of Michigan, Ann Arbor, MI.

[34] Kamat, V. R., and Martinez, J. C. (2001), “Visualizing Simulated Construction Operations in 3D”, Journal of Computing in Civil Engineering, 15(4), American Society of Civil Engineers (ASCE), Reston, VA, 329-337.

[35] Vincenty, T. (1975), “Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations”, Survey Review, 176, Maney Publishing, Leeds, UK, 88-93.

Chapter 3

Continuous User 3D Spatial Tracking in the Augmented World

3.1. Introduction

By definition, Augmented Reality (AR) provides the user with an environment in which both groups of real and virtual objects coexist and possibly change position and orientation over time. To create the illusion for the observer of the scene that these two groups of objects are mixed into a single scene, they have to be precisely aligned and maintain their relative position and orientation to the observer at each instant of time. In the realm of AR, this is often referred to as object registration [1]. Registration of real objects is done automatically as the user moves in the scene since the human visual system is capable of interpreting information from visible lights to build a 3D representative world of the surrounding environment visible through the eyes. Figure 3.1 shows how the registration process is performed for real objects. In this Figure, the observer first looks at the front view of two pieces of construction equipment (Figure 3.1.a) and eventually starts moving in the scene in order to see the rear view of the same equipment (Figure 3.1.b). Although the observer is moving around and changing position and head orientation, the observer's visual system continuously registers the two real objects in the visible view and hence objects seem to stay fixed at their location.

(a) **(b)**

Figure 3.1 – Automatic Registration of Real Objects [2]

The above process fails when it comes to registering virtual CAD objects as they are not actually present in the real scene. As a result, additional methods have to be applied in order to precisely calculate the position and orientation of CAD objects inside the visible augmented world. These calculations should be performed in a way that at each instant of time, CAD objects are perceived as authentic by the user's mind. This is important because the human eye is adept at noticing minute discrepancies in objects oriented incorrectly in 3D space. Achieving correct registration of virtual objects requires continuous tracking of accurate user position and orientation as well as the user's relative distance to each virtual object in the field of view when rendering and overlaying the virtual CAD objects at their corresponding real world locations.

Figure 3.2 shows the registration process of CAD objects in more detail. In this Figure, there are a number of real pieces of construction equipment in the background. The objective of this example is to superimpose a CAD model of a virtual excavator over the real background at a specific location and orientation (facing north). In order to achieve this, the AR system must track the user's movement and head orientation repeatedly over time. Using the obtained tracking data, local transformation between the user and the CAD object is constantly calculated and applied to the object. As a result, the composite augmented view is updated continuously over time with the position and orientation of the virtual CAD model being adjusted based on the user's movements [1].

Figure 3.2 – Precise Registration of Virtual CAD Objects in an Augmented Scene

Figures 3.2.a to 3.2.d depict accurate augmented views an AR scene observer expects to see when looking at the scene from four different locations (a, b, c, and d) indicated on the plan view. In each case, the final result is a mixed view of real and virtual objects and is sensitive to the observer's location and orientation. When the observer stands at point (a) and looks towards the north, the rear end of the virtual excavator is expected to be seen (Figure 3.2.a). At location (b), the observer is standing in front of the virtual excavator looking towards the south. In this case the observer must see the excavator from the front (Figure 3.2.b). When the observer is at location (c) looking west, the right side of the excavator must be visible (Figure 3.2.c). Finally, in location (d) when the observer looks east, only the left face of the virtual excavator must be visible in the composite AR output (Figure 3.2.d). In summary, the augmented virtual excavator must stay fixed to its real-world location while the AR observer navigates around it on the jobsite. In this research, this was achieved using Global Positioning System (GPS) and Three Degree-of-Freedom (3DOF) head orientation tracking to monitor the observer's (user's) position and head orientation.

3.2. Indoor and Outdoor User Tracking

Each operational environment defines a different set of challenges for accurate user registration in mobile augmented reality. For indoor applications, since the dimensions of the environment are predefined (e.g. a laboratory room), physical restrictions on user's movements and location can be conveniently determined by the AR system. In addition, an indoor environment can be deemed *prepared* by placing sensors, optical markers, or location tracking cameras at important locations. These fixed devices simplify the task of registration while also serving as reference points when incorporating virtual CAD objects into the scene.

For example, Yuan et al. [3] proposed a registration method for AR systems based on the coordinates of four specified points in space (e.g. four corners of an optical marker) to register augmented contents on top of real views. Kaufmann and Schmalstieg [4] designed a mobile collaborative AR system called Construct3D for mathematics and geometry education. Construct3D uses optical tracking to appropriately register the virtual geometric shapes on top of the live background. Figure 3.3 shows snapshots of two indoor AR visualization tests presented in [5,6] in which virtual CAD objects were superimposed on the real scene views based on the visibility of optical markers.

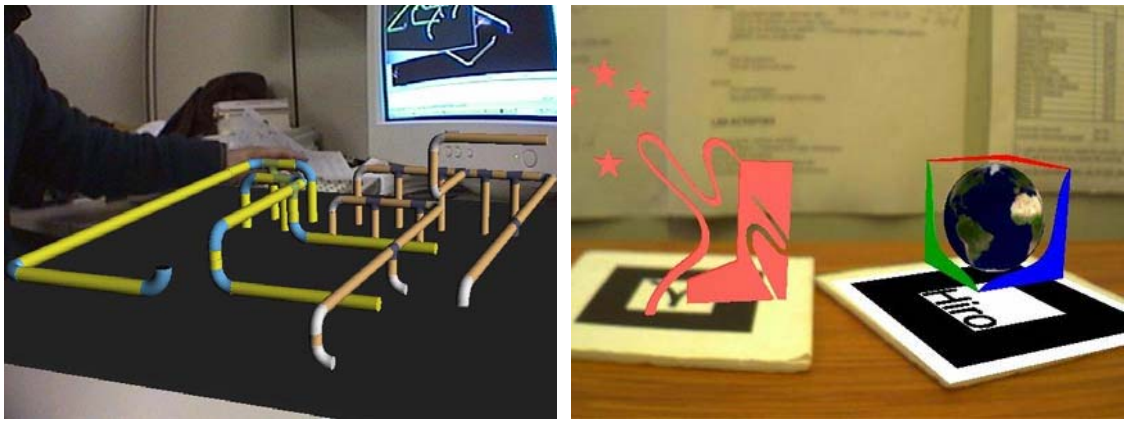


Figure 3.3 – Optical Marker-Based Indoor AR Visualization Tests [5, 6]

Unlike indoor environments, a user operating outdoors is located in a wide maneuvering area which provides unlimited combinations of possible viewing locations and orientations. This essentially means that most of the indoor positioning techniques that are based on preinstalled infrastructure such as trackers and markers may not be robust or scalable enough to accommodate all possible user situations. Similarly, the natural variable lighting condition present in an unprepared outdoor environment makes it difficult to employ optical-based tracking methods for these types of AR applications [7]. Therefore, mobile AR systems require methods capable of obtaining non-intrusive measurements of both physical position and viewpoint orientation, without imposing restrictions on the user's freedom of movement [1].

The use of AR has been explored in several engineering and scientific domains. In addition, the idea of designing and implementing an integrated AR system with all the necessary components installed and secured in one platform has been introduced in a limited number of previous studies, each applicable to a specific field of engineering or scientific research [8]. However, the prototypes designed and implemented in all prior studies have been custom created for the respective research projects. None of them presents a general purpose and modular hardware and software interface in a way that can be readily shared and reused by researchers exploring AR in other engineering fields.

Hence, the main motivation for this part of the research was to remedy this situation by designing a general purpose extensible and reusable mobile architecture addressing both hardware and software issues in the form of an AR backpack and a modular core software framework to perform the task of accurate registration. The designed architecture can be easily adapted and extended by AR application developers for use in any domain that requires a similar hardware configuration. The presented design takes advantage of the GPS and 3DOF head orientation data in order to describe a user's real time location and heading. Figure 3.4 shows the latest version of an AR platform called UM-AR-GPS-ROVER developed in this research. The initial version of UM-AR-GPS-ROVER which was designed to test the registration and graphics algorithms will be discussed in Section 3.7. The components of the initial platform were later modified

based on several product studies and case analyses that led to the design presented in Figure 3.4. The details of these studies will be presented in Sections 3.4 and 3.5. Based on the above discussion, this platform had to be equipped with:

- Computing devices capable of rapid position calculation and image rendering including an interface for external input (i.e. both user commands and a video capturing the user's environment),
- An interface to display the final augmented view to the user, and
- External power source for the hardware components to ensure continuous operation without restricting user mobility.

The design also had to take into account ergonomic factors to avoid user discomfort after long periods of operation.

Figure 3.4 – Overview of the UM-AR-GPS-ROVER Hardware Framework

Another critical component in the designed mobile AR architecture was a robust software interface. This interface must facilitate the acquisition of positional data (i.e. longitude,

latitude, and altitude) from the GPS receiver as well as head orientation measurements (i.e. yaw, pitch, and roll angles) from a 3DOF head tracker. It must also handle captured video scenes of the user's surroundings, and include a video compositor engine that renders virtual objects onto the scenes of the real world and displays the combined image to the user in real time.

The presented architecture takes advantage of an Object Oriented Design (OOD) by assigning a separate code module (i.e. class) to each hardware component. These individual modules are interconnected by the core platform to work in parallel. This way, future replacements, modifications, and upgrades of any peripheral component will not affect the overall integrity of the system. Each code module is device independent as long as its corresponding device provides the system with output that follows a standard format.

3.3. Main Contributions

The objective of this step of the work was to design and implement a modular platform that is both extensible and reusable and can be conveniently adapted for any AR related domain.

Reusability, as referred to in this dissertation, is defined as the measure of generality of the framework which allows potential AR application developers to use the provided methods of the presented framework in their own AR application with minimum level of effort required to modify or adjust the low level software interface. Similarly, the term extensibility in this dissertation describes the fact that the AR framework has been designed in a way that a potential AR application developer can easily disable or enable parts of it either temporarily or permanently depending on the specific needs of the AR application being developed. Not all AR applications use the same set of hardware components and as a result, parts of this framework may not be applicable to some applications. However, the extensibility feature of the framework provides an AR application developer with a highly flexible architecture in which plugging or unplugging

each hardware component does not affect the overall integrity of the framework. Since all mobile AR platforms must fulfill similar operational requirements, the design methodology outlined here can be easily modified and extended to other emerging areas of AR research.

A mobile outdoor AR platform requires a certain minimum set of hardware components and the ability to programmatically interact with them. An important consideration in the selection of peripheral components for an AR platform is compatibility of data transfer protocols as well as operational control. In order to address this issue, a software library was also developed in this research which is capable of communicating with the peripheral tracking components. A user interface built based on the developed software library was designed to accommodate several hardware component models in which the input data format follows open, industry adopted standards.

Hence, the primary contributions of the research presented in this Chapter are the designed mobile hardware apparatus, and software methods in the AR framework that implement the developed georeferenced registration algorithms to accurately superimpose virtual CAD objects into an AR animation. The hardware apparatus and software methods have been designed to be conveniently extended and reused by other AR application developers simply by importing the provided class libraries into their code, thereby saving significant time and effort that would otherwise be required to reimplement low level hardware interfacing software.

3.4. Hardware Components Selection

As shown in Figure 3.4, all mobile computing systems used in AR research are generally equipped with the same major components:

- A Head Mounted Display (HMD),
- User registration and tracking peripherals, and
- A mobile computer typically a laptop, to control and facilitate system operation.

The following Subsections will discuss the necessary tasks a mobile outdoor AR system should be capable of doing together with the need for and the rationale in selecting each component, and their role in fulfilling the critical requirements for successful mobile outdoor AR operation. They will also provide analyses of various products as well as important technical specifications considered in the selection process. Figure 3.5 provides a detailed illustration of the latest hardware components of UM-AR-GPS-ROVER [9].

Figure 3.5 – Hardware Components of UM-AR-GPS-ROVER

3.4.1. Registration Devices

As discussed in Section 3.2, the problem of accurate user registration requires two distinct sets of data to be measured continuously during the simulation. The first critical record that must be maintained is the user's global position. This measurement is taken in

the form of longitude, latitude, and altitude, which can be extracted using standard GPS equipment. The second important set of data is that of the user's head orientation which is usually recorded in the form of roll, pitch, and yaw angles and can be measured with a 3DOF orientation tracker. These six measurements together are used to calculate the relative distance and heading of each virtual CAD object with respect to the user, given its predetermined coordinates, and render it accurately in the final augmented view [1,10].

Several factors were considered important in the selection of a GPS receiver for this research. Different GPS receiver and antenna models are compatible with certain GPS services such as Wide Area Augmentation System (WAAS), OmniSTAR XP, and HP. This, in turn, provides varying levels of measurement accuracy. Since an accurate augmented image is necessary and slight errors in virtual object position are noticeable to the user, a very accurate service was desired. However, considering the fact that different AR applications may demand different levels of accuracy, a generic data communication interface was designed based on a standard data transmission protocol that could be used to acquire data from different GPS models. As a result, GPS data acquisition methods provided in this research are all based on the National Marine Electronics Association (NMEA) protocol, a widely accepted standard for GPS data communication [11].

Initially and for prototyping purposes, a Garmin ETrex GPS receiver was selected to obtain user's global position in the AR platform. This was followed by a DeLorme Earthmate LT-20 receiver. These two devices mainly served as convenient means for the proof-of-concept phase in which the main concern was to assure that the platform can function properly at an acceptable integration level between all the peripheral devices. However, for final validation stage, and to take into account the accuracy requirements and equip the AR platform with a highly reliable positioning tool, a Trimble AgGPS 332 Receiver with a Combo L1/L2 Antenna was ultimately selected. Table 3.1 summarizes the important technical specifications of the three GPS devices used in the development of the presented AR platform.

The GPS receiver transmits several data fields to the laptop computer over a standard RS-232 interface using NMEA format. Under NMEA, incoming GPS data follows certain sentence formats with each sentence containing several information fields such as positioning, accuracy level, and number of satellites observed. For the purpose of this research, incoming sentences starting with $GPGGA were extracted to obtain longitude, latitude, and altitude data of the user (Subsection 3.5.1). The L1/L2 antenna allows for Differential GPS (DGPS) accuracy within approximately 10-20 centimeters of actual position, when using the OmniSTAR XP or HP subscriptions. Real Time Kinematics (RTK) technique can also be integrated into the framework which allows for accuracies up to 1 centimeter. The GPS antenna was mounted on a segment of pipe, which was secured to the interior of the backpack to prevent lateral movement. To take into account the small difference between the actual position of the user's head and the one for the GPS antenna, a one-time calibration is done as soon as the AR platform starts receiving data.

Table 3.1 – Comparison of GPS Receiver Models Used in This Research

Model	GPS Accuracy	External Interface	Data Format	Dimensions (inch)
Garmin eTrex	<3 m (WAAS)	RS-232	NMEA 0183	4.4 x 2.0 x 1.2
DeLorme Earthmate LT-20	<3 m (WAAS)	USB	NMEA 0183	1.9 x 2.6 x 0.6
Trimble AgGPS332	<10-20 cm (OmniStar XP)	RS-232	NMEA 0183	5.2 x 2.2 x 8.6

Just as with the GPS receiver, the orientation accuracy and resolution provided by various head tracker models were critical factors. In order to limit restrictions imposed on the user's motion, it was important to maximize the range in which accurate measurements could be obtained. For prototyping purposes, an Intersense InterTrax 2 orientation tracker was used. Although the quality of orientation data obtained from this tracker was quite satisfactory, it lacked an internal compass. Without this feature, it would be necessary to calibrate the device to due north at the start of each session. Such manual calibration

could introduce potential accumulating error that would affect all subsequent measurements during operations and diminish the accuracy of the final augmented images. In addition, when using inertial trackers, rapid motion or long tracking periods lead to drift error accumulation. It was thus decided that the orientation tracker should include a built-in compass which led to the selection of a TCM5 magnetic orientation tracker. Table 3.2 shows a comparison between the 3DOF orientation tracking modules.

The TCM5 orientation tracker employs solid-state magnetic field sensors which measure compass heading through a full 360 degree of rotation. The tracking device was placed at the highest point inside the user's helmet, directly above the top of the head, and parallel to the forward line of sight. This module also uses an RS-232 protocol to transmit data at speeds which facilitate real time position and orientation calculation.

Table 3.2 – Comparison of 3DOF Head Tracking Models Used in This Research

Model	Angular Resolution	Angular Range			Built-in Compass	Dimensions (mm)
		Pitch	Roll	Yaw		
Intersense InterTrax 2	0.02^{o}	±80º	±90º	±180º	No	94 x 27 x 27
PNI TCM5	$< 0.01^{o}$	±90º	±90º	±180º	Yes	35 x 43 x 13

3.4.2. Input/Output Devices

The first required interface that must be implemented exists between the computing system and external environment. There must be a method of obtaining continuous visual images of the user's environment, which are ultimately integrated with the virtual images to produce the final, augmented view. Since the designed AR platform was mainly intended to be used for outdoor applications where simulated operations are visualized in a wide range operational environment such as a construction jobsite, a monoscopic visual system was adopted in this research. The selection of a monoscopic visual design enables the AR platform to display animated operations occurring in a wider range compared to a stereoscopic visual design which is more suitable for short range applications under

controlled conditions. Furthermore, in a stereoscopic visual design, different images must be generated for each eye which reduces the effective performance of the image generation system [12].

While a camcorder was used in the initial stages of the research [13], it was decided that a digital video camera could be effectively incorporated into the platform. The two most critical factors in the selection of a video camera were the resolution of the video images, and the speed with which the images are transmitted to the video compositor engine, so that they can be augmented and displayed to the user in real time.

A Fire-i Digital Firewire camera was mounted on the front side of the HMD, directly in line with the user's eyes. Firewire protocol was chosen for its relatively higher speed in transmitting captured images, in comparison with a serial or Universal Serial Bus (USB) connection. It captures video of the user's surroundings at a 640x480 resolution, transmitting to the laptop at approximately 400 Mbps. This camera captures scenes from the landscape within a 42 degree horizontal and a 32 degree vertical view angle.

In order to meet the goal of real time response to the changes in the environment, the augmented view must be made available to the user as soon as the video compositor engine operating on the laptop integrates the virtual images into the video of the landscape. Thus, the final component necessary for the AR platform interface is an HMD. HMD devices are mainly divided into two categories: Video See-Through and Optical See-Through. An optical see-through HMD is transparent, allowing the user to see their surroundings firsthand, with only the virtual images being displayed by the device. A video see-through HMD, however, is opaque, preventing the wearer from actually viewing their surroundings. Instead, they are able to see only what is being captured as input by the video camera. The fact that in a video see-through HMD, the captured views from the real environment and the virtual CAD objects can be processed and combined by the computer facilitates the analysis and correct alignment and coordination of the final composite visualized scene with respect to time [14]. Hence, the i-Glasses SVGA Pro video see-through HMD was selected for the AR platform in this research. This

HMD connects to the laptop via a standard 15-pin VGA port, providing a 800x600 video resolution on two separate viewing screens.

3.4.3. Interaction Devices

The mobile computing backpack, while being self-contained, should be capable of continuously obtaining both instructions from the user and information from the outdoor environment. During regular site operation, it may often be necessary for the user to change computer or component settings, or to select a new mode of operation after one task is complete. It is desirable to allow the user to do this without having to temporarily halt operation and remove the backpack to physically access the laptop. In selecting components to accommodate user input, the decision was made to include both a keyboard and a touchpad to provide full interactivity with the laptop, even though the user will not be physically accessing the laptop. The primary factors in the evaluation of these components were:

- Ergonomics, including size, layout, and potential placement, and
- Accessibility to the user while operating in the field.

The components also needed to have an intuitive interface, so that configuration and operation settings can be adjusted easily without impeding operational efficiency. The WristPC wearable keyboard and the Cirque Smart Cat touchpad were finally selected. These two devices were both integrated into the mobile AR platform through a USB port, and together provided the necessary interface functionality to the laptop. While the touchpad could be clipped to the shoulder strap of the backpack, the miniature keyboard was worn on the user's forearm with both components placed opposite the user's dexterous hand. The necessary visual interface was then overlaid onto the augmented landscape being displayed to the user on the HMD screen.

3.4.4. External Power Source

The fully integrated mobile AR platform must also be capable of uninterrupted operation for a reasonable period of time. While individual component batteries provided adequate power, each device had a different estimated battery life, and all parts must be operational for the platform to operate. It would be very inefficient for the user to halt their operation in order to replace or recharge individual components. To take this into account, an external power source had to be included to provide additional operation time for both the laptop computer and peripheral components. However, the process of selecting an external power source should balance a gain in operation time with the added physical burden that must be borne by the user, as general purpose batteries typically increase in both weight and volume as the available power increases.

One reasonable method to provide additional power was the use of external, component-specific batteries, usually sold as an optional accessory by the manufacturer. The i-Glasses SVGA Pro HMD offers a 6 oz. external Lithium Polymer battery and charger, which can be used exclusively by the HMD unit. When fully charged, this battery will power the HMD for up to 4 hours. Also, the TCM5 orientation tracker provides a case for 3 AAA batteries, which can be stored in the backpack and plugged into the module's connection cable.

After studying the individual components and selecting a target time for continuous operation, the individual power requirements and operating times for each of the devices were used to estimate the necessary capability of an external power supply. Table 3.3 displays the components that need an external power source to assure desired operation time.

Table 3.3 – Final Power Schedule of UM-AR-GPS-ROVER

Component	Requires External Power	Power (W)
Head-Mounted Display	Yes	14.000 - 15.000
Orientation Tracker	AAA batteries	0.7000 - 01.000
Firewire Digital Camera	Yes	0.9000 – 01.000
GPS Receiver	Yes	03.500
Keyboard	No (through USB)	00.035
Touchpad	No (through USB)	02.500
Laptop Computer	Yes	68.000

Several external power packs with special concern given to physical dimensions and power supplying capability were considered. For supplying external power to the final prototype, the Powerbase NiMh High-Power External Portable Power Pack was selected. This battery has slightly smaller dimensions than that of a typical laptop, allowing it to be easily stored in the backpack, adjacent to the laptop. This 12V power pack is rated at 7.5 Amp-Hours and is able to provide approximately 4-6 hours of additional laptop operation. In addition, a cigarette lighter outlet was found to be convenient since it is a Plug and Play (P&P) device that can also be used in a car. It also simplifies the process of splitting the single outlet into multiple outlets to accommodate several components. Since the power pack offers only one cigarette lighter outlet, a splitting adaptor was added to provide a total of three outlets. Two of these three ports provide dedicated power to the Trimble GPS Receiver and the Fire-i Digital Firewire Camera, with a slot remaining free for the laptop computer, when necessary.

3.4.5. Mobile Backpack

The final issue for the AR mobile computing prototype designed in this research was the selection of a frame and backpack apparatus. This device is worn by the user, with the laptop and hardware components stored securely within. Ergonomic considerations played a significant role in the selection of an initial backpack model, since user comfort is a critical factor in the design of any outdoor mobile AR platform. Compact design,

component security, and proper weight distribution were all analyzed during this selection process. While the creation of a customized configuration was considered, it was determined that a Commercial-off-the-Shelf (COTS) component would be used for the initial design to demonstrate mobile operation, with the intent to revisit this option in the future.

The Kensington Contour Laptop Backpack was ultimately chosen as the first generation backpack for the mobile computing platform. This particular model provides adequate internal space for the GPS receiver, laptop computer, power pack, and additional cords traveling between the AR components. The backpack design also provides easy access to the components being carried in the main compartments, allowing hardware connections to be quickly checked when necessary. This product is endorsed by the American Chiropractic Association (ACA) and equipped with an adjustable lumbar support near the lower back, allowing weight to be distributed down to the hips instead of the shoulders or back.

3.5. Software Interface Design

In order to communicate with the input devices throughout a core platform and provide appropriate augmented output to the user, an OOD approach was adopted. The software interface design process in this research followed a cyclic pattern shown in Figure 3.6. As shown in this Figure, the process started with module data structure design during which the structure of each individual communication module (i.e. C++ class) was designed. The output of this process were algorithms and pseudo code to perform specific tasks such as creating a communication handle to a hardware device, and extracting data via that handle. Based on the output of this step, the actual C++ implementation of each module was created. The developed C++ modules were then individually validated to assure proper functionality and data acquisition from the registration devices. Once the modules were validated, they were integrated into a prototype application. The prototype application was then run and the results were validated again. The overall performance of the system always served as an effective feedback to continuously improve the initial

design and optimize the functionality of individual modules. The dashed arrow in Figure 3.6 represents possible refinements to the original module designs.

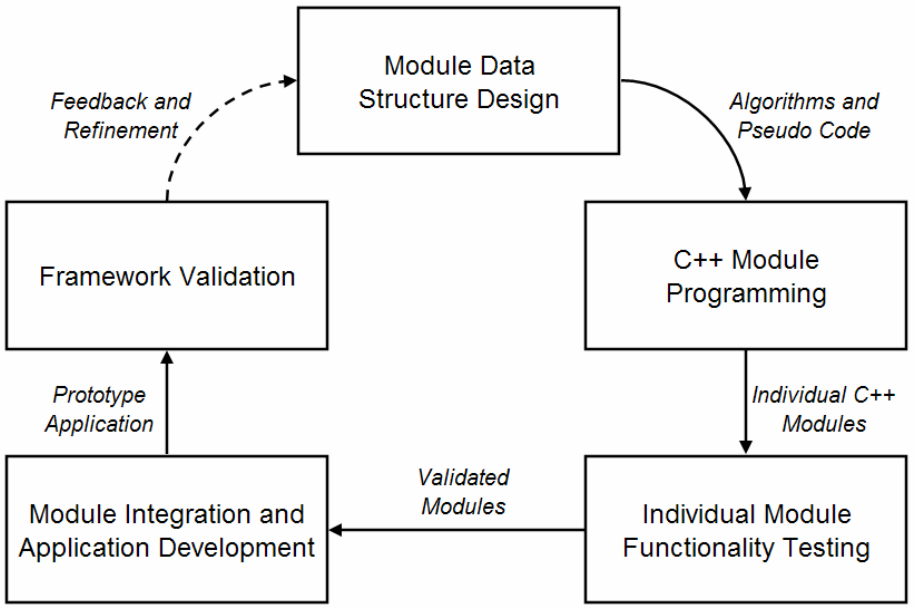


Figure 3.6 – Steps in Software Interface Design

To make the code as reusable as possible, the output of the implementation stage for each individual module was exported as a Dynamic Link Library (DLL) file. Having done so, the means and methods of each module while being organized in a class structure, can be later imported by potential AR users to be applied to their specific application. As described earlier, the two most important issues that have been taken into account in order to allow for such a modular design were extensibility and reusability. A very distinctive feature of the current software architecture which highlights the importance of reusability is that it uses American Standard Code for Information Interchange (ASCII) as well as binary data acquisition mechanisms to communicate with the positioning and orientation devices through serial port connections. The main advantage of having such a relatively low level communication interface is that the established methods are generic and can easily be reused and reapplied to other registration devices provided that they output data following a standard format.

Figure 3.7 shows the current class structure of the designed AR platform using OOD terminology [15]. In this figure, a solid line with a black circle at one end connecting two classes represents an “aggregation” relationship between two classes in which one class has an instance of the other as a member variable. Likewise, a solid line with a white circle at one end represents a “using” relationship between two classes in which one class uses methods and functions of the other to perform certain operations. All the individual components are pieced together in a wrapper class which provides methods and functions to open ports, initialize the peripheral devices, communicate with them, extract real time data from them, display appropriate augmented graphical output, update the output on a real time basis while the user is moving around freely, and finally close ports at the end of the operation.

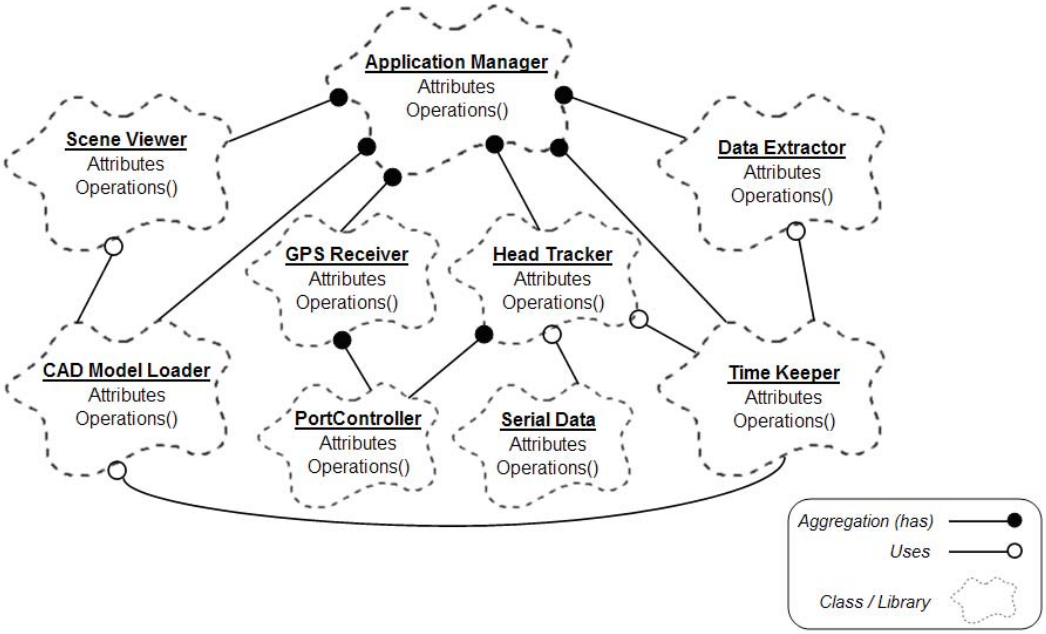


Figure 3.7 – Object Oriented Class Structure of the AR Framework

Figure 3.8 depicts the main loop of the application in terms of events occurring at each frame. As shown in this Figure, at the beginning of each loop run, positional and orientation data are obtained from the registration devices (i.e. GPS receiver and head orientation tracker), checked for validity, extracted, and converted to numerical values. Based on these values, the coordinates of the user’s viewing frustum are updated. The virtual contents of the viewing frustum (i.e. CAD objects) are then refreshed by

recalculating the relative distance between each CAD object and the user, and appropriately placing each CAD object inside the viewing frustum to reflect the effect of the latest changes in user's position and head orientation in the final augmented view. The details of this step are presented in Section 3.6 and Appendix D. The loop continues as long as it is not interrupted by a user exit command. The dotted arrows indicate the internal processes performed at each step of the main loop.

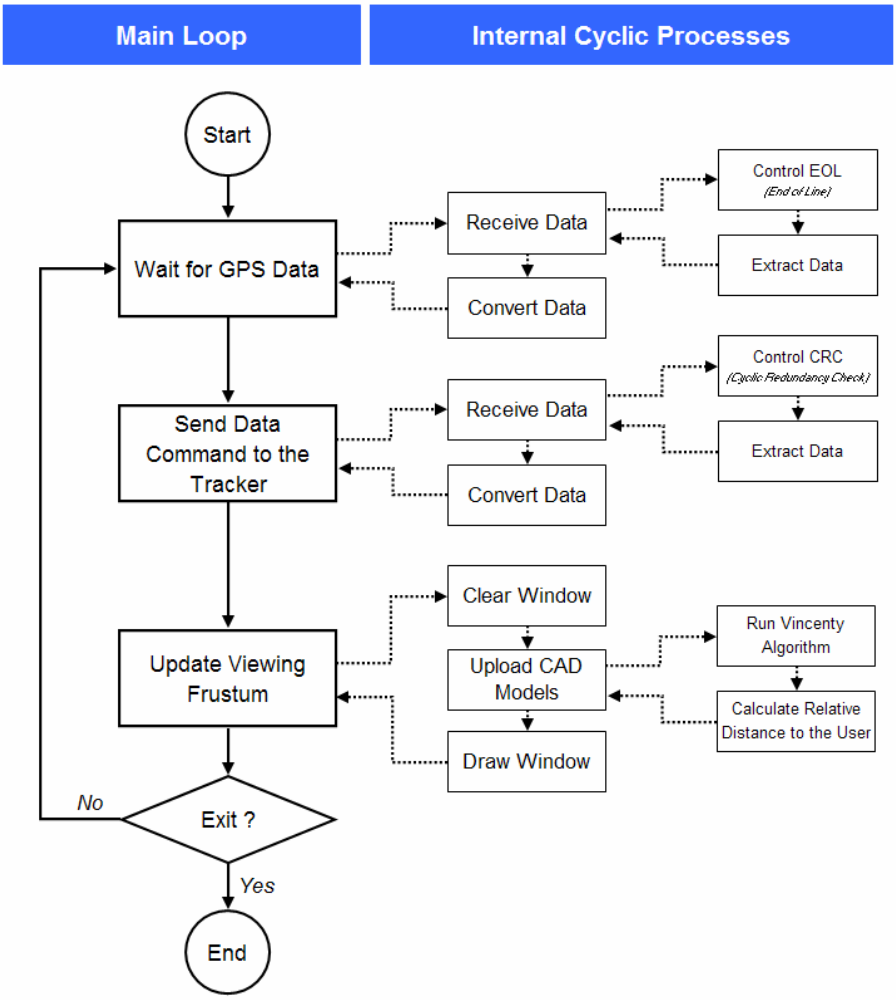


Figure 3.8 – AR Application Main Loop

3.5.1. Positional Tracking

As described in Subsection 3.4.1, the incoming data from the GPS device are in the format of sentences starting with $GPGGA indicator which include the three main

position components (i.e. longitude, latitude, and altitude). After the GPS port is successfully opened, the incoming data sentence is acquired in real time and further broken into parts to extract the positional data. Figure 3.9 shows a sample data stream from a GPS receiver as used in this research.

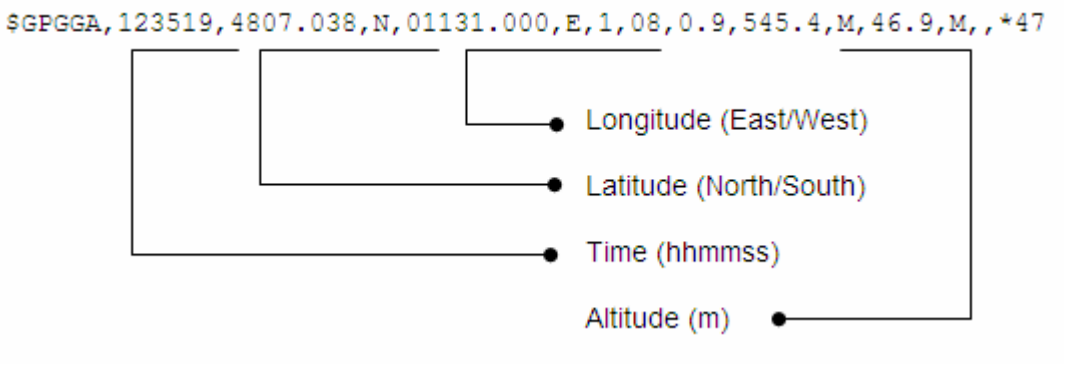


Figure 3.9 – Sample GPS Data Stream Following the NMEA Standard

There were two important steps in communicating with the tracking devices:

- Establishing a communication channel, and
- Obtaining and extracting incoming data.

To establish a communication channel, a serial port communication library called PortController.NET was initially used. This library provides means and methods to open and close a serial port, as well as to receive and send data from and to the hardware device connected to the serial port. The original library was substantially modified in this research to derive two more specific classes used for establishing port communication. These classes were designed and implemented in C++.

Figure 3.10 shows the designed data transmission algorithm for the GPS receiver. Figure 3.11 shows the main functionalities provided within the GPS class. As shown in this Figure, the GPS class uses three main functions to communicate with the GPS device. `UMGPS::Initialize()` opens the serial port to which the GPS device is connected and sets the communication parameters (e.g. baud rate, stop bits, and others). `UMGPS::ReceiveData()` is a function running in the background the main

responsibility of which is to keep track of the events happening in the GPS thread. As soon as it receives an EOL (End of Line) character, it realizes that a complete data line has been received (Figure 3.9). It then starts extracting the numerical values from the data stream. Just before the AR application terminates, it calls `UMGPS::Shutdown()` to end the communication with the GPS device and release the designated serial port [16].

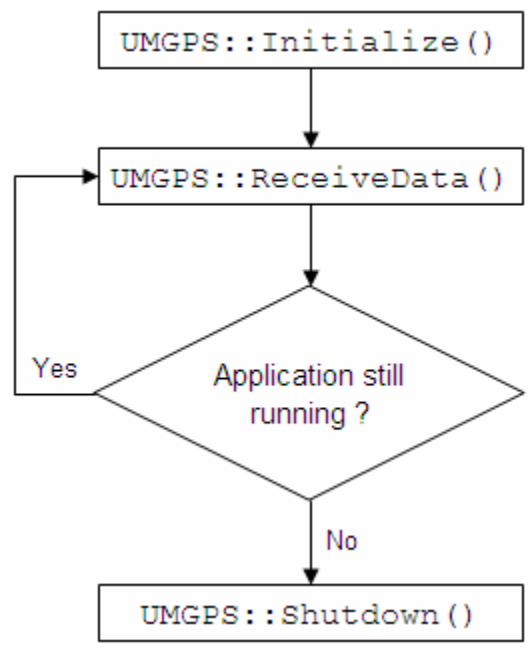


Figure 3.10 – Designed GPS Data Transmission Algorithm

```
__gc class UMGPS
{
      public:

            Initialize();
            Shutdown();
            ReceiveData();
};
```

Figure 3.11 – Designed C++ GPS Communication Class

3.5.2. Head Orientation Tracking

The incoming data from the head orientation tracker is in the format of three angles (i.e. yaw, pitch, and roll). After the head orientation tracker port is successfully opened, the incoming data is acquired in real time and further broken into parts to extract the orientation data. Figure 3.12 shows a sample data stream from the 3DOF head orientation tracking device as used in this research.

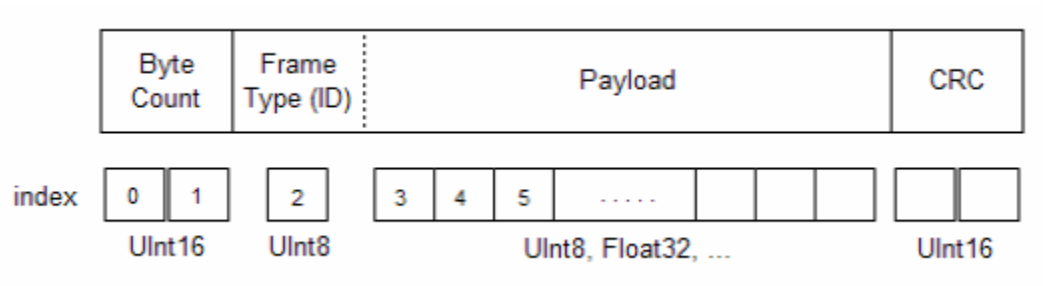


Figure 3.12 – Sample 3DOF Head Orientation Tracking Data Stream

Since the contents of each packet come in a binary format over a serial connection, it is very sensitive to data corruption. As a result, the main content of each packet is followed by a code called Cyclic Redundancy Check (CRC). In general, a CRC is a mathematical transformation applied to a series of bytes that produces an integer result that can be used for error detection. Upon receiving data from the orientation tracker, the AR application calculates the CRC value of the packet using its actual contents and then compares this value to the received CRC. If the two don't match, the conclusion is that the actual data packet has been affected while being transferred and hence is not reliable for use. As a result, the packet is simply disregarded, and the application waits for the next incoming data packet. Otherwise, the data stream is error-free and will be extracted into its components. Again, the numerical values for each of the orientation angles (i.e. yaw, pitch, and roll) are obtained using a set of binary operations applied to the data packet.

In the packet structure shown in Figure 3.12, the main value for each orientation angle is obtained by converting the content of *Payload* to its numerical equivalent. In order to determine which angle this value corresponds to, the *Frame ID* needs to be extracted.

This is because each of the individual orientation angles has their own corresponding *Frame ID*. The orientation tracker communication interface is more involved than the one for GPS. The main reason is that unlike the GPS device that sends out data continuously in the background, the tracker needs to be prompted each time to prepare a data packet and send it to the AR application. Also, since the data comes in a binary format, it needs to be validated before being used inside the AR application. Figure 3.13 shows the designed data transmission algorithm for the 3DOF head orientation tracker.

Figure 3.14 shows the main functionalities provided within the 3DOF head orientation tracker class. As shown in this Figure, to open the serial port to which the tracker is connected and set up the connection parameters, `UMTCM::Initialize()` is called first. `UMTCM::Control()` gives control to the tracker to prepare a binary data packet and send it to the AR application. In order to obtain 3D rotation angles, this function needs to be called three consecutive times with three different *Frame ID* tags. Each time the function is called, it sends out a control command to the tracker which prepares the tracker to send out a single packet containing binary values of yaw, pitch, or roll angles. `UMTCM::ReceiveData()` is then called to receive the binary packet. This function in turns calls `UMTCM::CRC()` to check the validity of the received data by performing a CRC test. As soon as the data packet is marked valid, its content is extracted by `UMTCM::ExtractData()` which also stores the numerical values for the three orientation angles in their corresponding variables for later use inside the AR application [16].

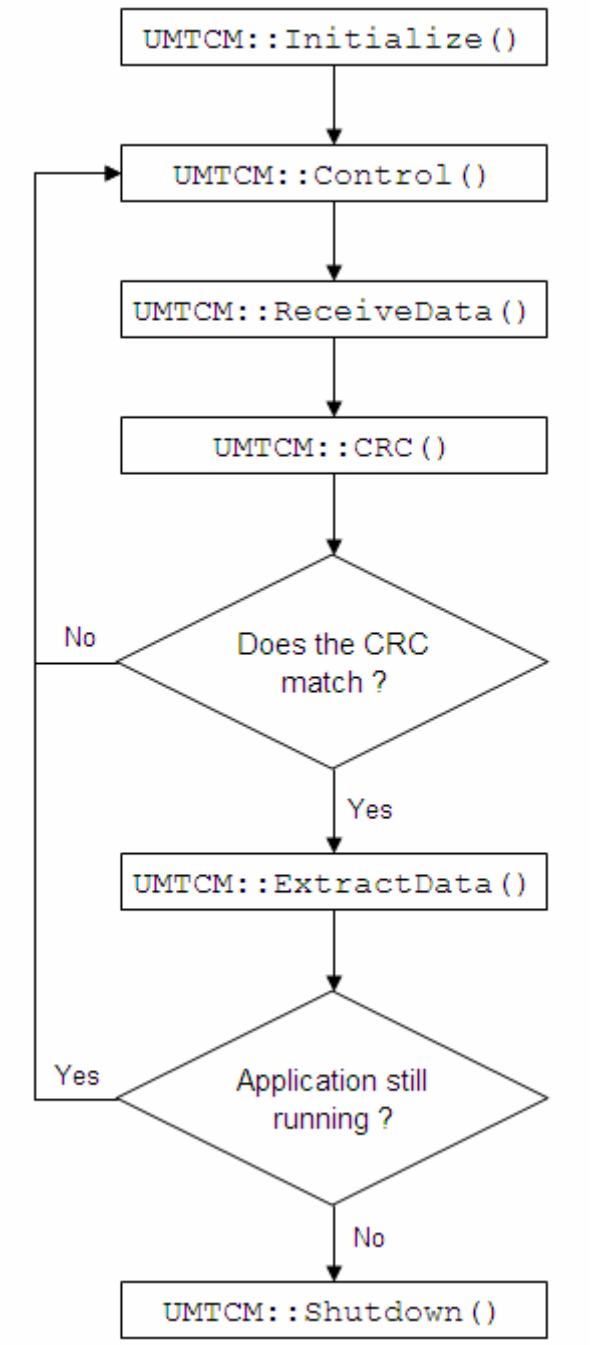


Figure 3.13 – Designed 3DOF Head Orientation Data Transmission Algorithm

```
__gc class UMTCM
{
      public:

            Initialize();
            Shutdown();
            Control();
            ReceiveData();
            CRC();
            ExtractData();
};
```

Figure 3.14 – Designed C++ 3DOF Orientation Tracker Communication Class

3.6. Registration of Computer-Generated Graphics

As the user moves inside the AR animation, the relative distance (R) between the center of the viewpoint (i.e. user's eyes) and each virtual CAD object is calculated using the Vincenty method [17]. This method provides a set of equations to calculate the horizontal distance component between two points with known latitude and longitude, and is accurate to 0.5 millimeters. The important characteristic to note regarding these equations is that the computed distance between two points is at Mean Sea Level (MSL) or more precisely along the "ellipsoid datum".

However, it has been shown that if the points in context are located two kilometers above the MSL, the distance obtained by Vincenty method will be 0.03% smaller than the actual distance. Thus, for the distance computation in this research, the Vincenty method was found to provide sufficient accuracy for obtaining relative distances between a user and augmented virtual models. The Vincenty method used in this research takes advantage of the most accurate and widely used models for the earth ellipsoid (i.e. the closest elliptical shape to the actual surface of the Earth in a certain region) in the North American

continent, the WGS-84/GRS–80. In this model, the earth ellipsoid is defined using the geometric parameters shown in Figure 3.15.

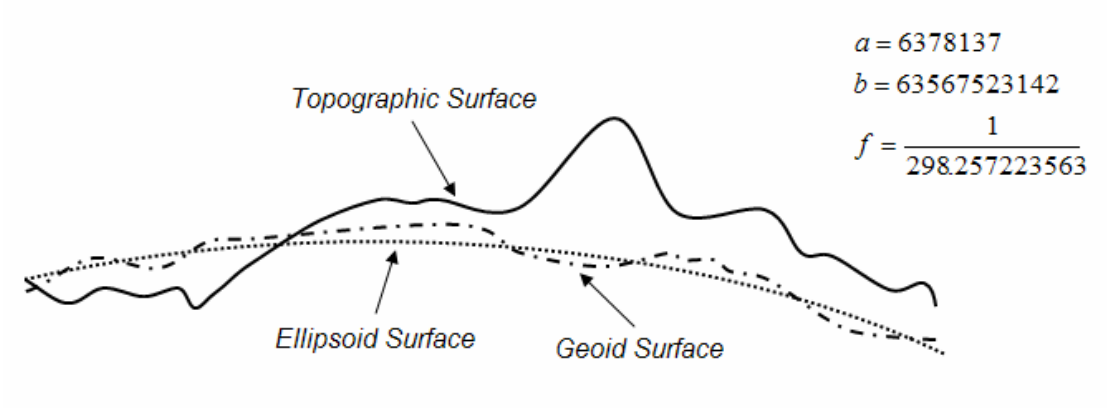


Figure 3.15 – Basic Geometrical Parameters of the Earth Surface in WGS84

In this Figure, *a* and *b* are the major and minor semi axes of the ellipsoid and *f* is the flattening ratio which is the difference in the length of the two semi axes (i.e. *a*-*b*) divided by the length of the major semi axis (i.e. *a*). The primary variables used in this method are presented in Figure 3.16 in the form of pseudo code. All angles are measured in radians. In addition, the "East" and "North" directions are assumed positive while the "West" and "South" are considered to be negative in sign.

$$
\begin{aligned}
&\textit{// Primary variables and constants}\\
&\Delta Long = Longitude_2 - Longitude_1\\
&U_1 = tg^{-1}\left[(1-f)tg(Latitude_1)\right]\\
&U_2 = tg^{-1}\left[(1-f)tg(Latitude_2)\right]\\
&SinU_1 = Sin(U_1)\\
&SinU_2 = Sin(U_2)\\
&CosU_1 = Cos(U_1)\\
&CosU_2 = Cos(U_2)\\
&Lambda = \Delta Long\\
&LambdaP = 2\pi\\
&Count = 0
\end{aligned}
$$

Figure 3.16 – Primary Geometrical Parameters Used in Vincenty Method

The Vincenty method follows an iterative procedure to calculate the distance between any two given points until it reaches an acceptable level of convergence in which the difference between the last two calculated values of *Lambda* and *Lambdap* is less than a specific amount, or until a certain number of iterations have been performed, whatever comes first. Figure 3.17 presents the pseudo code for the iterative part of the algorithm. In the pseudo code of Figure 3.17, the error margin has been set at 10^{-12} meters. The maximum number of iterations that the loop should execute in trying to reach the desired accuracy is given by the variable *count*.

// Iterative loop to achieve a certain level of accuracy

$$while\left(\left|Lambda - LambdaP\right| > 10^{-12} \;\&\&\; Count < 100\right)$$

{

$$SinLambda = Sin(Lambda)$$

$$CosLambda = Cos(Lambda)$$

$$SinSigma = \sqrt{(CosU_2 \times SinLambda)^2 + (CosU_1 \times SinU_2 - SinU_1 \times CosU_2 \times CosLambda)^2}$$

$$CosSigma = SinU_1 \times SinU_2 + CosU_1 \times CosU_2 \times CosLambda$$

$$Sigma = tg^{-1}\left(CosSigma \,/\, SinSigma\right)$$

$$Alpha = Sin^{-1}\left(CosU_1 \times CosU_2 \times SinLambda \,/\, SinSigma\right)$$

$$CosSqAlpha = Cos^2(Alpha)$$

$$Cos2SigmaM = CosSigma - 2 \times \left(SinU_1 \times SinU_2 \,/\, CosSqAlpha\right)$$

$$C = 1/16 \times CosSqAlpha \times \left(4 + f \times (4 - 3 \times CosSqAlpha)\right)$$

$$LambdaP = Lambda$$

$$Dummy = Cos2SigmaM + C \times CosSigma \times (-1 + 2 \times Cos2SigmaM^2)$$

$$Lambda = \Delta Long + (1 - C) \times f \times Sin(Alpha) \times (Sigma + C \times SinSigma \times Dummy)$$

$$Count = Count + 1$$

}

Figure 3.17 – Iterative Computation Algorithm Used in Vincenty Method

The final non-iterative segment of the algorithm computes the distance S (in meters) between the two points with known latitude and longitude values. The pseudo code for this segment of the algorithm is shown in Figure 3.18.

// Final computations to obtain the relative distance between two given points

$$uSq = CosSqAlpha \times \left(a^2 - b^2 \big/ b^2\right)$$

$$A = 1 + uSq \big/ 16384 \times \left(4096 + uSq \times \left(-768 + uSq \times (320 - 175 \times uSq)\right)\right)$$

$$B = uSq \big/ 1024 \times \left(256 + uSq \times \left(-128 + uSq \times (74 - 47 \times uSq)\right)\right)$$

$$Temp1 = B \big/ 6 \times Cos2SigmaM \times (-3 + 4 \times SinSigma^2) \times (-3 + 4 \times Cos2SigmaM^2)$$

$$Temp2 = Cos2SigmaM + B \big/ 4 \times \left(CosSigma \times (-1 + 2 \times Cos2SigmaM^2) - Temp1\right)$$

$$DeltaSigma = B \times SinSigma \times Temp2$$

$$S = b \times A \times (Sigma - DeltaSigma)$$

Figure 3.18 – Final Distance Computation Using Vincenty Method

3.6.1. Horizontal Distance Calculations

Figure 3.19 depicts the initial approach used in this research to register virtual CAD models in a user's view and superimpose them in the final augmented scene. In order to perform the registration task, the GPS receiver obtains the user's global position in real time. Knowing the same pieces of data for a virtual CAD object (either from a text file created manually or from the simulation scenario), the relative distance between the user and the virtual object (R) is calculated using the Vincenty method. As soon as the distance between the user and the virtual object is determined, the corresponding relative heading angle (based on the geographical North direction being zero) of the vector connecting the two points (UO) is calculated. This is depicted by the symbol α in this Figure.

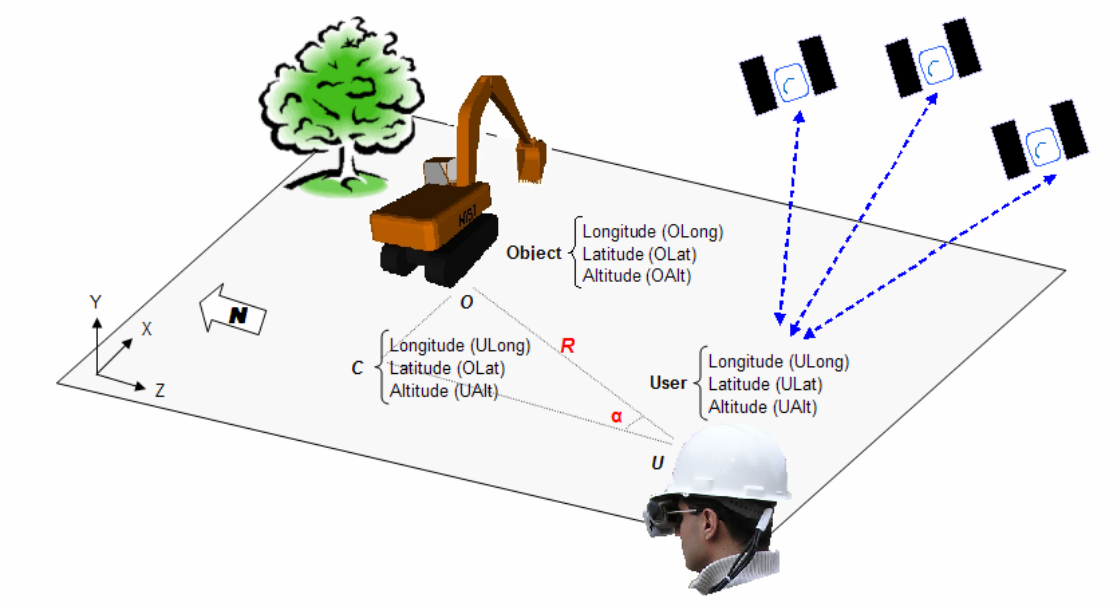


Figure 3.19 – Initial Approach for Accurate Registration of Virtual CAD Objects

To find α, an imaginary point *C* is added to the calculations. This point has the same longitude as the user's longitude and its latitude is the same as the latitude of the object. Together with points *U* and *O*, these three points create a right triangle (Note that *UC* always points to the geographical North direction). The additional side lengths *CO* and *UC* of this triangle can be calculated by applying the Vincenty method on the corresponding vertices. Using trigonometric equations, the heading angle (α) is then calculated as $tg^{-1}(CO/UC)$. By the end of this set of calculations, two important pieces of information about the objects in the user's view are available: the relative distance, and the relative angle.

At this stage, transformation matrices can be effectively used to manipulate registered CAD objects as required (Chapter 4). In order to do that, each object is first subjected to a translation matrix by which it is translated into the depth of view by an amount equal to the distance between the two points (*R*). Then a rotation matrix is applied to the object by which it is rotated about the Y axis (vertical axis) by an amount equal to the calculated relative angle (α). These two transformations update the virtual object's location based on

the last user movement. Figure 3.20 is a schematic illustration of the transformation procedure applied to each virtual object inside the user's field of view.

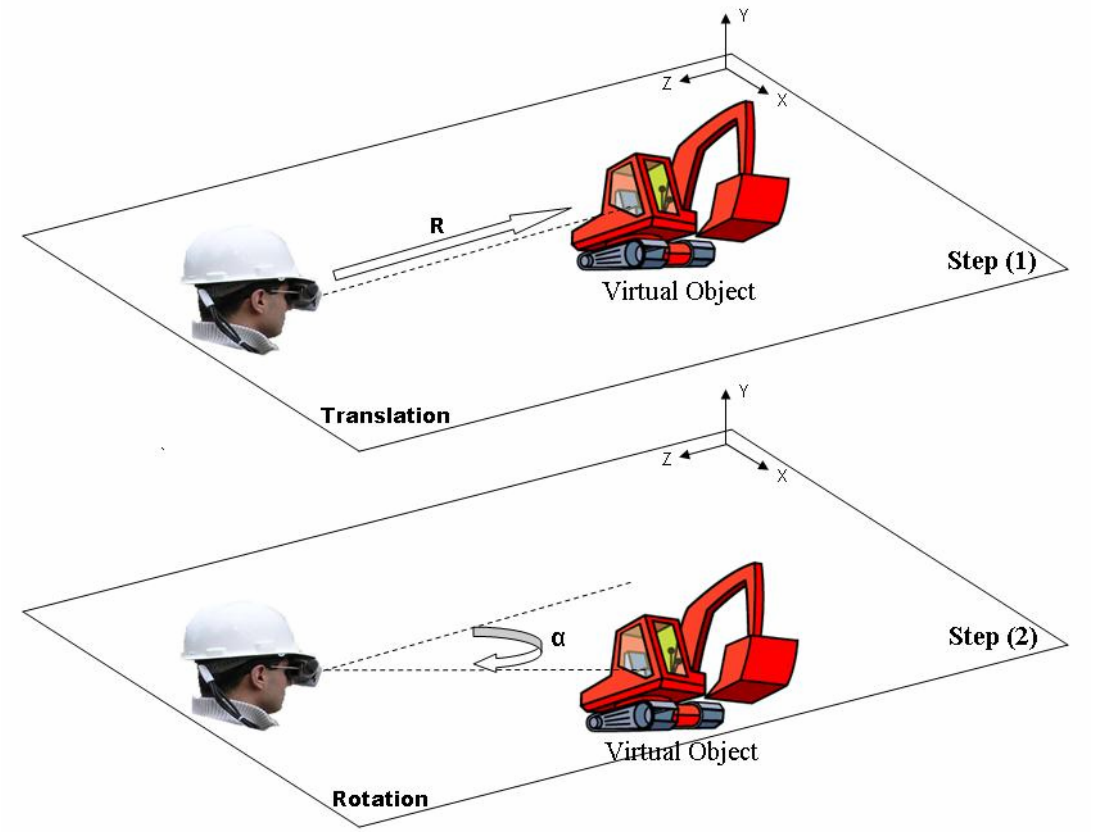


Figure 3.20 – Transformation of a CAD Object in the Field of View

3.6.2. Elevation Difference Calculations

Revisiting Figure 3.19, the basic assumption in calculating the horizontal displacement (R) and angle (α) is that both the user and the CAD object have the same elevation (i.e. global altitude). Thus, in case the object is located at a higher or lower elevation than the user, further adjustments are needed requiring additional computation steps. In Figure 3.21, for example, the object has a lower altitude than the user. In this case, the relative pitch angle between the user and the object (β) must be calculated using properties of triangle *USO*.

The virtual object is then rotated about the X axis by an amount equal to the calculated angle β. As shown in Figure 3.21, the object is moved along the *SO'* curve as a result of this rotation. For simplicity, a good assumption can be translating the object along the cord *SO* in which case the final position of the virtual model ends up to be the point *O* instead of *O'*. A final adjustment may be needed to the object's initial side roll angle to represent the possible ground slope (equal to β in Figure 3.21). In this study, a pure rotation equal to the slope is being applied to the object about its local X axis so that it lies completely on the ground. In other words, the ground plane is assumed to be uniform and parallel to line segment *UO'*.

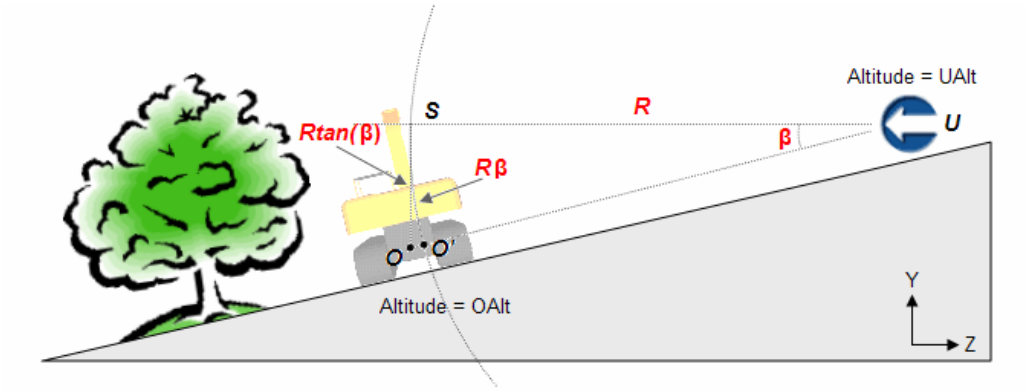


Figure 3.21 – Calculation of Elevation Difference between User and a CAD Object

If a translation along *SO* is preferred as opposed to a rotation along *SO'* a minute positional error may occur which can be safely neglected for the purposes of this study without experiencing any adverse visual artifacts. However, for wider range applications (e.g. objects that are to be placed far away from the user), the length of *OO'* can be large in which case, instead of translating the object from *S* to *O*, a rotation equal to β must be applied to transform the object from *S* to *O'*.

3.6.3. Modified Transformation Method (MTM)

The initial transformation calculation method described in Subsections 3.6.1 and 3.6.2 provided accurate and acceptable results in terms of translational and rotational values.

However, the implementation of this method to place a virtual CAD object inside the user's field of view requires some additional steps to be executed. For example, once a rotation (about X, Y, or Z axes) is applied to a CAD object to place it in the correct position, the local orientation of the object also changes equal to the amount of the applied rotation. To correct this, an additional rotation matrix has to be locally applied to the object. Figure 3.22 shows how this local orientation change occurs when applying a global rotation to a virtual CAD object.

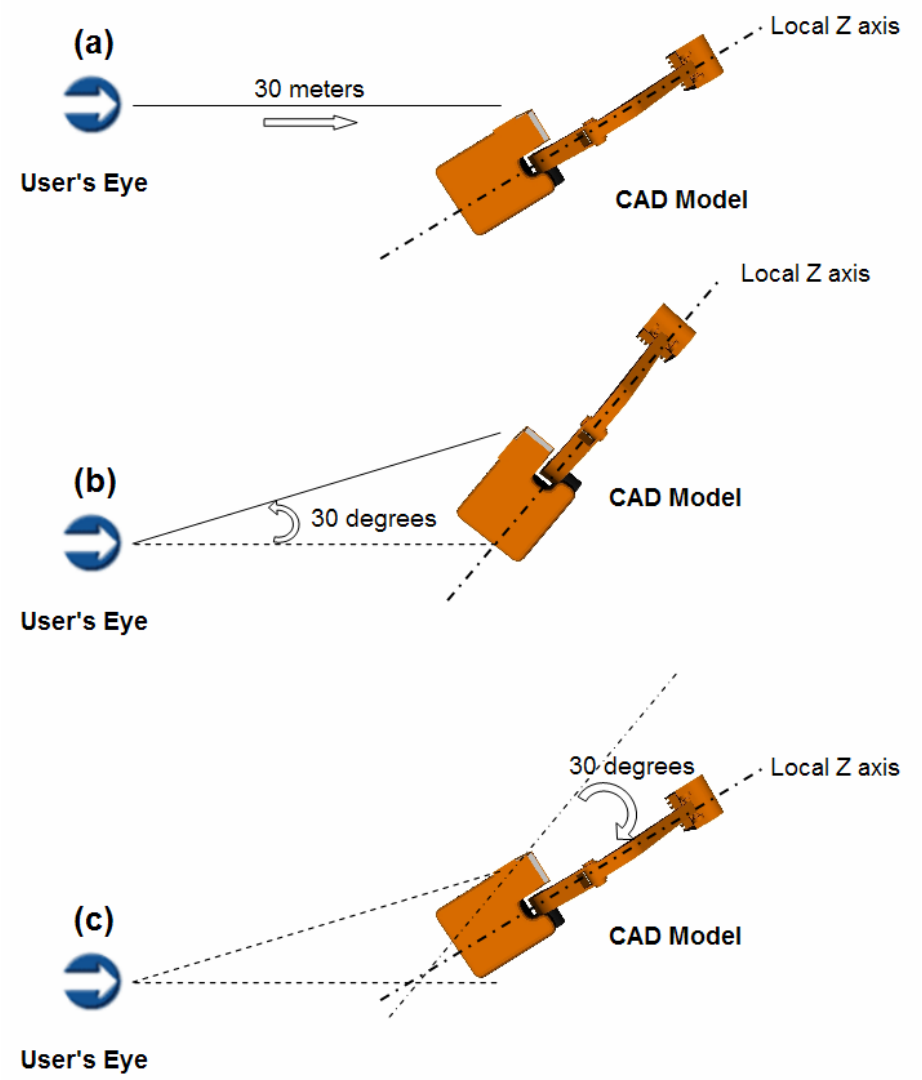


Figure 3.22 – Steps to Adjust the Local Orientation of a Virtual CAD Object

This Figure shows the top view of an augmented scene in which the virtual CAD model of an excavator is to be placed 30 meters from the user with a corresponding heading angle of 30 degrees. Following the procedure outlined in Subsection 3.6.1, the CAD model is first translated 30 meters away from the user's eye (Figure 3.22.a). A rotation about Y axis will then be applied to the CAD model to take into account the relative heading angle (Figure 3.22.b). At this stage, although the CAD model is located at the correct position, its local orientation has been changed by 30 degrees (equal to the rotation value) which needs to be further adjusted. This can be done by locally applying a -30 degree rotation to the CAD object (Figure 3.22.c). As a result, to place a CAD object at a point located on the same plane as the user's eye is located (i.e. plane XZ), at least three different transformations are needed. In case there is a difference of elevation, one more step has to be executed in order to translate the CAD model along the Y axis.

For a limited number of CAD objects, this process is found to be reasonable. However, as the number of CAD objects increases and considering the fact that the positional errors accumulated after all these steps can exceed the acceptable level of accuracy, a higher precision method was needed. This was achieved in this research by developing the MTM. The MTM, when applied on two points with known global positions, calculates their relative position along the three major axes in meters [1]. These values are then used inside a single translation matrix to position the CAD object correctly in the scene. As shown in Figure 3.23, the MTM yields a different set of output values compared to the original Vincenty method described in Subsections 3.6.1 and 3.6.2 as it calculates distance components between two points along the three major X, Y, and Z axes rather than the length of the line segment directly connecting the two points together.

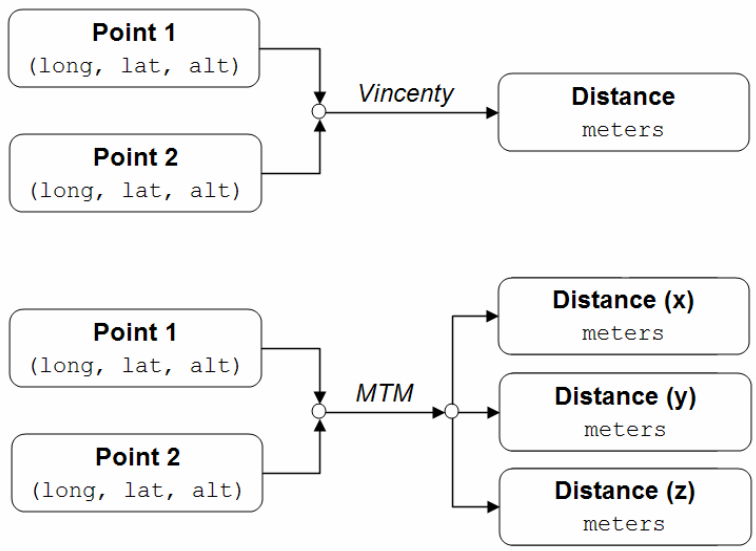


Figure 3.23 – Comparison between the Original Vincenty Method and the MTM

Using the MTM, the relative position components along the X and Z axes (*CO* and *CU* in Figure 3.19) together with the elevation difference along the Y axis (*SO* in Figure 3.21) are used to create a single translation for the virtual CAD object. This way, the rotational values between the user and the object don't change and as a direct result, local orientation of the CAD object remains unaffected. Figure 3.24 shows how the application of the MTM changes the transformation procedure initially introduced in Figure 3.20.

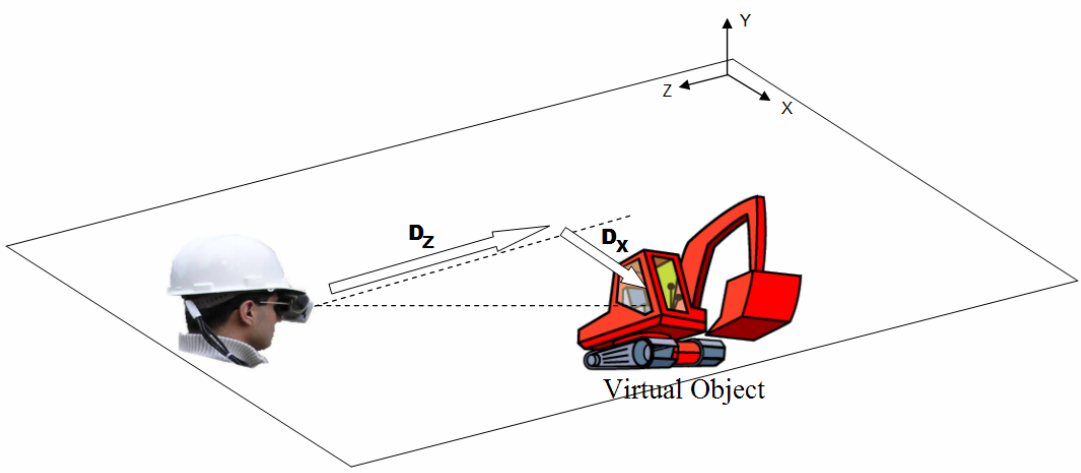


Figure 3.24 – The MTM Procedure to Place a Virtual CAD Object in the Field of View

In this Figure, a virtual CAD model undergoes a single step translation procedure (as opposed to a translation followed by a rotation shown in Figure 3.20) in order to be placed on the appropriate location relative to the user's position. Note that in this Figure, the virtual object and the user are assumed to be on the same plane and as a result, no elevation difference calculation is necessary.

When the AR user turns the head, the relative change in the head orientation is obtained using the three pieces of data coming from the 3DOF head orientation tracker. Applying the reverse rotation to the virtual objects leads to a final augmented view wherein the objects' locations are unaffected by the user's head movement. In addition, in cases where the user both moves and rotates the head at the same time, all the computation steps in the described procedure are continually repeated so that the final composite AR view remains unaffected as a user moves freely in the environment.

3.7. Development of UM-AR-GPS-ROVER Mobile AR Platform

The designed registration algorithms were implemented in a mobile AR prototype platform named UM-AR-GPS-ROVER that served as a proof-of-concept, and also helped validate the initial research results [1]. The initial design of UM-AR-GPS-ROVER comprised of two main components working in parallel. In addition to the software components of the tool, supporting hardware devices were appropriately integrated to provide the necessary sensing, input, and output capabilities. The connected hardware devices primarily consisted of a GPS receiver, a 3DOF head orientation tracker, an optional HMD, and a laptop computer. Figure 3.25 presents snapshots of the hardware setup used in the initial design of UM-AR-GPS-ROVER. Details of the initial hardware components used in UM-AR-GPS-ROVER are presented in Table 3.4.

Figure 3.25 – Initial Hardware Setup of UM-AR-GPS-ROVER

Table 3.4 – Initial Hardware Components of UM-AR-GPS-ROVER

Functionality	Hardware Component
Positional tracking	Delorme Earthmate GPS device
Head orientation tracking	InterSense InterTrax2 head tracking device
Video capturing	Sony DCR-TRV33 mini-DV digital video camera recorder
Video output	Laptop screen or i-glasses SVGA Pro 3D HMD
User input	Laptop keyboard and touchpad

UM-AR-GPS-ROVER was completely developed in a C++ environment using OpenGL functionalities. In order to communicate with the GPS receiver, a C++ library called GPSToolkit was used which provided methods to open the GPS port, receive data from the GPS unit, and extract them in form of longitude, latitude, and altitude. To establish communication with the 3DOF head orientation tracker, the sample code provided by the device manufacturer was modified and used inside the AR application.

The implemented UM-AR-GPS-ROVER platform was used to place several static and dynamic 3D CAD models at several known locations in outdoor augmented space [18]. In particular, the prototype was successfully tested in many outdoor locations at the University of Michigan north campus using several 3D and 4D construction models (e.g. buildings, structural frames, pieces of equipment, etc.). The initial design of UM-AR-GPS-ROVER supported a variety of CAD objects including those created in the Virtual Reality Modeling Language (VRML) format which provides several features and

functionalities to create a scene description of 3D interactive models. VRML models can be created in any external CAD program (e.g. 3D Studio MAX, or Auto CAD) and exported to other graphical applications [19]. Another useful feature of VRML is that the initial scale and orientation of each of the CAD models created using VRML can be modified when they are loaded in an external application. This feature is widely used in this study so that the dimensions and orientation of the loaded models comply with the global coordinate frame properties of the real environment.

To validate the capability of the UM-AR-GPS-ROVER platform to perform accurate registration, global positions of CAD models were measured on site and input to the system via a text file. The user's position and orientation were also tracked by a GPS receiver and a 3DOF head orientation tracker while moving on the site. The implemented MTM registration algorithm calculated real time relative distance components between the user and the CAD objects and the final view was updated accordingly to produce a realistic augmented view. Figure 3.26 presents two snapshots of structural steel frames registered in outdoor AR.

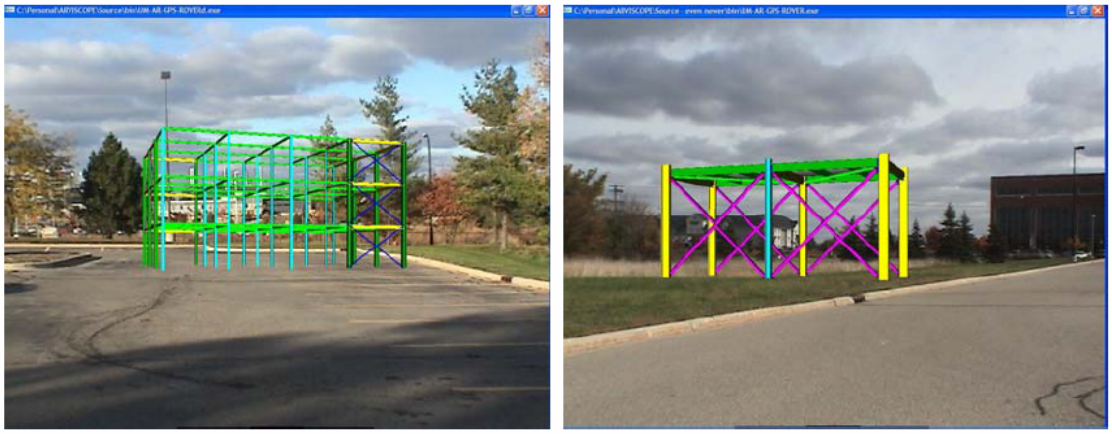


Figure 3.26 – CAD Steel Frames Registered in AR Using UM-AR-GPS-ROVER

In order to test the ability of the prototype to augment dynamically changing graphical models in a user's view, a 4D CAD [20] model of a structural steel frame was registered at a known outdoor location. In particular, scheduled construction activities for the erection of the steel frame (columns, beams, girders, and connection) were graphically

animated with the passage of simulated project time. The simulated project time was introduced as an extra dimension for each augmented virtual model such that an object (e.g. steel beam or column) is not superimposed in the augmented view unless the simulation clock passes its scheduled completion time.

Position data of the virtual CAD models was provided via a text file in which model ID numbers, and their longitude, latitude, and altitude values had been previously stored. In order to allow UM-AR-GPS-ROVER to identify the appropriate appearance time for each CAD object, time tags for all virtual models were also stored in the same text file. The content of a sample text file for an AR scene consisted of six virtual models superimposed on a real background is shown in Figure 3.27.

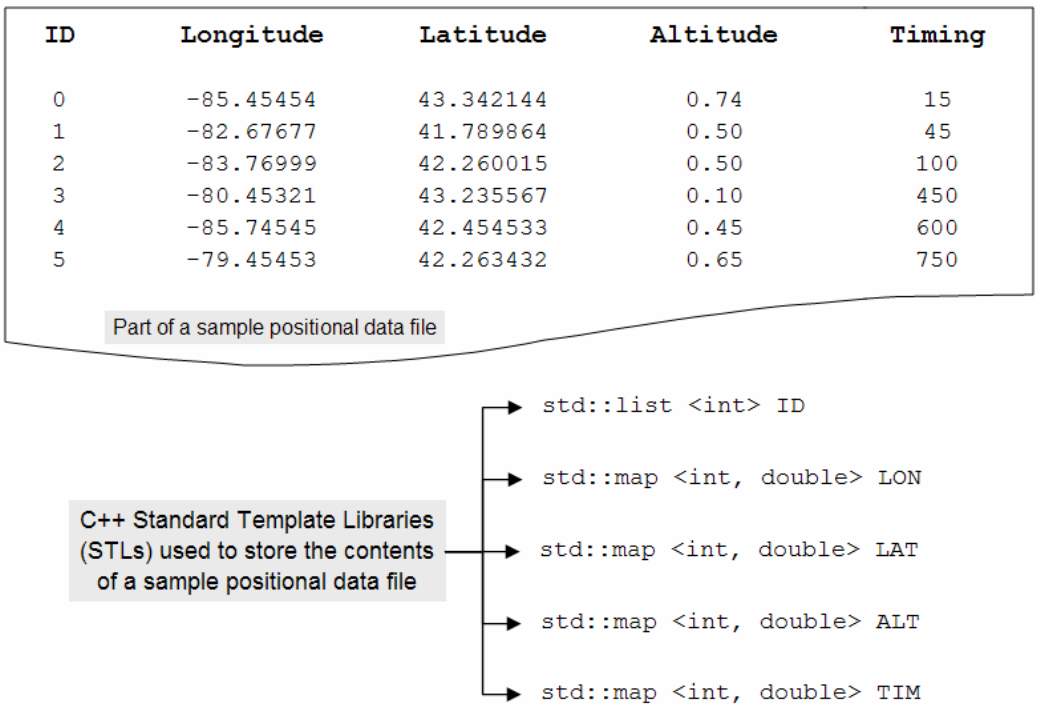


Figure 3.27 – Sample Text File Containing the Properties of Virtual CAD Objects

At each frame in the graphical AR visualization, the text file was searched to determine whether there are object(s) with a time tag smaller than the current simulation time. If an object falls into this category, its graphical data is read from a corresponding graphical

file and is displayed in the user's view at the correct expected location. Figure 3.28 presents snapshots of the 4D graphical AR simulation.

Figure 3.28 – AR Visualization of a 4D CAD Model in UM-AR-GPS-ROVER

As discussed earlier, the accuracy of the acquired positioning data from the GPS receiver was always a major concern. However, in the validation stage of UM-AR-GPS-ROVER and in order to prove the feasibility of the developed methods and algorithms in this research, it was decided to choose WAAS technology since it provides a free GPS signal correction service. Since the methods and functionalities developed in this research were

designed in compliance with NMEA which is a widely accepted standard for GPS data transmission, future upgrades to higher accuracy devices would not affect the integrity of the AR platform. This is mainly because the issue of accuracy is more internal to the incoming data streams from the GPS device and as long as these data streams follow the NMEA format, any future developed AR platform using the provided registration methods will be capable of receiving and extracting them in the form of useful position values.

Considering all these factors, in choosing the location of the outdoor tests at this stage given the geographical layout of the state of Michigan, the main factor was to perform the tests in areas in which the southern sky horizon was clear enough and not blocked by either natural (e.g. trees) or human-made (e.g. buildings) objects for the WAAS satellite to be viewed by the GPS receiver. This in fact led to an increased level of accuracy in the validation stage. To be more specific, the overall accuracy level achieved in outdoor tests was consistently about one meter which is an acceptable range considering that the GPS receiver used in these tests was not a high accuracy device.

3.8. Summary and Conclusions

The primary advantage of graphical simulation in AR compared to that in VR is the significant reduction in the amount of effort required for CAD Model Engineering. AR superimposes graphical images of virtual objects on top of a user's views of the real surrounding space thereby precluding the need to CAD model the entire simulation environment. In order for AR graphical simulations to be realistic and convincing, real objects and augmented virtual models must be properly aligned relative to each other. Without accurate registration, the illusion that the two coexist in AR space is compromised.

In this research, it was found that tracking systems traditionally used for AR registration are intended for use in controlled indoor spaces and are unsuitable for unprepared outdoor environments such as those found on typical construction sites. In order to

address this issue in the presented research, the global outdoor position (i.e. longitude, latitude, altitude) and 3D orientation (i.e. yaw, pitch, and roll) of a user's viewpoint were tracked using a GPS receiver and a 3DOF head orientation tracker respectively. The tracked information was reconciled with the known global position and orientation of CAD objects to be superimposed in the user's view. Based on this computation, the relative translation and axial rotations between the user's eyes and the CAD objects were calculated at each frame during visualization. The relative geometric transformations were then applied to the CAD objects to generate an augmented outdoor environment where superimposed CAD objects stayed fixed to their real world locations as the user moved freely in the animation.

In order to validate the designed registration algorithms in real life situations, they were integrated into UM-AR-GPS-ROVER, an AR animation prototype equipped with a mobile computing apparatus. UM-AR-GPS-ROVER is the result of a successful integration of a general purpose and modular hardware and software interface that can be readily shared and reused by researchers exploring AR in other fields. UM-AR-GPS-ROVER, as a reconfigurable and extensible platform, facilitates future component replacements and code improvement by allowing potential AR users to make modular modifications to necessary application components instead of exploring and developing an entirely new AR application in response to new technologies and scientific improvements. It coordinates operations of the hardware and software components, and interfaces with the user and their environment simultaneously, with the ultimate goal of seamlessly integrating virtual objects into the real world surroundings.

While the author's primary scientific domain of interest is construction, the principles and components described in this Chapter can be extended to developing any AR or location-aware application in other engineering and scientific domains. The modular design of this platform allows for reusability and pluggability of methods within different AR domains, and allows for continual modernization of equipment to incorporate new technologies and procedures that emerge as AR research continues. As a result, the algorithms and methods described in this Chapter can be conveniently reused by other

AR application developers simply by importing the provided class libraries into their code thereby saving a significant amount of time and effort that would be otherwise required to reimplement low level software interfaces to perform registration and data acquisition tasks.

3.9. References

[1] Behzadan, A. H., and Kamat, V. R. (2007), "Georeferenced Registration of Construction Graphics in Mobile Outdoor Augmented Reality", Journal of Computing in Civil Engineering, 21(4), American Society of Civil Engineers (ASCE), Reston, VA, 247-258.

[2] Portland Mall Light Rail Project Photo Gallery – available at: http://www.portlandmall.org/about/photogallery.htm (accessed: November 2007)

[3] Yuan, M. L., Ong, S. K., and Nee, A. Y. C. (2005), "A Generalized Registration Method for Augmented Reality Systems", Journal of Computer and Graphics, 29(6), Elsevier Science, New York, NY, 980-997.

[4] Kaufmann, H., and Schmalstieg, D. (2003), "Mathematics and Geometry Education with Collaborative Augmented Reality", Journal of Computer and Graphics, 27(3), Elsevier Science, New York, NY, 339-345.

[5] Dunston, F., Wang, X., Billinghusrt, M., and Hampson, B. (2002), "Mixed Reality Benefits for Design Perception", In Proceedings of the 19th International Symposium on Automation and Robotics in Construction (ISARC), NIST, Gaithersburg, Maryland, 191-196.

[6] OSGART (ARToolkit for OpenSceneGraph) – avaialble at: http://www.artoolworks.com/community/osgart/gallery.html (accessed: November 2007)

[7] Azuma, R., Lee, J. W. Jiang, B., Park, J., You, S., and Neumann, U. (1999), "Tracking in Unprepared Environments for Augmented Reality Systems", Journal of Computer and Graphics, 23(6), Elsevier Science, New York, NY, 787-793.

[8] Piekarski, W., Smith, R., and Thomas, B. (2004), "Designing Backpacks for High Fidelity Mobile Outdoor Augmented Reality", In Proceedings of the 3rd International Symposium on Mixed and Augmented Reality (ISMAR04), IEEE & ACM, Arlington, Virginia.

[9] Behzadan, A. H., Timm, B. W., and Kamat, V. R. (2008), "General Purpose Modular Hardware and Software Framework for Mobile Outdoor Augmented Reality Applications in Engineering", Journal of Advanced Engineering Informatics, 22(1), Elsevier Science, New York, NY, 90-105.

[10] Kamat, V. R., and Behzadan, A. H. (2006), "GPS and 3DOF Angular Tracking for Georeferenced Registration of Construction Graphics in Outdoor Augmented Reality", In Lecture Notes in Computer Science (Intelligent Computing in Engineering and Architecture), Ian F.C. Smith (Editor), Springer, New York, NY, 368-375.

[11] El-Rabbany, A. (2002), Introduction to GPS: The Global Positioning System, Artech House Inc., Norwood, MA.

[12] Lastra, A. A. (1997), "Technology for Virtual Reality", Department of Computer Science, University of North Carolina, Chapel Hill, NC.

[13] Behzadan, A. H., and Kamat, V. R. (2006), "Animation of Construction Activities in Outdoor Augmented Reality", In Proceedings of the 11th Joint International Conference on Computing and Decision Making in Civil and Building Engineering (ICCCBE-XI), Montreal, Canada, 1135-1143.

[14] Barfield, W., and Caudell, T. (2001), Fundamentals of Wearable Computers and Augmented Reality, Lawrence Erlbaum Associates, Philadelphia, PA.

[15] Booch, G., Maksimchuk, R. A, Engel, M. W., Brown, A., Conallen, J. Houston, K. A. (1994), Object-Oriented Analysis and Design with Applications, Addison Wesley, Reading, MA.

[16] Behzadan A. H., and Kamat V. R. (2007), "Reusable Modular Software Interfaces for Outdoor Augmented Reality Applications in Engineering", International Workshop on Computing in Civil Engineering, American Society of Civil Engineers (ASCE), Pittsburgh, PA, 825-837.

[17] Vincenty, T. (1975), "Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations", Survey Review, 176, Maney Publishing, Leeds, UK, 88-93.

[18] Behzadan, A. H., and Kamat, V. R. (2005), "Visualization of Construction Graphics in Outdoor Augmented Reality", In Proceedings of the Winter Simulation Conference (WSC), IEEE, Orlando, FL, 1914-1920.

[19] Kamat, V. R., and Martinez, J. C. (2001), "Visualizing Simulated Construction Operations in 3D", Journal of Computing in Civil Engineering, 15(4), American Society of Civil Engineers (ASCE), Reston, VA, 329-337.

[20] Koo, B., and Fischer, M. (2000), "Feasibility Study of 4D CAD in Commercial Construction", Journal of Construction Engineering and Management, 126(4), American Society of Civil Engineers (ASCE), Reston, VA, 251-260.

Chapter 4

Construction and Management of a Dynamic Augmented Scene

4.1. Introduction

An augmented scene presents a combination of virtual and real objects spatially (and possibly logically) arranged and linked in a single view. Real world objects are independent entities that occupy certain space in the 3D environment and can potentially change their position and orientation over time. Virtual objects, however, have to be first calculated and appropriately placed in the scene before they can be manipulated. There are two major approaches to model and place virtual objects in a graphical scene. Such models can be either created as a *single* unit or assembled from *discrete* components [1]. Modeling the scene as a single unit requires the modeler to create a static model with all the individual elements appropriately placed in their correct position and orientation in the scene. The resulting model using this approach is usually suitable for graphical applications in which the main purpose is to render, visualize, and examine large data sets ranging from individual components, to subassemblies, to entire complex layouts [1].

Another approach which is more effective for animating a graphical scene is to create the entire graphical contents from discrete CAD models. This provides the modeler with high manipulation level to individually change position, orientation, and scale of each of the CAD objects without affecting the integrity of the scene. Moreover, the individual scene

components can be created in separate modeling tools (e.g. AutoCAD, Photoshop, AC3D, 3D Studio). This brings significant flexibility and speed to the modeling and animation processes. In this research, discrete component modeling approach was selected as it results in more flexibility and higher manipulation control when animating a scene consisting of virtual and real objects.

4.2. Main Contributions

In order to perform a construction operation, several types of physical resources (i.e. equipment, personnel, material) have to be deployed on the site. As a result, for a 3D visualization to be a realistic representation of the complex dynamics of a typical construction site, it must be able to demonstrate, manipulate, and manage a significant amount of CAD objects included in the simulation model of the construction operation. The common problem arising in almost every large scale visualization system is the amount of data that must be loaded and rendered at each frame. As the size and complexity of the animated environment increases, slow rendering speed becomes the bottleneck for real time user interaction with the scene even if fast CPU speed and powerful graphic accelerator cards are used [2,3]. Overcoming the hardware limitations by reducing the amount of time and CPU work required to handle the CAD objects inside an AR animation was the main motivating factor for this part of the research. ***The primary contribution of the research presented in this Chapter is a flexible and dynamic methodology to facilitate the tasks of constructing and managing the contents of the AR animation of a simulation model by continuously loading, rendering, and displaying the virtual CAD models and real surrounding environment in which the operation takes place.***

The objective of this part of the research was successfully achieved by first arriving at a computer interpretable description of the visible space in front of the user's eyes (i.e. viewing frustum). This helped in limiting the number of CAD objects necessary to be rendered at each animation frame. In particular, only CAD objects located inside this visible space need to be loaded and displayed which significantly decreases the

computation load on the graphics card and CPU and at the same time increases the frame rate. The next major step was to establish an efficient object structure to optimize the management of CAD objects inside the user's viewing frustum. This was achieved by applying the concept of scene graphs and introducing a new Augmented Reality (AR) based hierarchical object structure to facilitate CAD object manipulation inside the AR animation.

4.3. Creating an AR Perspective Viewing Frustum

The very first step in creating an animated scene is to define the space in which virtual and real objects coexist. Without having a precise contextual definition of the 3D scene, real and virtual objects in the scene are not able to establish spatial and contextual links with each other and as a result, proper registration (as discussed in Chapter 2) cannot be achieved and maintained throughout the animation. In order to construct the augmented space, the first set of information is the location and dimensions of the visible world to the observer of the scene. This is achieved by defining a viewing frustum centered on the eyes of the user (i.e. scene observer). Since the real world is perceived by the human eyes in a perspective manner, the same visual basis has to be applied to the virtual contents of the augmented world in order to ensure proper alignment of the two worlds. As a result, a perspective viewing frustum has to be defined with the user's eyes always located at the projection point.

Figure 4.1 shows the parameters of a perspective viewing frustum used in this research. In this Figure, the volume of space inside the truncated pyramid confined by near plane (marked by points P_1 through P_4) and far plane (marked by points P_5 through P_8) is called the perspective viewing frustum. Objects inside this limit are visible to the user and hence have to be displayed and animated by the AR platform. Other objects are either culled (if completely located outside the frustum) or clipped (if partially located inside the frustum) [4]. Since the user (observer of the scene) can potentially move in the scene, the origin (i.e. frustum projection point) is not fixed and moves with the user. As a result, the

contents of the perspective viewing frustum may change over time due to the user and/or the objects changing position (and orientation).

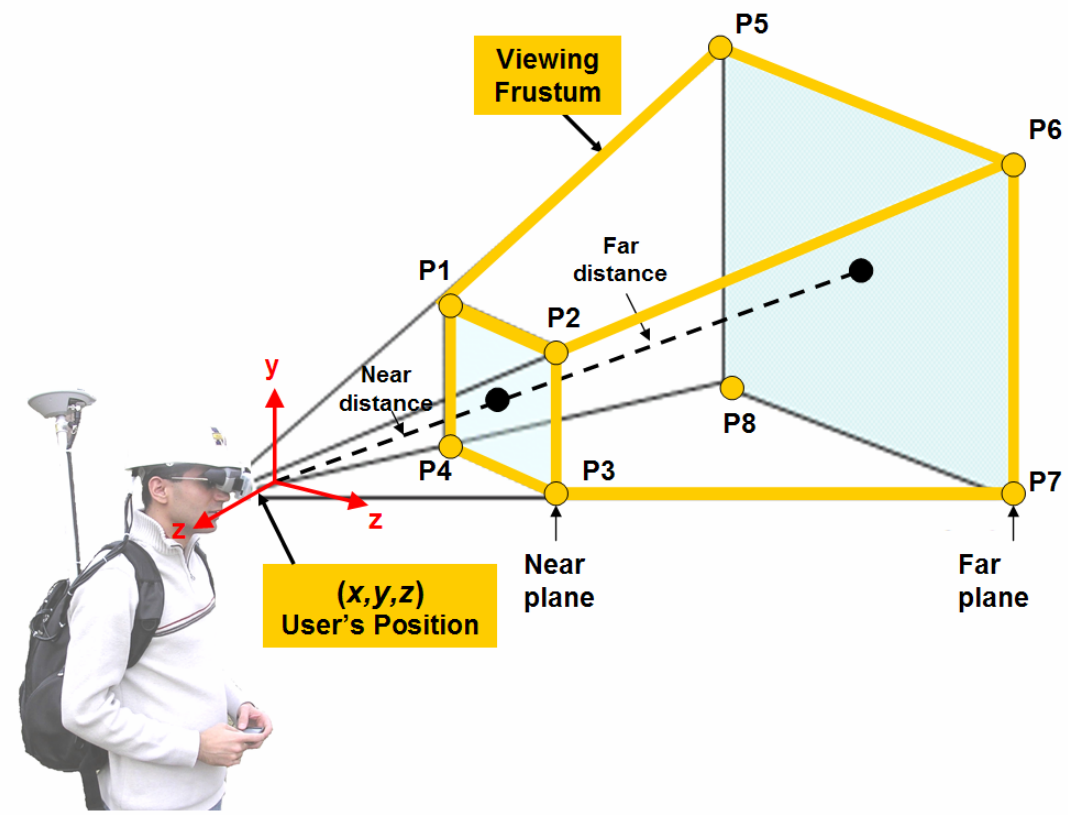


Figure 4.1 – Definition of a Perspective Viewing Frustum

Note that relative distances of near and far planes to the user can be changed depending on the amount of the 3D space the user expects to observe. In addition, the extent of the frustum can be changed by increasing or decreasing the horizontal and/or vertical field of view angles as shown in Figures 4.2 and 4.3. In these Figures, the Vertical Field of View (VFOV) angle is set to be 45 degrees (Figure 4.2) and the Horizontal Field of View (HFOV) angle is set to be 60 degrees (Figure 4.3). The near and far planes in these Figures are also set to be 1 and 500 meters from the user respectively. A virtual object is visible to the user as long as it is located inside the perspective viewing frustum. The real world contents of the augmented scene (which are captured by the video camera) are also visible to the user as long as they are inside the camera perspective viewing frustum. To achieve best results, different parameters of the virtual and real perspective viewing

frustums (near plane, far plane, VFOV angle, and HFOV angle) had to be set to allow for proper alignment of the two worlds in one single augmented view.

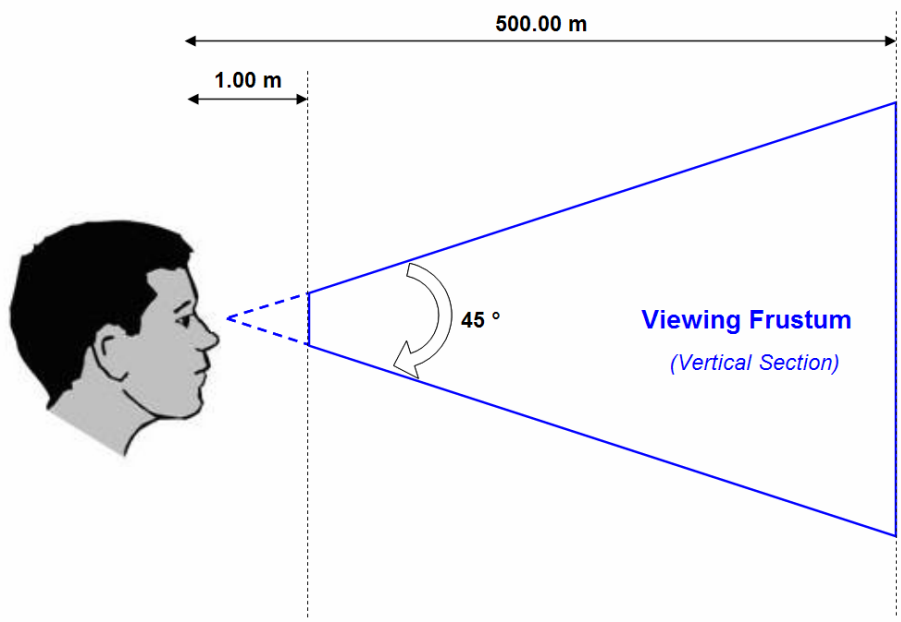


Figure 4.2 – Side View of a Perspective Viewing Frustum

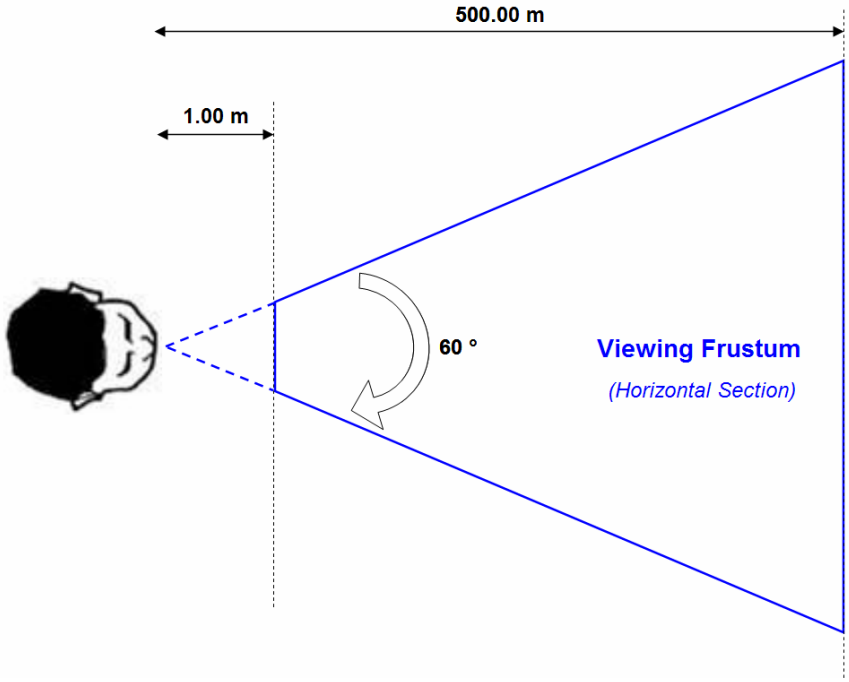


Figure 4.3 – Planar View of a Perspective Viewing Frustum

4.4. Developing the Augmented Scene Graph

The concept of scene graphs can be effectively used to facilitate the creation and management of assembled scenes. In general, a scene graph is a data structure used to organize and manage the contents of hierarchically organized scene data [1,5]. In the realm of computer graphics, scene graphs are means to create and manipulate hierarchical organization of shapes, groups of shapes, and clusters of groups that collectively define the content of a scene [5]. Scene graphs address low level problems that generally arise in scene composition and management. In computer animation, scene graphs are widely used as structures that arrange the logical and spatial representation of a graphical scene [1,6]. Scene graphs allow the scene modeler to focus on what 3D objects to render rather than how to render them. It eliminates several low level programming complexities involved in rendering process of 3D objects in a scene [1,7].

There are several computer graphics implementations that build on the concept of scene graphs. Some of them (e.g. Java3D [8]) are in form of an Application Programming Interface (API), designed to be integrated into standard programming languages such as C/C++ or Java. Alternatively, scene graphs may be in the form of a text or binary file. Such files usually adopt an authoring (or markup) language format and are generally synonymous with web applications. Examples of two such implementations are the Virtual Reality Modeling Language (VRML) and Extensible 3D (X3D). VRML is an international standard similar to Hyper Text Markup Language (HTML) [9,10], whereas X3D works within the constructs of the Extensible Markup Language (XML) file format [11,12]. In order to use the advantages of scene graphs in creating graphical contents of the augmented scene in this research, an open source, cross platform graphics toolkit called OpenSceneGraph was used. Based around the concept of scene graphs, OpenSceneGraph provides an object oriented framework on top of OpenGL freeing the developer from implementing and optimizing low level graphics calls, and provides many additional utilities for rapid development of graphics applications [13].

Each scene graph is in fact a collection of nodes in a graph structure connected together via child-parent relationship. A node is an object that can be part of or entirely comprise a scene graph. Each node is a collection of one or more *fields* (values) and *methods* that together perform a specific function. Each node encapsulates the semantics of what is to be drawn. A method applied to a parent node will affect all its child nodes. In order to build a larger scene graph, smaller scene subgraphs can be created separately and attached together via links to the highest level node called the *root node* [1, 6]. A *link* connecting two separate nodes describes the relationships that exist between them in a meaningful way. The nodes are ideally arranged in a hierarchical manner that corresponds spatially and semantically to the modeled scene [14].

Figure 4.4 shows a sample scene graph. In this Figure, the node `Jobsite` is the root node. Scene subgraphs are created and attached to the root node to complete the scene structure by encapsulating the entire jobsite. In Figure 4.4, scene subgraphs `Dozer`, `Truck`, and `Terrain` are all children of the root node. Nodes `Dozer` and `Truck` have also their own children nodes connected to them at the lowest level of the hierarchy. These lowest level nodes contain the geometrical description of the individual components of their parent nodes.

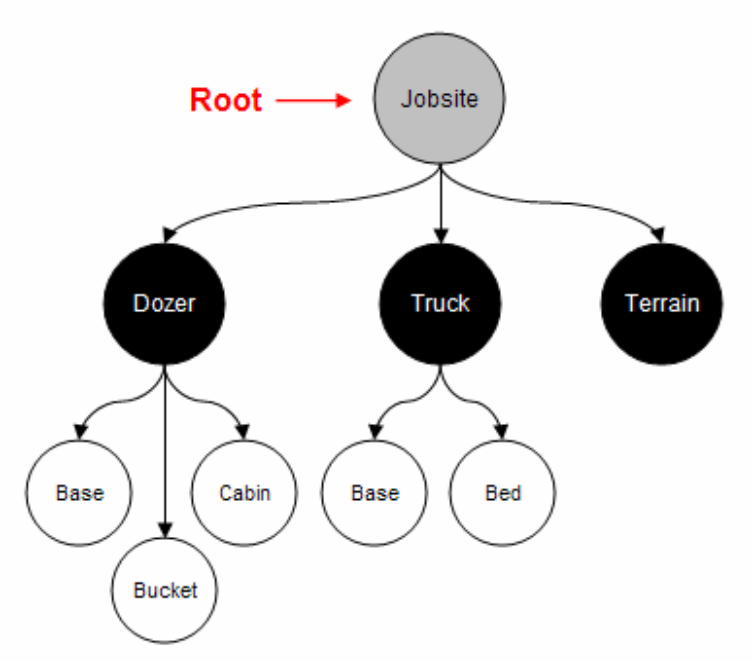


Figure 4.4 – A Sample Scene Graph Hierarchy

By definition, `Dozer` and `Truck` nodes that group together a number of geometrical nodes are called *group nodes*. The low level nodes containing the geometrical description of the components are called *leaf nodes*. A single CAD file can be assigned to each leaf node which contains the geometric shape, properties, and dimensions of the node. The traditional scene graph as shown in Figure 4.4 has been previously used by researchers to create Virtual Reality (VR) based animations [1]. Although this type of scene graph correctly represents the graphical contents of an imaginary construction jobsite, there is still no indication of the real world and the observer of the scene as part of the real world for that matter. Since this research was focused on augmented reality visualization, a new hierarchical representation of the animated scene had to be conceptualized and developed in order to encapsulate both groups of real and virtual objects in one scene graph. As described in Chapter 3, in a mobile AR visualization system, the augmented graphics have to be precisely registered and displayed in accurate positions relative to the observer's eyes. As a result, the developed AR-based scene graph must be able to store, retrieve, and display graphical data from a viewpoint that represents the observer's eyes. Since the observer can constantly change position and/or head orientation in the animation, this viewpoint has to change accordingly which will cause the entire animated contents of the scene to change. In order to consider the latest changes in the observer's global position and head orientation angles, a mobile root node is used in the adopted AR-based scene graph in this research. The designed AR-based scene graph is shown in Figure 4.5 in which the same imaginary jobsite is represented in the presence of the real world. The root node in the scene graph shown in this Figure represents a moving viewpoint connected to the observer's eyes and moves as the observer changes position and/or head orientation in the augmented animation.

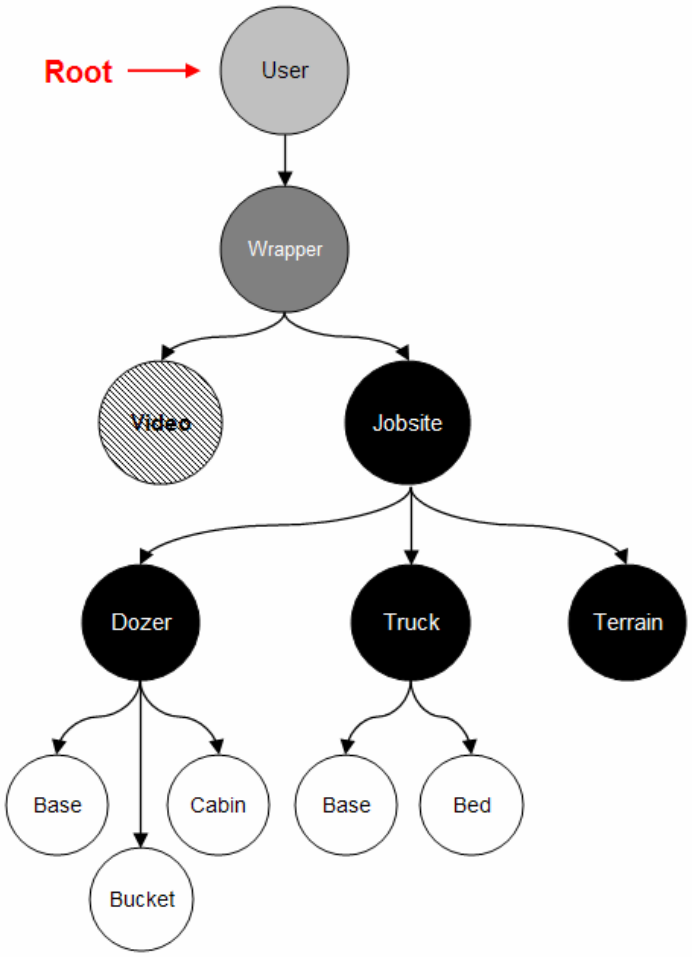


Figure 4.5 – The Structure of the Developed AR-based Scene Graph

In this Figure, a new *video node* is also a part of the scene graph. This node is not a regular node in the AR-based scene graph as it does not store any geometrical data to represent a single CAD model or a group of CAD objects. Instead, it is capable of storing and displaying the latest video texture captured by the video camera in the background. The AR application combines this texture data with the original virtual world data through the *wrapper node* to construct the augmented representation of the scene.

As described earlier, there is a major factor that distinguishes the two scene graphs presented in Figures 4.4 and 4.5. In a purely VR-based scene graph (Figure 4.4), the root node has a fixed position and orientation in the scene and the CAD objects connected to this node through the scene graph only change position and orientation when they move or rotate. However, in an AR-based scene graph (Figure 4.5), the root node itself moves

and/or rotates as a change occurs in the user's position and head orientation. While CAD objects may change their position and orientation due to them being individually manipulated in the scene, any change in user's position and orientation will affect the entire scene graph and updates the augmented view immediately [15].

There are two coordinate systems involved in creating a scene graph: world (global) coordinate system, and object (local) coordinate system. Since the position of the root node in the scene graph presented in Figure 4.4 is always fixed, objects can be placed at a relative distance from the root assuming the root coordinates to always be the origin of the global coordinate system during the animation. However, in the AR-based scene graph shown in Figure 4.5, the position and orientation of the root node can potentially change over time due to any user's movement in the augmented scene. As a result, the global position of the root node (i.e. the user) has to be acquired in every frame using accurate tracking devices and the entire scene has to be reconstructed based on the new positional and orientation values of the root node [15].

As described in Chapter 3, these values are mainly obtained from the tracking devices connected to the mobile user such as Global Positioning System (GPS) receivers, Three Degree-of-Freedom (3DOF) head orientation trackers, and motion sensors [16, 17]. The author has previously developed a reusable framework for real time communication with standard GPS devices and head orientation trackers [18]. The term *standard device,* in the context of this dissertation, is defined as a tracking device which complies with a certain data transmission protocol. For example, the framework developed in this research is capable of establishing real time communication with every type of GPS receivers as long as it outputs data streams (containing longitude, latitude, and altitude) following the National Marine Electronics Association (NMEA) standard [19]. In addition, and as described earlier in Chapter 3, the developed framework can communicate with a wide range of head orientation tracking devices that send binary data packets containing the 3D head orientation angles (i.e. yaw, pitch, and roll angles).

The AR scene is created by appropriately placing individual CAD objects each of which defined by a geometrical model. Each geometrical model is created in its own local coordinate frame, stored as a leaf node in the AR-based scene graph, and placed at an appropriate position and orientation inside the coordinate frame of its parent node. This is accomplished using *transformation nodes*. As shown in Figure 4.6, transformation nodes allow scene graph developers to manipulate the location (translation), orientation (rotation), and size (scale) of different nodes. They are basically group type nodes that translate the local coordinates of their child nodes into the coordinates of their parent nodes [1,14].

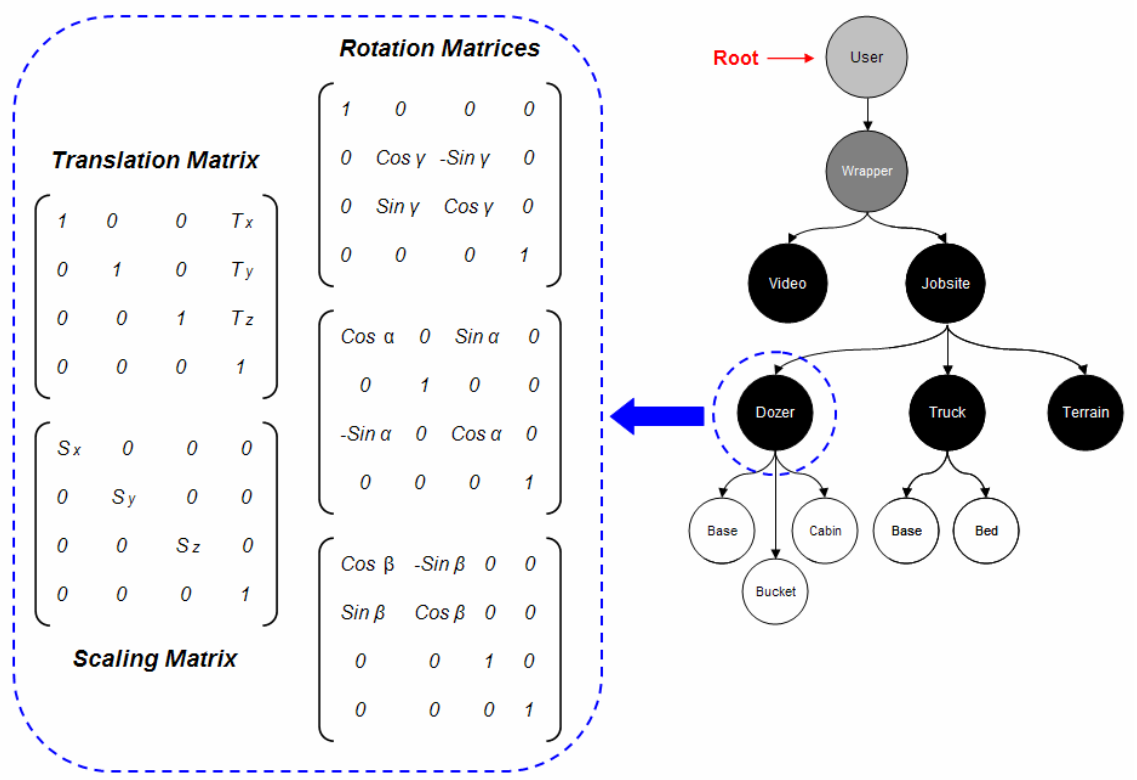


Figure 4.6 – Transformation Matrices Connected to a Scene Graph Node

Transformation nodes are an implicit part of the scene graph. For example, in Figures 4.4 and 4.5, each of the group nodes `Truck`, `Dozer`, and `Terrain` has an implicit transformation node in order to be able to be placed and oriented at desired positions relative to the node `Jobsite`. Moreover, the node `Jobsite` in Figure 4.5 has an implicit transformation node so it can be positioned and oriented relative to the mobile

user at each instant of time. The overall transformation of a single CAD object relative to the root node is obtained by multiplying the individual matrices of Figure 4.6 as follows:

$$T_{object}^{user} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos\gamma & -Sin\gamma & 0 \\ 0 & Sin\gamma & Cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Cos\alpha & 0 & Sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -Sin\alpha & 0 & Cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Cos\beta & -Sin\beta & 0 & 0 \\ Sin\beta & Cos\beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation data of the scene (model) relative to the root node (world) is stored in a special matrix called *model view matrix*. In an AR-based scene graph, as shown in Figure 4.5, the root node always represents the user's eyes. Hence, there is a one to one correspondence between the user's eyes and the origin of the animated scene. As a result, the identity matrix is often used as the initial model view matrix. In order to draw a scene graph, its nodes are traversed downward starting from the root node and the following operations are performed for each link out of the root node:

- The current model view matrix is saved by pushing it onto the matrix stack.
- The model view matrix on the right is multiplied by the transformation matrix associated with the link.
- The drawing procedure is called recursively, pretending that the endpoint of the link is the root.
- The original model view matrix is restored by popping it from the matrix stack.

From the visualization point of view, the main difference between traditional VR-based and modified AR-based scene graphs is the mechanism through which different views of the scene are displayed to the observer (i.e. the user). In a purely VR environment, this is done by changing viewpoints which are basically a set of hidden cameras in the scene. The user can switch between cameras in order to view the scene from different perspectives. Cameras are not part of the scene graph and hence do not have a nodal representation. They are external mechanisms for visualizing the encapsulated data in the scene graph. In contrast, in an AR-based scene graph the term viewpoint has a completely

different definition. The scene is viewed by mobile users as they walk in the animation. As such, there is only one moving viewpoint attached to a user's eyes. This means the only viewpoint in an AR animation which involves a mobile root node is a Six Degree-of-Freedom (6DOF) viewpoint attached to the scene observer's eyes. Users can change their global position (i.e. longitude, latitude, and altitude) while simultaneously rotating their head in 3D global space (i.e. yaw, pitch, and roll angles) [16]. As shown in Figure 4.5, this single viewpoint has a nodal representation and in fact, serves as the root of the scene graph.

4.5. Animating and Updating the Augmented Scene Graph

Animating a scene graph is achieved by manipulating the values of the transformation nodes. Position, orientation, and scale of each component in the scene can be manipulated relative to its parent node. Updating these values at each frame leads to a dynamically changing scene. Since this is done continuously over time, a smooth animation is displayed to and viewed by the user. Figures 4.7 and 4.8 show how a virtual CAD model of a dozer (previously introduced in Figure 4.5) can be manipulated inside the perspective viewing frustum. As described earlier, the relative position of the dozer can be changed due to two main reasons:

1) Dozer moves in the scene from one point to another, or
2) User's global position changes and new GPS data is captured by the application.

Figure 4.7 shows the situation in which the user is looking at the augmented scene from a fixed position and the CAD model of the virtual dozer changes its position. In fact, this situation simplifies the AR-based scene graph and makes it very similar to the traditional VR-based scene graph of Figure 4.4 in which the root node is fixed in the global space. As the dozer moves 5 meters along the positive X axis, its global coordinates change as well. Knowing the global position of the user in terms of longitude, latitude, and altitude

(coming through the GPS device), the relative distance between the user and the dozer is calculated at each frame using the Modified Transformation Method (MTM) described in Chapter 3. The values of the dozer transformation node are updated based on the calculated relative distance and the position of the dozer is modified accordingly inside the user's viewing frustum. As a result, the user views a moving dozer inside the augmented viewing frustum.

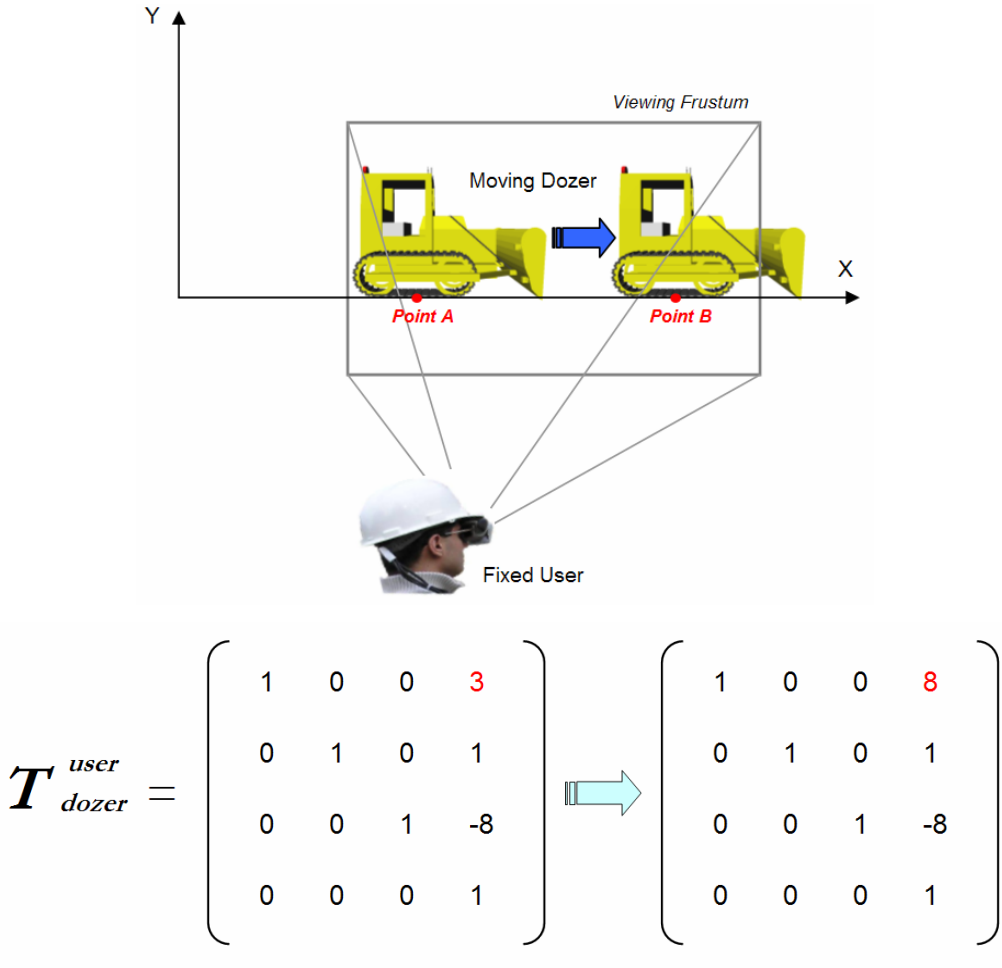
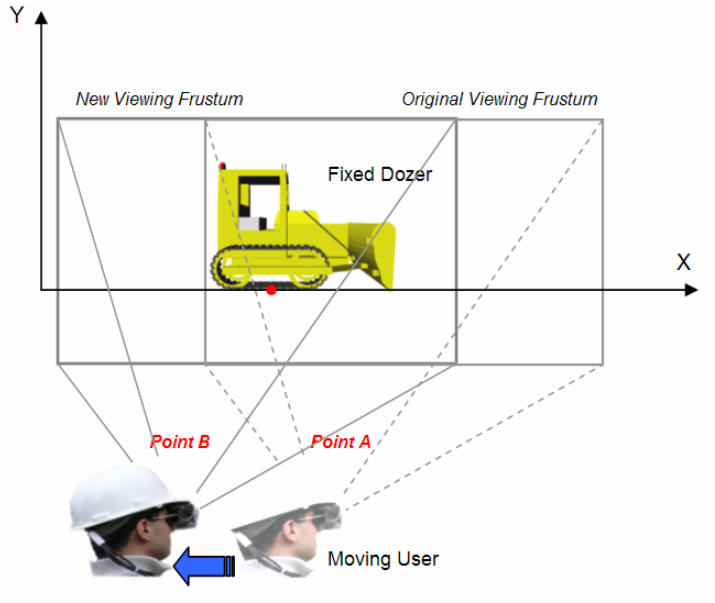


Figure 4.7 – Animating a Moving CAD Object in an AR Scene with a Fixed User

As shown in Figure 4.7 and recalling the transformation matrices assigned to each CAD object from Figure 4.6, the dozer's movement along the X axis from point *A* to point *B* will directly affect the translation component of its transformation matrix. In this Figure, T_{dozer}^{user} denotes the transformation matrix of the dozer CAD model inside the local

coordinate frame of the user. In another situation shown in Figure 4.8, the CAD model of the virtual dozer is fixed and the user changes position over time.



$$T_{dozer}^{user} = \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -8 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 & 8 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -8 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 4.8 – Animating a Fixed CAD Object in an AR Scene with a Moving User

As the user moves 5 meters to the left, global coordinates are continuously acquired from the GPS device. Again, knowing the global coordinates of the CAD object, the relative distance between the user and the dozer is calculated at each frame using the MTM described in Chapter 3. Consequently, the transformation values of the dozer node are updated based on the calculated relative distance and the position of the dozer is modified accordingly inside the user's viewing frustum. As a result, the user movement to the left is interpreted as a translation of the dozer to the right. This is essentially equivalent to the case in which the user is fixed and the dozer moves to the right inside the viewing

frustum. As shown in Figure 4.8 and recalling the transformation matrices assigned to each CAD object from Figure 4.6, user's movement along the negative X axis from point *A* to point *B* by 5 meters will change the translation component of the transformation matrix assigned to the dozer CAD model along positive X axis by 5 meters. In this Figure, T_{dozer}^{user} denotes the transformation matrix of the dozer CAD model inside the local coordinate frame of the user.

A very similar procedure is applied to the case in which the user's head orientation changes. Figure 4.9 shows an augmented viewing frustum in which a CAD model of a virtual truck is displayed to the user. As shown in this Figure and recalling the transformation matrices introduced in Figure 4.6, the initial orientation of the CAD object relative to the user's direction of look can be expressed in form of the identity matrix T_{truck}^{user}.

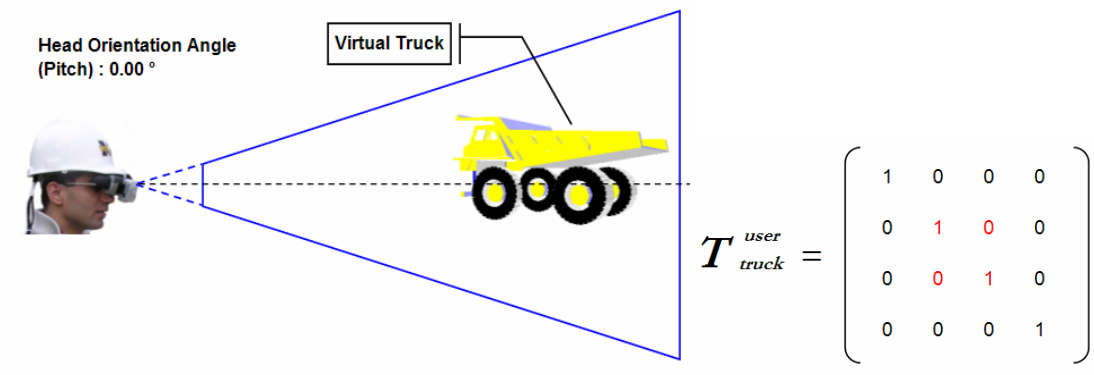


Figure 4.9 – Initial Orientation Configuration of a CAD Object

As shown in Figure 4.10, if the user's head rotates upward by 30°, the head orientation tracker determines a change in the pitch angle and the new pitch angle is sent to the AR application. Recalling Figure 4.5, since the user's eyes are at the root node of the augmented scene graph, the entire scene will be rotated 30° which results in a disoriented scene.

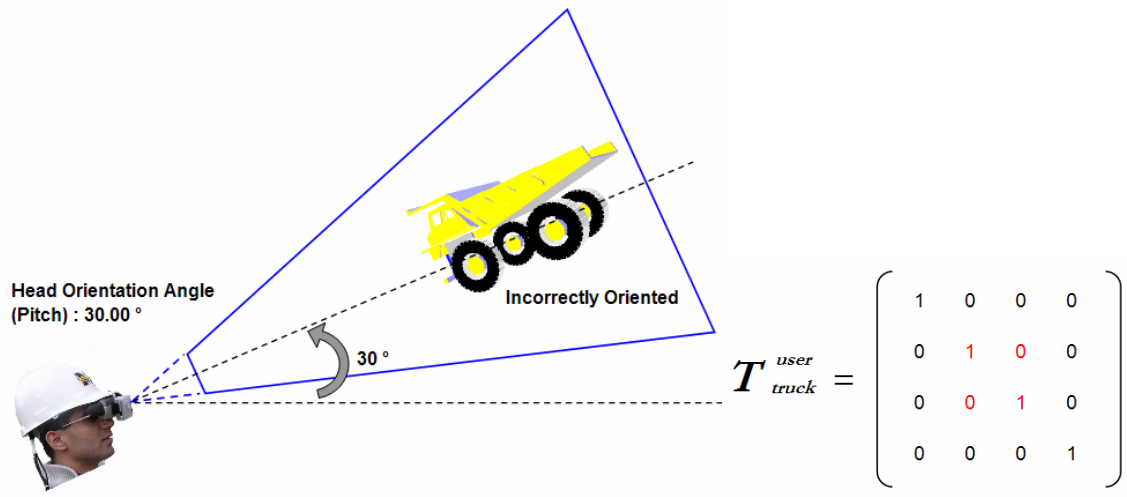


Figure 4.10 – Incorrectly Oriented Augmented Scene due to User's Head Rotation

This leads to a distorted and unacceptable visual output especially due to the fact that although the user's view of the real world (which is captured by the video camera) has been completely changed, the virtual contents of the scene have remained exactly the same. To fix this problem, the T^{user}_{truck} transformation matrix needs to be updated. This is shown in Figure 4.11 in which the orientation of the virtual truck is modified by -30°. The new transformation matrix will be used immediately at the next scene graph traversal (i.e. next animation frame) to update the position and orientation of the CAD object inside the user's viewing frustum. The same steps are necessary in case the user's head orientation changes about the other two major axis (i.e. roll and yaw angles).

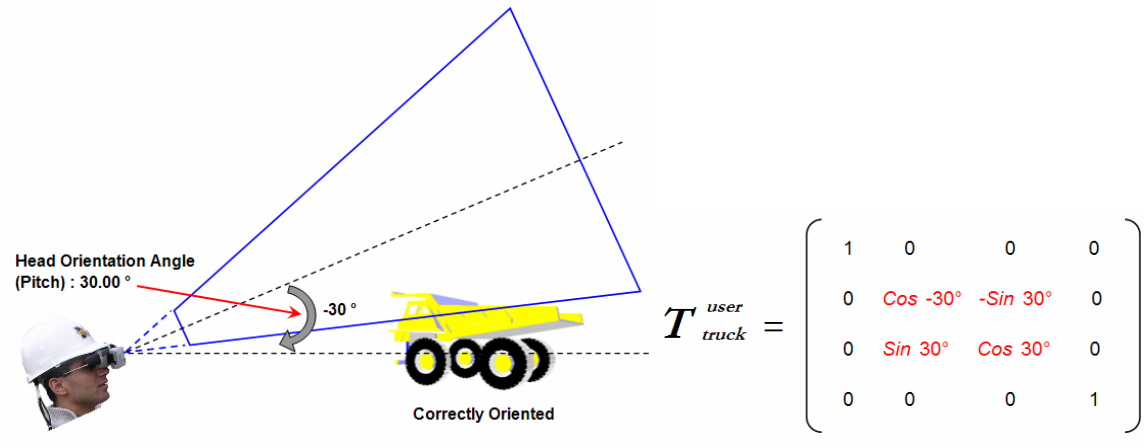


Figure 4.11 – Correctly Oriented Augmented Scene due to User's Head Rotation

The values of a transformation node can be manipulated as a result of simultaneous user and CAD object movements (i.e. change in position and orientation). In this case, each movement is handled separately using the procedures described above and the resulting values are combined to produce the final animated scene graph. In order to achieve a realistic animation, every CAD object has to move smoothly between two points rather than jump from one point to another. This is achieved by updating the viewing frame continuously at high refresh rates.

The construction of an augmented scene starts with extending the hierarchical structure of the corresponding AR-based scene graph by sequentially uploading and placing virtual CAD objects in the augmented view. Figure 4.12 shows how the structure of an AR-based scene graph is related to the actual contents of the augmented scene. Since leaf nodes contain geometry information, for every class of CAD objects, one leaf node has to be created. One or more CAD objects can then share the same leaf node if they have the same geometry. The scene graph of Figure 4.12 starts to take real shape as the node `RedTruck` is added to the scene graph and linked to its corresponding geometry through the leaf node `Truck`. In order to place the red concrete truck in the scene, the `RedTruck` transformation node is added to the parent node called `Jobsite`. The same procedure is applied to the scene graph again to create a new leaf node (i.e. `Form`), add an object conforming to this geometry (i.e. `WoodenForm`), and finally add the transformation node of the object to the parent node `Jobsite`. The last virtual object to be placed in the scene is a concrete bucket. Again, a geometry node (i.e. `Bucket`) is added to the scene graph. An object node (i.e. `OrangeBucket`) is added to the scene graph, and connected to the to the `Jobsite` node through a transformation node. In addition, note that the node `Jobsite` itself is a grandchild of the root node.

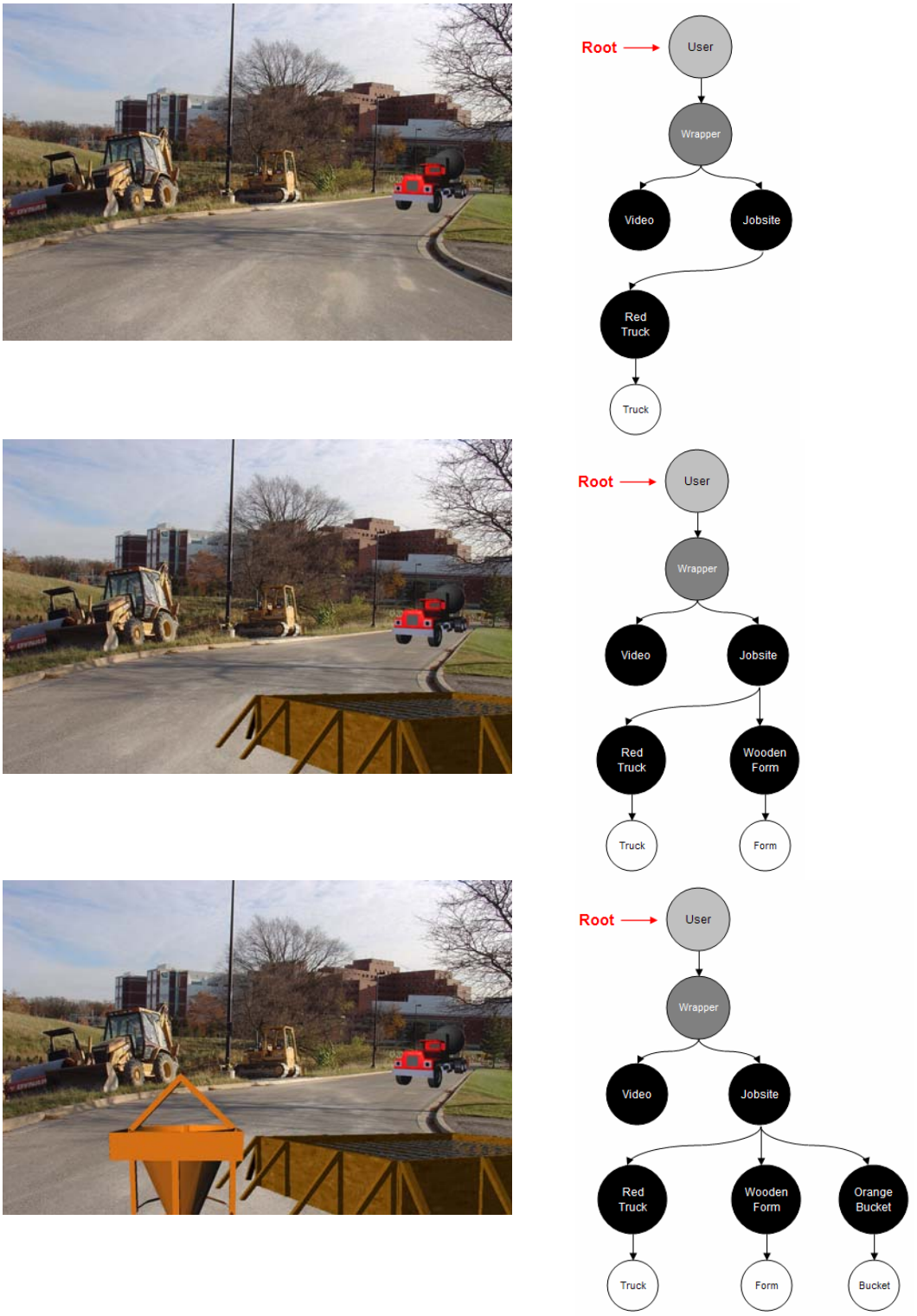


Figure 4.12 – Relation between the Scene Graph and the Augmented Scene

Scene graphs internally calculate and use bounding volume of each leaf node to determine which CAD objects are visible to the viewer [14]. A bounding volume is an

imaginary simple 3D primitive (e.g. box, sphere) around a more complex 3D object [20]. Bounding volumes are calculated from the leaf nodes upwards to the root node. Following this process, the bounding volume of a parent node in a scene graph is obtained by merging the individual bounding volumes of its immediate child nodes. Having calculated and stored all the bounding volumes in a scene graph, the graphics pipeline can then determine which CAD objects or groups of CAD objects are partially or completely outside the viewing frustum of the user and hence have to be culled. The bounding volume of a particular node encapsulates all the geometries described by all nodes that descend from it. As a result, if this bounding volume is outside the viewing frustum the entire subgraph is disregarded (culled) from the rendering stage. In scene graph terminology this is often referred to as *pruning* [14].

4.6. Geometric Instancing

Geometric instancing in the context of this dissertation is defined as the ability of the visualization engine to construct complex scenes potentially consisting of a large number of CAD objects from a limited number of virtual models and to maintain performance levels as the size of the operation increases [1]. The common problem arising in almost every large scale visualization system is the amount of data that must be loaded and rendered at each frame. As the size and complexity of the animated environment increases, slow rendering speed becomes the bottleneck for real time user interaction with the scene even if fast CPUs and powerful graphic accelerator cards are used [2,3].

Scene graphs support geometric instancing when constructing the graphical contents of an augmented scene. The application of scene graphs allow the modeler to conveniently create all CAD objects in the scene that use the same geometrical representation by making instances of a single CAD file associated with a class of objects. As a result, the CAD file needs to be loaded only once by the application which saves a lot of memory and running time and speeds up the process of scene rendering. Figure 4.13 shows the AR-based scene graph of a scene in which three identical dump trucks are placed in the augmented view.

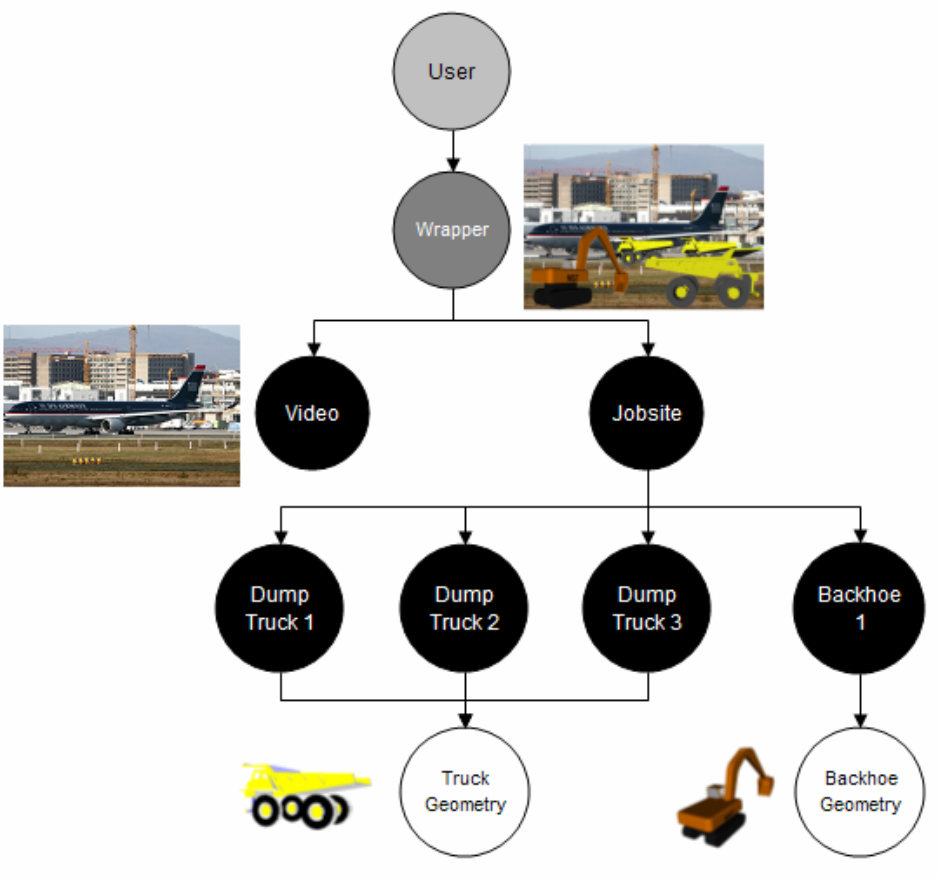


Figure 4.13 – Using Geometric Instancing in the AR-Based Scene Graph

Although there are four CAD objects present in the scene (i.e. one backhoe and three dump trucks), only two distinct CAD models need to be loaded (i.e. one for truck geometry and one for excavator geometry). In theory, an infinite number of identical dump trucks can be created and placed in the scene using the same geometrical representation of the truck already introduced to the scene graph as a leaf node.

Using geometric instancing, even more complex scenes such as an entire steel frame structure consisting of hundreds of beams and columns can be depicted by loading only a few CAD models of steel sections, and placing them repeatedly at appropriate locations using multiple transformation nodes. This is due to the capability of the transformation nodes to position, rotate, and scale the same geometry. Figure 4.14 shows how the scene graph of a complex augmented scene is constructed and populated from a limited number of CAD models.

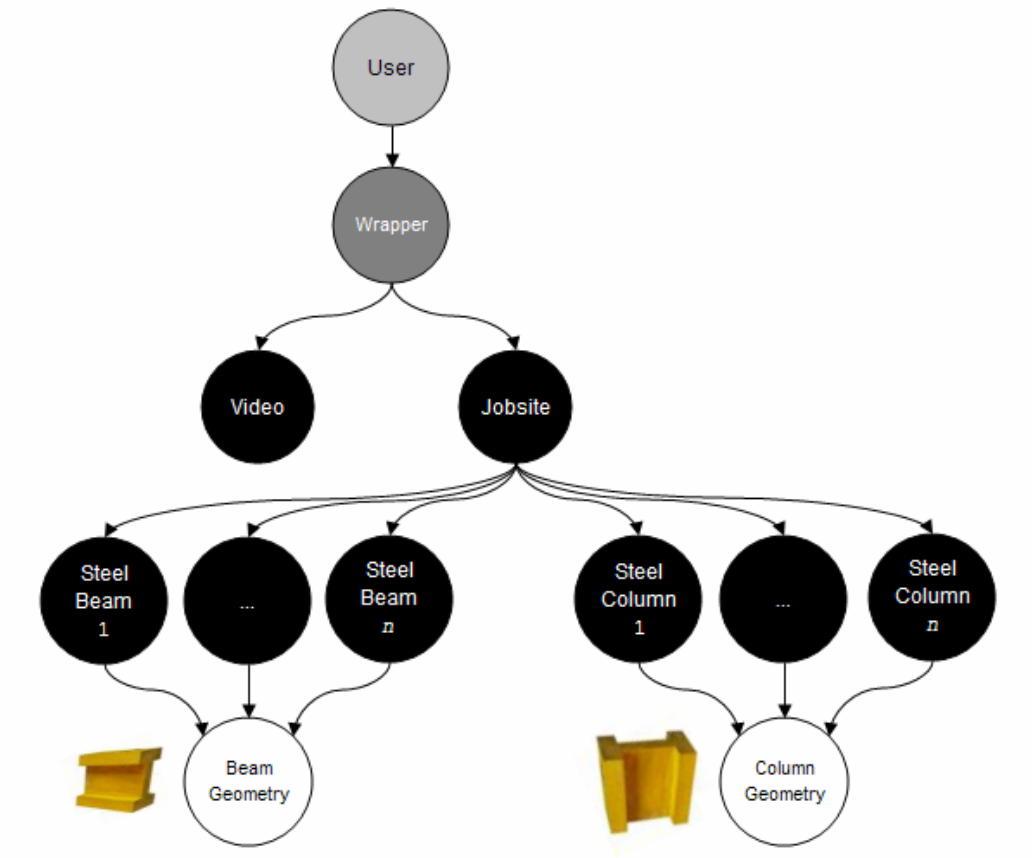


Figure 4.14 – Creating Large Number of Elements from a Few CAD Models

4.7. Summary and Conclusions

Developing and implementing effective CAD object management methods is a crucial step in order to create realistic visual output in AR animations of simulated processes. Virtual objects should be designed and displayed as independent entities (similar to real objects) in an AR scene so that their position, orientation, and size can be easily manipulated during the course of the animation. This is essential in creating the illusion that both groups of objects (real and virtual) coexist in a single scene and operate in a seamless manner. The high variability involved in a simulated process as dynamic as a construction operation requires the graphical engine of the AR application to create, load, and handle a large amount of 3D objects in each animation frame. As the size of the operation increases, CAD object manipulation becomes a memory intensive task that can

easily reduce the animation speed and the ability to interact with the animated process in real time.

Establishing a hierarchical database of all CAD objects operating in an augmented scene is a promising approach to optimize the process of manipulation of the virtual contents of an AR animation. Scene graphs have proven to be effective means in organizing such databases and are widely supported by several industrial standards and commercial graphical libraries [3,4,21]. Updating and maintaining the graphical content of a scene graph, however, requires the development of algorithms and methods specific to the platform on which the AR application is based. Such algorithms must be designed taking several parameters into consideration.

One such important parameter is geometric instancing which is defined in this dissertation as the ability of the visualization engine to construct complex scenes potentially consisting of a large number of CAD objects from a limited number of virtual models while maintaining performance levels. This is extremely important since most of the time, graphical hardware and CPU processing speed are the primary bottlenecks in creating and displaying animations at acceptable frame rates. Hence, an efficient approach is to create a complex scene consisting of a large number of virtual objects from as few individual CAD models as possible. Again, scene graphs are very effective tools to achieve this goal since all CAD objects in the scene that use the same geometrical representation can be conveniently created by creating instances of a single CAD file associated with a class of objects. As a result, the CAD file needs to be loaded only once by the application which saves a lot of memory and running time and speeds up the process of scene rendering.

4.8. References

[1] Kamat, V. R., and Martinez, J. C. (2002), "Scene Graph and Frame Update Algorithms for Smooth and Scalable 3D Visualization of Simulated Construction Operations", Journal of Computer-Aided Civil and Infrastructure Engineering 17(4), Blackwell Publishers, Malden, VA, 228-245.

[2] Zara, J., Chromy, P., Cizek, J., Ghais, K., Holub, M., Mikes, S., and Rajnoch, J. (2001), "A Scalable Approach to Visualization of Large Virtual Cities", In Proceedings of the 5th International Conference on Information Visualization, London, UK, 639-644.

[3] Kalra, P., Magnenat-Thalmann, N., Moccozet, L., Sannier, G., Aubel, A., and Thalmann, D. (1998), "Real-Time Animation of Realistic Virtual Humans", Journal of Computer Graphics and Applications, 18(5), IEEE, Piscataway, NJ, 42-56.

[4] Shreiner, D., Woo, M., Neider, J., and Davis, T. (2004), OpenGL Programming Guide, Addison Wesley, Reading, MA.

[5] Nadeau, D. R. (2000), "Volume Scene Graphs", In Proceedings of the Symposium on Volume Visualization, IEEE, Salt Lake City, UT, 49-56.

[6] Behzadan A. H., and Kamat V. R. (2007), "Enabling Smooth and Scalable Dynamic 3D Visualization of Discrete-Event Construction Simulations in Outdoor Augmented Reality", In Proceedings of the Winter Simulation Conference (WSC), Washington, DC, 2168-2176.

[7] Dollner, J., and Hinrichs, K. (1997), "Object-Oriented 3D Modeling, Animation, and Interaction", Journal of Visualization and Computer Animation, 8(1), John Wiley and Sons, New York, NY, 33-64.

[8] Selman, D. (2006), Java 3D Programming, Manning Publications Co., Greenwich, CT.

[9] Ames, A., Nadeau, D., and Moreland, J. (1997), VRML 2.0 Sourcebook, John Wiley and Sons, New York, NY.

[10] Lipman, R., and Reed, K. (2000), "Using VRML in Construction Industry Applications", In Proceedings of the Web3D-VRML2000 5th Symposium on Virtual Reality Modeling Language, Monterey, CA, 119-124.

[11] Benoit, M. (2000), Applied XML Solutions, SAMS Publishing, Indianapolis, IN.

[12] Walsh, A. E., and Bourges-Sevenier, M. (2001), Core Web3D, Prentice Hall, Upper Saddle River, NJ.

[13] OpenSceneGraph Website, Introduction to OpenSceneGraph – available at: http://www.openscenegraph.org/projects/osg/wiki/About/Introduction (accessed February 20, 2008).

[14] Woolford, D. (2003), "Understanding and Using Scene Graphs", available at: http://www.itee.uq.edu.au/~comp4201/scene_graphs_dsaw.pdf (accessed February 20, 2008).

[15] Behzadan, A. H., and Kamat, V. R. (2007), "Georeferenced Registration of Construction Graphics in Mobile Outdoor Augmented Reality", Journal of Computing in Civil Engineering, 21(4), American Society of Civil Engineers (ASCE), Reston, VA, 247-258.

[16] Kamat, V. R., and Behzadan, A. H. (2006), "GPS and 3DOF Angular Tracking for Georeferenced Registration of Construction Graphics in Outdoor Augmented Reality", In

Lecture Notes in Computer Science (Intelligent Computing in Engineering and Architecture), Ian F.C. Smith (Editor), Springer, New York, NY, 368-375.

[17] Behzadan, A. H., Timm, B. W., and Kamat, V. R. (2008), "General Purpose Modular Hardware and Software Framework for Mobile Outdoor Augmented Reality Applications in Engineering", Journal of Advanced Engineering Informatics, 22(1), Elsevier Science, New York, NY, 90-105.

[18] Behzadan A. H., and Kamat V. R. (2007), "Reusable Modular Software Interfaces for Outdoor Augmented Reality Applications in Engineering", International Workshop on Computing in Civil Engineering, American Society of Civil Engineers (ASCE), Pittsburgh, PA, 825-837.

[19] Bennett, P. (2006), National Marine Electronics Association – FAQ: Edge of Space Science – available at: http://www.eoss.org/pubs/nmeafaq.htm (accessed December 5, 2006).

[20] Assarsson, U., and Moller, T. (2000), "Optimized View Frustum Culling Algorithms for Bounding Boxes", Journal of Graphics Tools, 5(1), A. K. Peters Ltd., Wellesley, MA, 9-22.

[21] Silicon Graphics Inc. (1998), Cosmo 3D Programmer's Guide, Mountain View, CA.

Chapter 5

Real Time Visual Occlusion Handling in an Augmented Reality Animated Scene

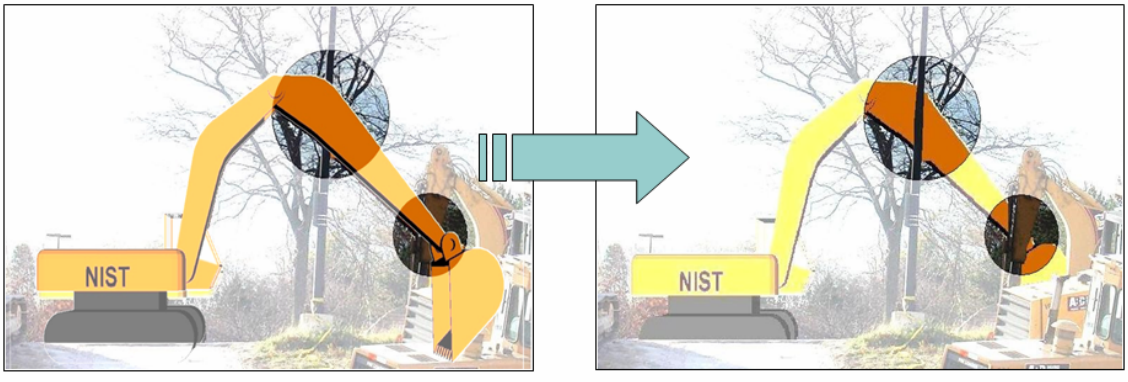
5.1. Introduction

The application of Augmented Reality (AR) in animating simulated construction operations has great potential in reducing the Model Engineering and data collection tasks, and at the same time can lead to more visually convincing output. As a tradeoff, however, the animation has to be capable of handling two distinct groups of objects: virtual and real. This is very essential in AR as the observer of the animation expects to see a mixed scene of seamlessly merged real and virtual objects in which both groups of objects operate and interact in a realistic manner. This introduces a number of challenges unique to creating AR animations. One of these challenges is interaction between objects. In a dynamic AR environment, interaction can be grouped into two main categories: visual and spatial.

Visual interaction between real and virtual objects is the result of reflection, absorption, and redirection of light emitted from and incident on the objects. Such effects include shadows, reflections, refraction, color bleeding, and occlusion. Spatial interaction between real and virtual objects include kinematic constraints, collision, response to external forces (e.g. deflection, bending). Kinematic interaction involves constraints or effects created by the motion of one object (real or virtual) on the position and orientation

of another object. In collision detection, a number of calculations are performed to determine when and where an object strikes another, thus preventing them from occupying the same physical space. The most complex type of interaction, however, occurs when there is an exchange of force and momentum between real and virtual objects. It is important to realize that most interactions are currently one-way (i.e. real objects can affect virtual objects, but virtual objects cannot typically affect real ones) [1].

The primary focus of this Chapter is to address the problem of visual occlusion between real and virtual objects in an augmented scene. Incorrect occlusion effect occurs when a real object blocks the observer's view of a virtual object. In a dynamic scene such as a construction operation, incorrect occlusion effect can occur very frequently and unexpectedly because the real and virtual resources and personnel can move freely with no constraints. Figure 5.1 shows a snapshot of an AR animation in which a virtual CAD model of an excavator is superimposed on the real scenes of the jobsite. In this Figure, two real objects (i.e. light pole, and the real excavator) are closer to the viewpoint and hence must partially block the virtual excavator at two spots (i.e. stick and shovel). However, the observer of the scene sees the snapshot in Figure 5.1(a) as opposed to the visually correct view shown in Figure 5.1(b) since occlusion cannot be automatically handled and corrected unless appropriate methods are designed and integrated into the AR application.



(a) **(b)**

Figure 5.1 – Example of Occlusion in an AR Scene

The fact that in all traditional AR applications, the real world is captured and displayed in the background while all virtual CAD objects are displayed on the foreground causes the final animation to be unable to show the correct occlusion effect since the two groups of objects are completely separate. As a result, automated real time occlusion handling becomes a critical step in animating dynamic simulation models in AR.

The properties of the AR display system (Subsection 3.4.2) influence the approach to correctly handling occlusion [2]. As shown in Table 5.1, in screen-based or projection-based environments such as the Cave Automatic Virtual Environment (CAVE) (which represent a purely virtual immersive environment), handling occlusion of virtual objects by real objects is relatively simple and straightforward since real objects are always between the display surface and the observer's eyes and therefore always occlude virtual objects. However, virtual objects cannot occlude real objects since virtual graphics are always shown on the background screens while all real objects are located between the observer and the virtual background. When using a Head Mounted Display (HMD) to observe the augmented scene (e.g. in this research), the display surface is always between the observer's eyes and real objects, and the virtual objects occlude real objects by default. As a result, additional steps are required to handle cases in which real objects must occlude virtual CAD objects.

Table 5.1 – Mechanisms for Handling Occlusion in Different Display Systems [1]

Display System	Virtual Occluding Real	Real Occluding Virtual
Screen-based or Back-projection (CAVE)	Impossible	Inherent
Semi-transparent HMD	Inherent	Semi-visible Software solvable
Video see-through HMD	Inherent	Software solvable

In the presented research, depth and frame buffer manipulation techniques were used to develop a new automated approach for handling occlusion correctly. The presented approach is unique since it can be easily integrated into a mobile AR platform (such as UM-AR-GPS-ROVER previously introduced in Chapter 3) which allows the observer of

the AR animation to walk freely in the scene looking at the ongoing operations from different positions. The presented method is capable of automatically resolving occlusion effects in real time to produce a visually convincing final output.

5.2. Prior Work in Occlusion Handling

Several researchers have demonstrated algorithms and methods to correctly handle occlusion effects in AR. For example, Fortin and Hebert [3] investigated a model-based and a depth-based approach. While the former is only suited for a static viewpoint and relies on a tracked bounding volume model within which the object's silhouette is carved, the latter makes it possible to change the viewpoint by exploiting a handheld stereo camera. There are some limitations to the depth-based approach mainly due to the performances of local stereo algorithms. If the texture of the object is too uniform, the dense stereo correspondence may not be possible or at least unreliable. In doubtful cases, such correspondences will be trimmed by filtering, leaving holes in the disparity map which can result in missing 3D information for some areas of the real objects.

In another research, Breen et al. [1] presented techniques for interactively performing occlusion and collision detection between static real objects and dynamic virtual objects in AR. They used computer vision algorithms to acquire data that model aspects of the real world in form of geometric models and depth maps. Lepetit and Berger [4] introduced a semi-automatic approach to resolve occlusion in AR systems. Using their approach, once the occluding objects have been segmented by hand in selected views called *key frames,* the occluding boundary is computed automatically in the intermediate views. In order to do that, the 3D reconstruction of the occluding boundary is achieved from the outlined silhouettes.

Fischer et al. [5] presented an algorithm based on a graphical model of static backgrounds in the natural surroundings, which has to be acquired beforehand. This algorithm is unable to deliver actual depth information for the scene. As a result, the main assumption was that whenever a real occluder is detected in front of the background, it is in front of

all virtual objects. Hence, their method is primarily suitable for interaction with the AR scenes using hands or pointing devices, which can mostly be assumed to be closer to the user than virtual objects.

Feng et al. [6] designed an optical-based algorithm to realize multilayer occlusion in indoor areas with objects only a few meters away from the viewer. The result of their work, however, caused unstable scenes as the process of object extraction was very sensitive to the change of ambient light illumination of the environment. Wloka and Anderson [7] presented a video see-through AR system capable of resolving occlusion between real and computer generated objects. The heart of their system was an algorithm that assigns depth values to each pixel in a pair of stereo video images in near real time. However, the use of stereo cameras caused their method to have difficulties in computing the depth for featureless (evenly lit, non-textured, and horizontal) rectangular image areas.

Figure 5.2 shows sample snapshots of previous work in occlusion handling in AR as presented in [1,3,4].

Figure 5.2 – Previous Work in Occlusion Handling in AR [1,3,4]

5.3. Main Contributions

Most work conducted in occlusion handling thus far, does not take into account the dynamics of the real world in which an AR animation takes place. While some of them use simplifying assumptions about the position of the real objects and the viewpoint from

which the scene is observed, several others use techniques that are most suitable for indoor controlled environments. For example, while the application of stereo cameras is an attractive option for real world depth acquisition, the result is largely dependent on the nature of the objects, their physical characteristics and appearance, and distances they are located from the observer of the scene.

Producing correct occlusion effects in real time in an outdoor unprepared environment such as a construction site was the main motivation for developing an automated occlusion handling method in this research. The attained results had to be convincing enough to the observer of the scene. At the same time, no additional constraints over the user's maneuvering ability as well as position and orientation of both groups of real and virtual groups of objects could be created in the AR application. In addition, the required hardware components had to be selected in a way that they do not limit the mobility of the AR platform due to factors such as heavy weight, dependence on ground power source, special care and maintenance, and user ergonomics.

Considering all these parameters, the final decision was made to develop a depth-based occlusion handling algorithm that would use real world depth map input coming through a remote sensing device such as a Laser Detection and Ranging (LADAR) camera connected alongside the video camera mounted on the observer of the scene. As discussed later in this Chapter, the main advantages of flash LADAR devices are their light weight, high data resolution, and ability to extract depth data in almost any environment type including outdoor construction sites. The limitations are constant noise in the incoming data, and limited operational range (on average less than 10 meters). However, the occlusion handling method introduced in the presented research has been designed as generic as possible so that future products with higher levels of accuracy and wider operational range can be easily plugged and used in the AR platform without any modifications or changes necessary to the core algorithms.

Thus, ***the main contribution of the research presented in this Chapter is a general purpose occlusion handling algorithm integrated into the core AR platform capable of***

detecting and correctly resolving occlusion effects in real time. This method requires no additional constraints over the position of real and virtual objects in the augmented scene and the user can also move freely inside the animation in real time.

5.4. Level of Detail in Depth Acquisition

As described earlier, obtaining depth values for real and virtual objects is the most important step in correctly handling occlusion effects in AR. An intuitive approach to calculate the depth of a virtual CAD object in the scene is to extract its position in the Z direction (positive direction pointing straight at the user) from the corresponding translation matrix of its scene graph node (Chapter 4) since the transformation matrices (translation, rotation, and scale) always maintain the latest geometric state (position, orientation, and scale) of an object. In fact, the graphical engine of the AR application (implemented in OpenGL) automatically keeps track of the depth of CAD objects relative to the viewpoint (user's eyes) and this depth value can be retrieved at any time using techniques that will be described later in this Section. The depth values for real objects, however, have to be acquired and recorded using more complex methods as they are not interpretably modeled in the computer. In other words, since the incoming video stream of the real world is captured by and displayed in a monoscopic visual system in this research, the computer's knowledge of a real scene is limited to the plain video captured by the video camera without any depth information.

The level of detail according to which the depth of real and virtual objects is determined depends on the characteristics of the environment the operation takes place in and also the nature of the objects in the scene. For a scene consisting of only a few simple geometrical primitives, measuring the depth values at the object level seems to provide satisfactory results. Based on this approach, the depth of an object is represented by the distance between the user's eyes and a single (or a few points) on that object. Selecting the most appropriate point on each object so that the depth is calculated as accurate as possible is a major challenge in this approach. Under some conditions, such as that shown in Figure 5.3, however, finding such a point becomes impossible and as a result, more

than one point has to be picked for depth calculations. In this Figure, a steel erection operation using a virtual tower crane to construct a virtual service building (i.e. Building **B**) in front of a real residential building (i.e. Building **A**) is shown. Following the object level of detail approach described above and based on the location from which the user is observing the scene, there are two possible positions to pick depth representative points on building **A** (i.e. points **x** and **y**). In addition, point **z** can be selected to determine the depth of the virtual building.

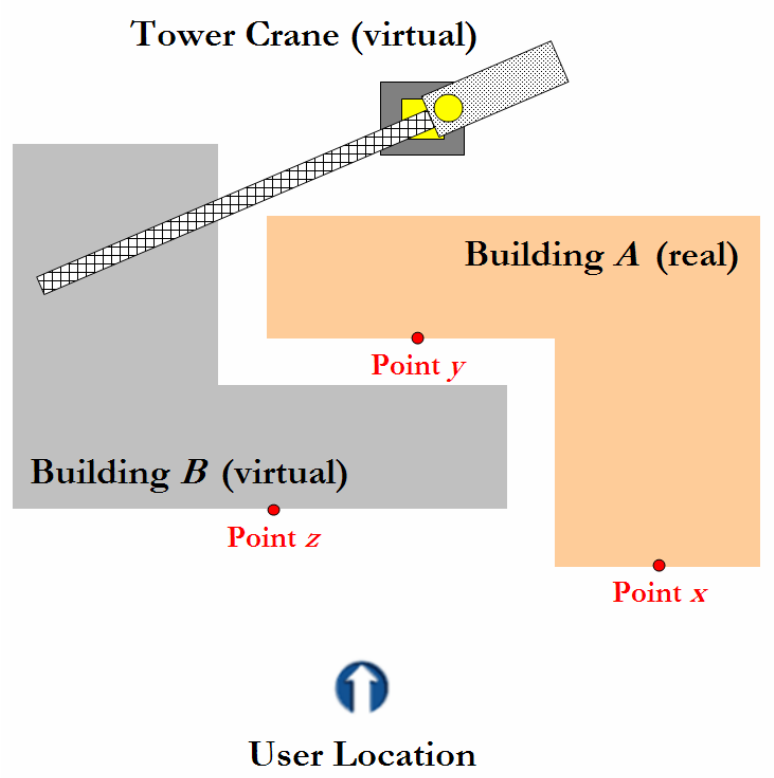
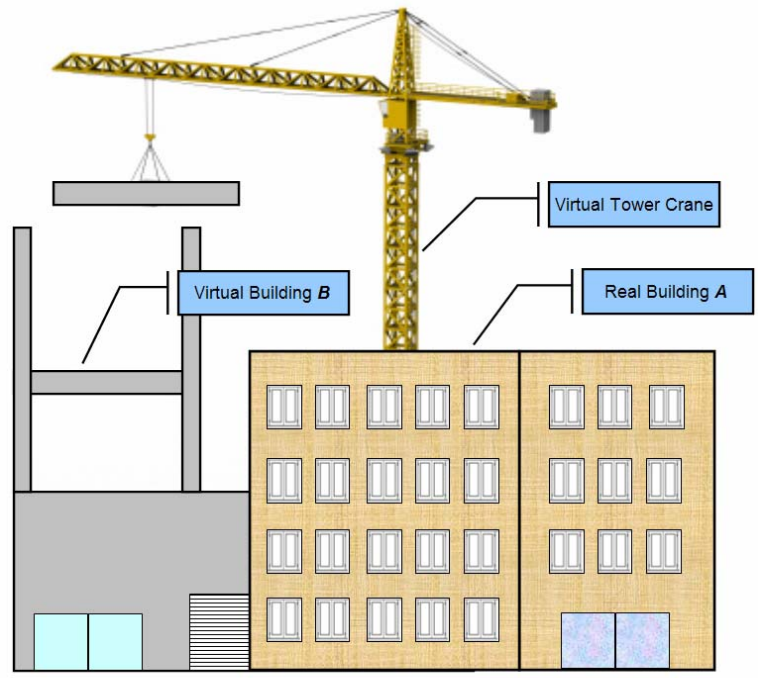


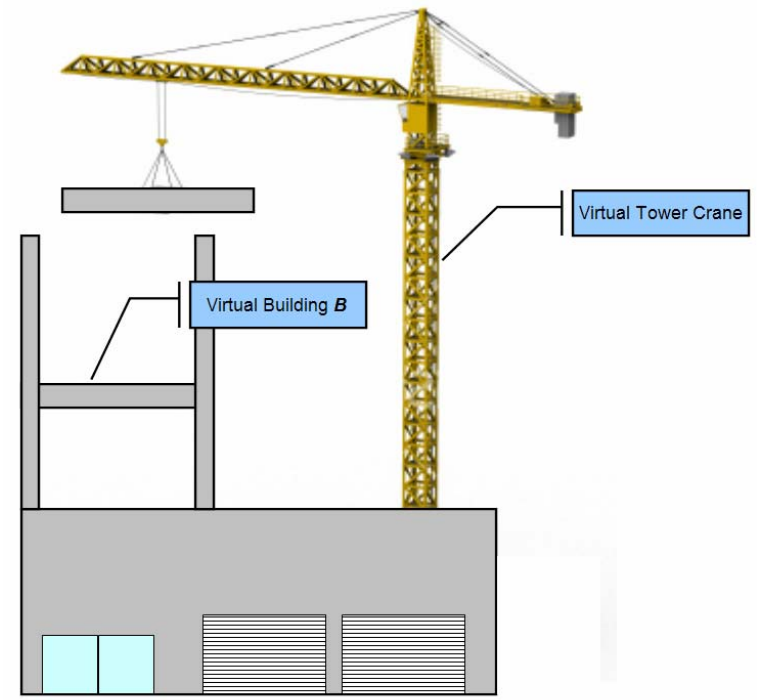
Figure 5.3 – Example of an Impossible Object Level of Detail Depth Calculation

Due to the way the scene is set up, choosing only one point on the real building (either point **x** or point **y**) is not enough for correct depth calculations. Having selected point **x** as the depth representative point for the real building **A**, since this point is closer to the user compared to point **z** (depth representative point for the virtual building), building **A** has to completely block building **B** in user's view of the augmented scene. If point **y** is selected to represent the depth of the real building, the virtual building will completely block the user's view of the real building as point **z** is closer to the user's eyes compared

to point **y**. As shown in Figure 5.4, none of these two cases are visually correct and convincing enough to the user of the augmented scene.



Real building incorrectly occluding virtual building



Virtual building incorrectly occluding real building

Figure 5.4 – Incorrect Occlusion Effects Using Object Level of Detail

For the specific case of Figure 5.3, simultaneous selection of point **x** and point **y** on the real building **A** will correctly resolve the occlusion effect. In fact, the geometry of building **A** has to be divided into two separate parts and the depth of each part has to be independently compared with the depth of building **B** in order to create a correct occlusion effect as shown in Figure 5.5.

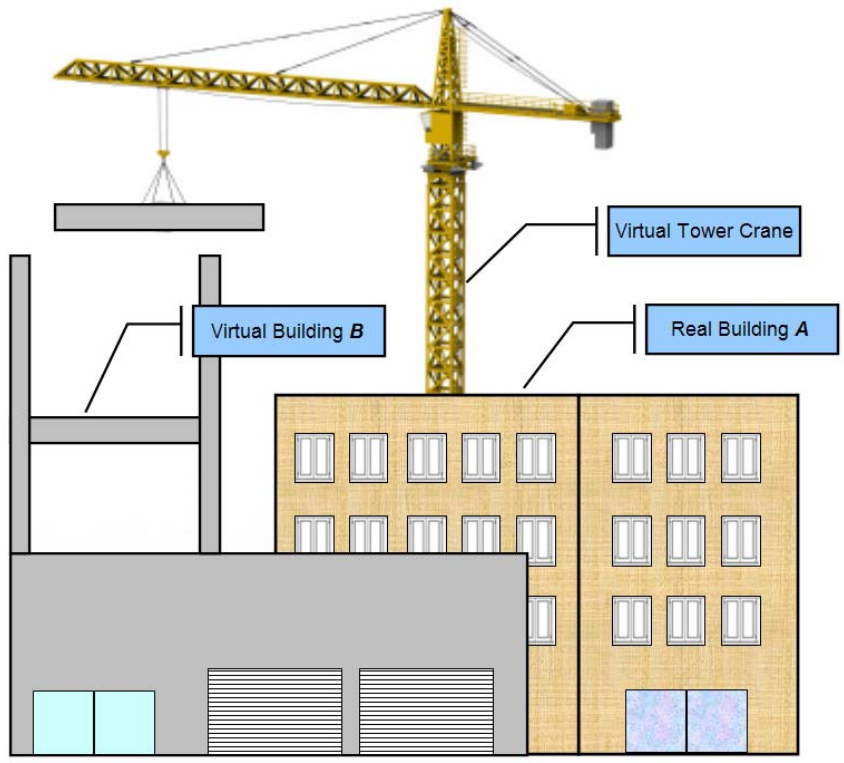


Figure 5.5 – Correct Occlusion Effect Using Multiple Depth Representative Points

In this Figure, parts of the real building (represented by point **y** in Figure 5.3) are blocked by the virtual building while the remaining part (represented by point **x** in Figure 5.3) is not occluded in the augmented view of the scene. Table 5.2 shows how the selection of different depth representative points will affect the final augmented view in terms of convincing occlusion effects. This example shows that objects operating in a dynamic environment such as a construction site can form a variety of scene setups. As a result, the process of point selection on all such objects becomes a significantly time consuming task. Even if all such points are selected before the animation starts, there is no guarantee that the selected points can best represent the real scene and avoid further confusion in the depth calculation stage in cases such as that shown in Figure 5.4.

Table 5.2 – Effect of Depth Representative Point Selection on Occlusion

	Real Building A		Virtual Building B	Occlusion Effect
Points Selected	***x***	***y***	***z***	
				N/A
	●		●	Incorrect
		●	●	Incorrect
	●	●	●	Correct

Another intuitive approach in resolving occlusion is to acquire and manipulate depth values at the polygon level of detail. Based on this approach, objects (real and virtual) are separated into smaller polygons. Rasterization algorithms are then applied to each pair of polygons to decide which has a lesser depth value and hence has to block the other. However, there are several cases in which it is impossible to make an accurate determination on which polygon is closer to the viewpoint. Figure 5.6 shows a scenario in which three polygons are to be placed in an animated scene. As shown in this Figure, polygon **A** is closer to the viewpoint compared to polygon **B**, and polygon **B** is closer than polygon **C**. Based on these two observations, a default conclusion is that polygon **A** has to be closer to the viewpoint than both polygons **B** and **C**, which is clearly not the case as shown in Figure 5.6.

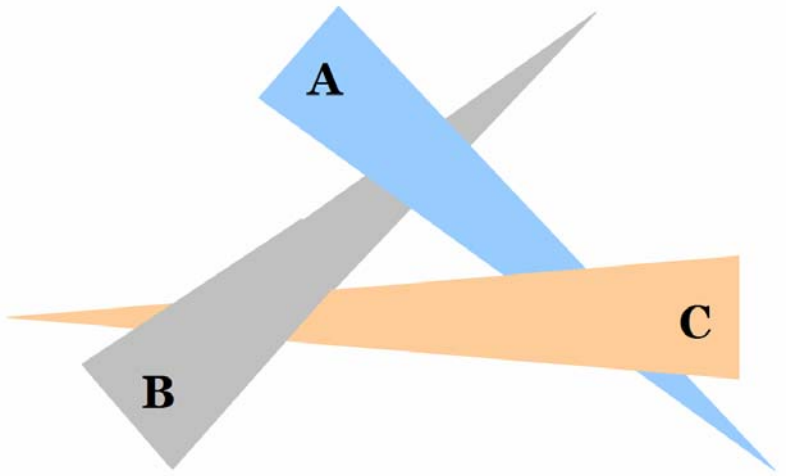


Figure 5.6 – Impossible Occlusion Handling Case Using Polygon Level of Detail

In order to take into account all such uncertainties in the way the real world is set up, and avoid any unexpected depth miscalculation and paradox, a pixel level of detail was

finally selected in this research. The depth acquisition and color manipulation algorithm designed in this research is shown in Figure 5.7.

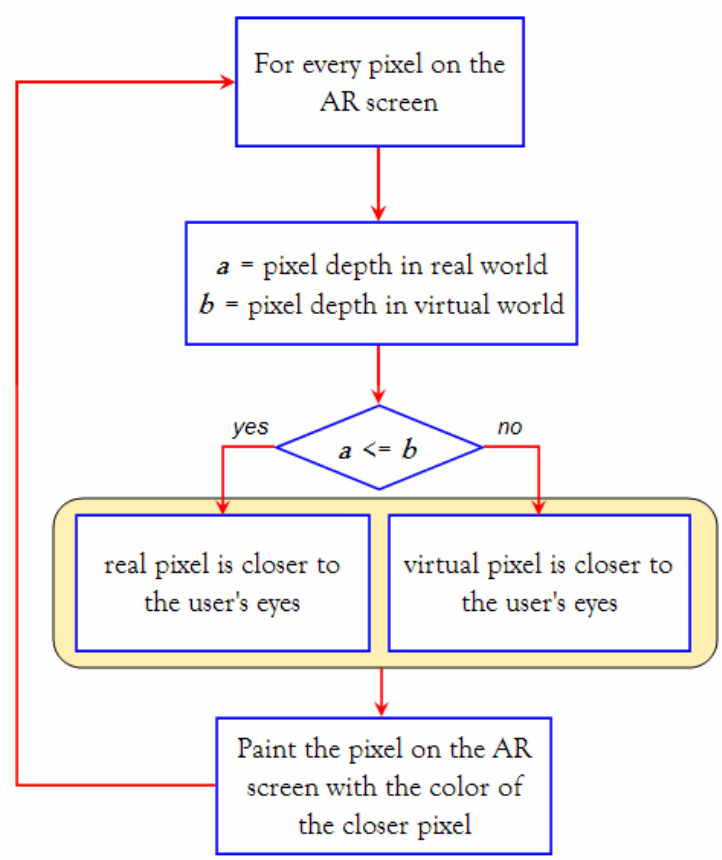


Figure 5.7 – Designed Depth Acquisition and Color Manipulation Algorithm

As shown in Figure 5.7, after the depth values for all real and virtual objects are obtained, the last step is to make a comparison at the pixel level to decide which object is closer to the user's eyes and hence has to be painted last. Following this approach, for a specific pixel on the screen, if the value of the real world depth is less than that of the virtual world, a real object is occluding a virtual object. Further steps have to be then taken in order to update the color scheme of that pixel so it is not painted in the color of the virtual object. These steps will be discussed in more detail in the next Section.

Figure 5.8 shows the graphical representation of the developed depth acquisition method for real and virtual objects.

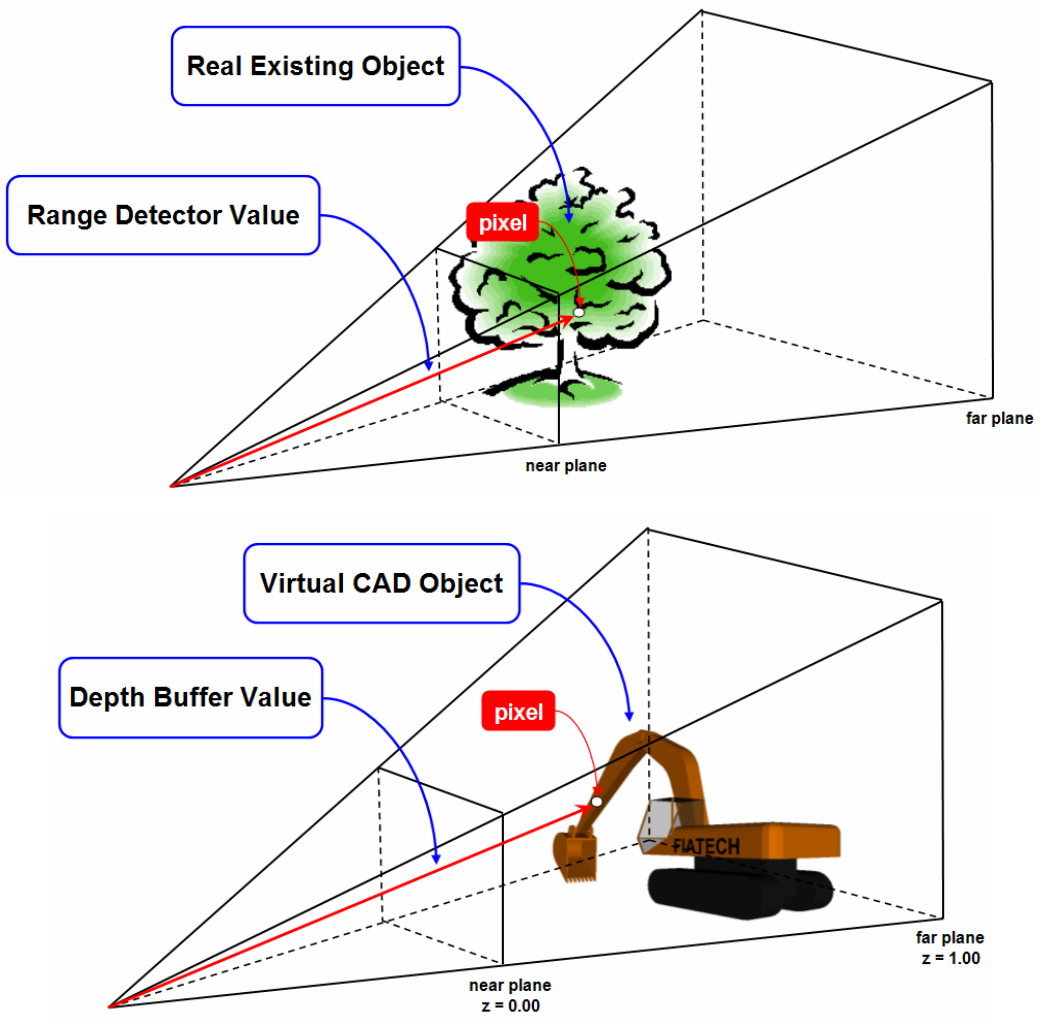


Figure 5.8 – Capturing the Depth of Real and Virtual Objects

The specific pixel on the screen as shown in Figure 5.8 represents portions of a real tree and a virtual CAD model of an excavator. The depth of this pixel in the real world is captured using methods that will be described later in this Section. The depth value for the same pixel in virtual world can be obtained using transformation matrices and geometric properties of the CAD model represented by the pixel.

Figure 5.9 illustrates the result of depth comparison performed at pixel level to determine whether or not the CAD object is occluded in the augmented view.

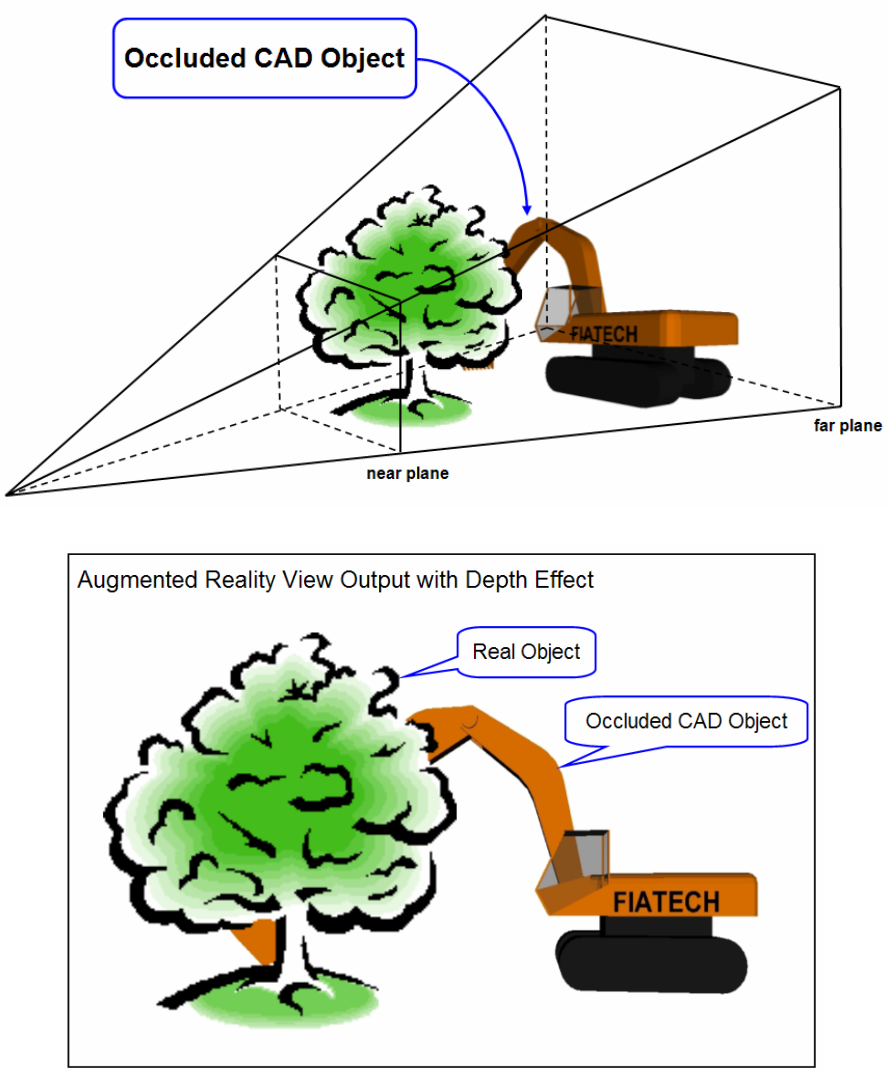


Figure 5.9 – Final AR Screen with Correct Occlusion Effect

The depth acquisition and color manipulation process depicted in Figures 5.7 through 5.9 has to be executed continuously by capturing the latest real and virtual depth values for all the pixels on the screen. Depth values of the real world can change if the viewpoint is moved (i.e. user changes position and/or head orientation) or there is a change in the contents of the real scene. Depth values of the virtual world can also change if CAD objects move in the scene. By constantly comparing these two sets of values, the depth effect can be included and represented in the augmented world to correctly resolve occlusion effects.

5.5. Depth Sensing Hardware Selection

Producing correct occlusion effects in real time in an outdoor unprepared environment such as a construction site was a main design consideration in developing the automated occlusion handling method in this research. As described in Chapter 4, the author has successfully designed and implemented a mobile computing apparatus equipped with components necessary to perform a walk-through AR animation in real time [8]. As shown in Figure 5.10, the apparatus takes advantage of real time positioning data coming through the Global Positioning System (GPS) receiver as well as 3D head orientation data supplied by the head orientation tracker device (inside the hard hat) to position the user inside the AR animation.

Figure 5.10 – Profile of a User Equipped with Mobile Computing Apparatus

In the mobile computing apparatus shown in Figure 5.10, the main computing task is performed by a laptop computer secured inside the backpack. While the real scene is captured by a video camera in front of the user's eyes, the rendered graphical scene is

displayed to the user through the HMD installed in front of the hard hat. At the same time, the user can interact with the system using a miniature keyboard and a touchpad. While the resulting AR animation has to be convincing enough to the observer of the scene, no additional constraint over the user's maneuvering ability as well as position and orientation of both groups of real and virtual objects has to be imposed by the AR application. In addition, the required depth sensing hardware components to perform the task of occlusion handling have to be selected in a way that they do not limit the mobility of the AR platform due to factors such as heavy weight, dependence on ground power source, special care and maintenance, and user ergonomics.

As noted earlier, depth acquisition at pixel level of detail has the advantage that the scene can be arbitrarily complex, while the processing time remains a constant time function of image resolution. Additionally, no geometric model of the real environment is needed during the animation. However, the depth map is dependent on the user's position and head orientation, as well as the location of real objects in the scene. Once the user or the real objects change their position and/or orientation, the depth map becomes invalid [1].

In order to take into account all such variations, the process of depth acquisition and comparison has to be done in real time. As a result, developing methods to sense and prototype objects of the real world was a significant step in this research. The hardware component required to perform depth acquisition had to be selected in a way that it could be easily integrated into any existing mobile AR platform. Hence, being lightweight and self-powered, having a convenient interface, and supporting an acceptable pixel resolution were among the important factors in selecting the hardware component. There was certainly a tradeoff between equipment mobility and data resolution. The heavier the camera is, the more data sample points it can collect and the resulting image is more accurate. This directly translates into more processing time which was not desirable for the purpose of this research as all calculations had to be performed in real time. Lighter cameras, although providing lower resolution depth images, can operate at a faster processing speed and hence were better fits for the AR platform of this research.

Several options were studied including high resolution cameras such as 3DLS (7 to 8 kg), Konica Vivid 9i (15 kg), I-Site 4400 LR (14 kg), FARO LS 420 (14.5 kg), and Leica HDS 3000 (17 kg). Although all these cameras provide full range data (wide horizontal and vertical scanning angles) they cannot be mounted on mobile platforms mainly due to their weight and dependence on external power sources. In addition, according to most manufacturers, fixed tripods have to be used as mounting bases in order to achieve best performance. Another category of imaging cameras is light weight cameras such as flash LADAR devices that are typically suitable for low range applications. As discussed in Section 5.3, this category of cameras was finally selected for this research to perform real world depth acquisition as it is a promising technology providing robust and accurate access to depth data of the real objects, and has already proven to provide satisfactory results in geometric modeling of construction sites for automation and robotics applications [9,10,11]. Table 5.3 shows some details of three such cameras that are most used in relevant research projects.

Table 5.3 – Manufacturer's Properties of Different Flash LADAR Devices

Model	Dimension (cm)			Pixel Resolution		Field of View (°)		Frame Rate	Range (m)
	X	y	z	H	V	H	V		
CSEM SR3000	5.00	6.70	4.23	176	144	39.6	47.5	50	7.5
CSEM SwissRanger2	14.60	3.10	3.30	160	124	42.0	46.0	30	7.5
PMD 19K	20.80	17.40	4.40	160	120	40.0	30.0	15	5.0 ~ 30.0

A flash LADAR system typically consists of a device constantly casting laser beams in the 3D real space and receiving the resulting reflected beams. Based on the travel time for each beam and knowing the speed of the laser beam, the distance between the flash LADAR system and each real object in the scene is calculated. Once installed in front of the user's eyes, these depth values reflect the distance between the viewpoint and the real objects. Figure 5.11 shows several flash LADAR devices studied in this research.

(a) CSEM SwissRanger2 **(b) CSEM SR3000** **(c) PKD 19K**

Figure 5.11 – Examples of Flash LADAR Devices Studied in This Research

Flash LADAR data represent a scene as a matrix. Each element in this matrix contains the depth value of the corresponding pixel on the screen. As a result, the concept of *screen matrix* was introduced and used in this research to provide a means to store and retrieve depth values more efficiently inside the AR platform. By definition, a screen matrix is a 2D matrix with dimensions equal to the pixel resolution of the screen. Each element in this matrix can hold data (e.g. RGB color, depth value) about the corresponding pixel on the screen.

Figure 5.12 shows a sample screen matrix for a 640x480 screen. As described earlier, the elements in this matrix can store depth values as well as color codes for their corresponding pixels on the screen. If the resolution of the depth data obtained from a flash LADAR device is less than the resolution of the actual screen, adjacent elements of the screen matrix can be clustered together and a single depth value (obtained from the flash LADAR device) can be assigned to all the elements inside a cluster. For example, if the incoming flash LADAR depth data has a resolution of 160x120 while the actual screen resolution is 640x480, four adjacent pixels (in both horizontal and vertical directions) can be grouped as one cluster and the same depth value can be assigned to all of them. Depth values for the pixels on the virtual screen are also retrieved from the OpenGL z-buffer. The z-buffer is in fact a memory buffer in the OpenGL graphics accelerator that holds the latest depth of each pixel along the Z axis. As the virtual

contents of the scene change over time, the values stored in the z-buffer are also updated to reflect the latest depth of the CAD objects relative to the viewpoint (i.e. user's eyes).

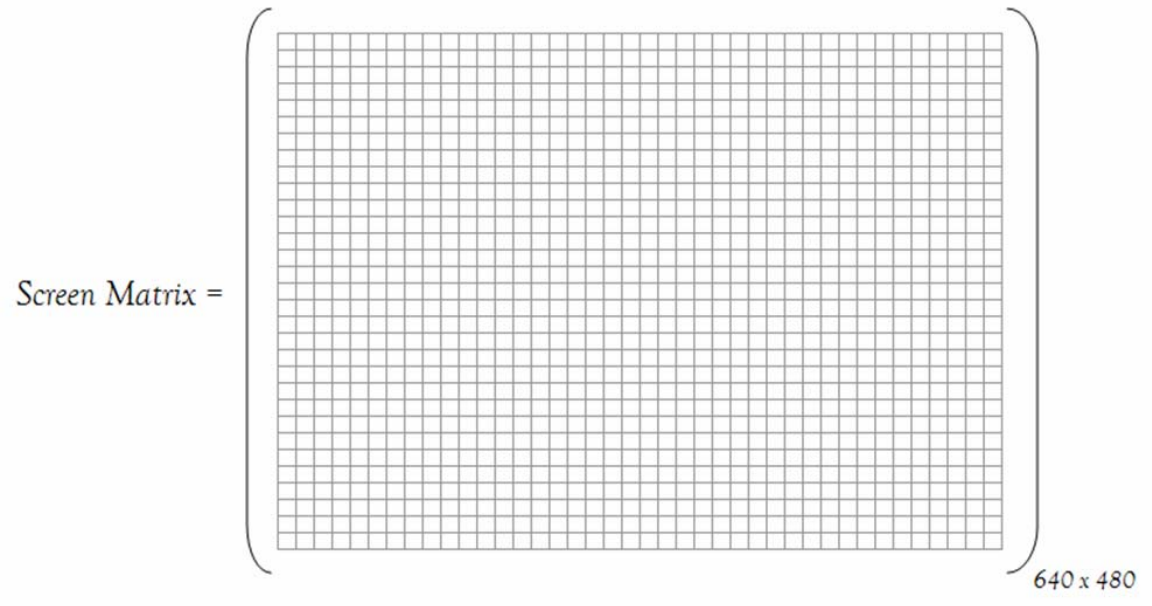


Figure 5.12 – Sample Screen Matrix for a 640 by 480 Screen

The fundamental difference between depth values obtained from a flash LADAR camera for real objects and those obtained from z-buffer for virtual objects is that while real world depth values are retrieved and reported as real distances (in terms of meters or feet) to the user, depth values for the virtual CAD objects fall between a [0,1] interval. A pixel located on the near plane of the perspective viewing frustum will be given a depth value equal to zero and a pixel located on the far plane of the perspective viewing frustum will be given a depth value equal to one. All intermediate pixels will have depth values between zero and one. However, the relationship between the depth values obtained from the z-buffer and corresponding metric depth values is not linear. A pixel with a virtual depth value of 0.5 is not located halfway between the near and far planes. In fact, this relationship, as shown in Figure 5.13, follows a hyperbolic equation [3]. In this Figure, Z_{near} and Z_{far} correspond to the metric distance between the user's eyes and the near and far planes of the perspective viewing frustum respectively. Z_{buffer} is the depth value of a specific pixel on the screen obtained from the z-buffer and Z_{real} is the metric equivalent of this depth value for the same pixel. As shown in Figure 5.13, z-buffer has higher

resolution for pixels closer to the user's eyes. For the specific case shown in Figure 5.13, more than 90% of all possible z-buffer values represent the depth of objects located within 10% of the distance between the near and far planes. This is mainly due to the fact that the human eyes are more sensitive to closer objects and can identify any short range visual discrepancies more rapidly, whereas discrepancies in objects that are farther away cannot be recognized as clearly as in closer objects. Depth values for the virtual objects are also stored in a separate screen matrix for later comparison with corresponding values of the real world.

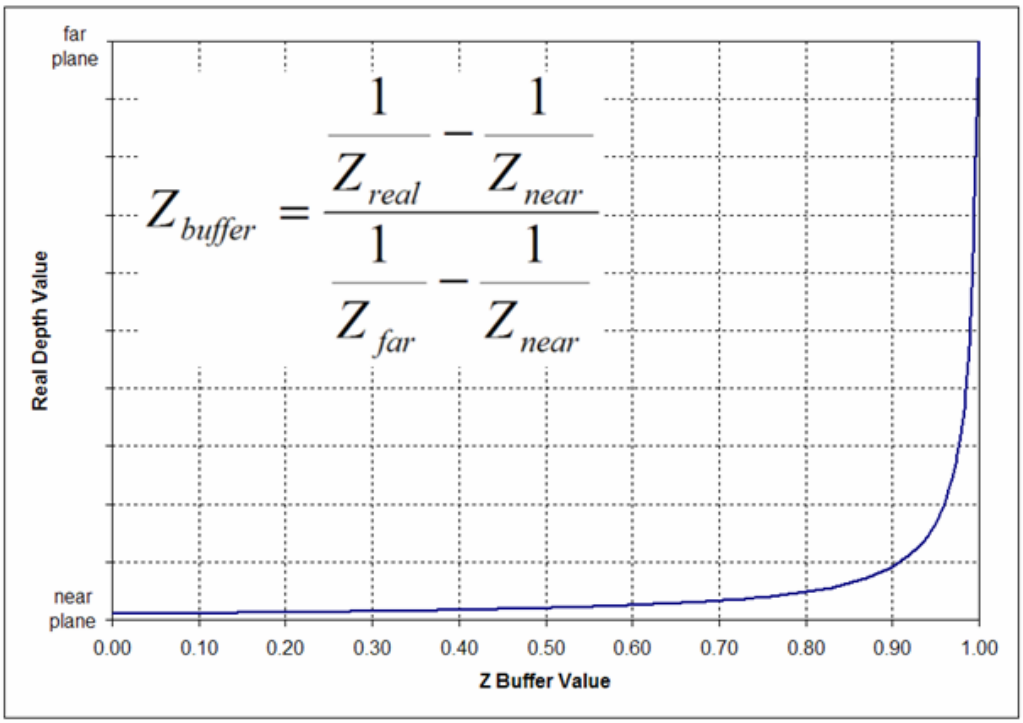


Figure 5.13 – Relation between Z-Buffer and Metric Virtual Depth Values

5.6. Frame Buffer Manipulation

After all depth values are obtained and stored appropriately in separate screen matrices, they have to be compared so that for each pixel a final decision is made on which group of objects (real or virtual) is closer and hence has to be displayed. Once this decision is made, the color of the pixel will be changed to the color of the closer object to the

viewpoint to create the impression that the pixel really represents the correct object. This requires an intermediate step which is obtaining the color values (in terms of RGB) for each individual pixel on the screen. This can be done by directly by reading OpenGL color buffer which stores pixel color values in real time. Two distinct readings are done to obtain pixel colors of the scene with and without virtual objects. Once depth and color readings are complete, four different matrices are provided to the AR application:

- **A** = Screen matrix of real world color values (captured texture from the video camera as stored in OpenGL color buffer before superimposition of CAD models)
- **B** = Screen matrix of real world depth values (LADAR camera data representing the depth of the raw video input coming through the video camera)
- **C** = Screen matrix of CAD models' color values (OpenGL color buffer after superimposition of CAD models)
- **D** = Screen matrix of CAD models' depth values (OpenGL depth buffer after superimposition of CAD models)

Figure 5.14 shows how these four matrices are constructed using the contents of depth and color buffers as well as the incoming data through the flash LADAR device. As shown in this Figure, starting from the top left hand corner, after the OpenGL frame buffer is refreshed, the content of the video frame is updated using the raw video image coming through the video camera. At this time, matrix **A** is constructed using pixel color values of the raw video image and matrix **B** is constructed using the depth data of the real world coming through the flash LADAR device. Once the OpenGL frame is updated, the virtual contents of the scene are displayed on top of the real background. At this point, matrix **C** is constructed using the contents of the OpenGL color buffer and matrix **D** is constructed using the OpenGL z-buffer values. Note that all these operations occur at each animation frame to handle occlusion cases continuously and as the animation is running.

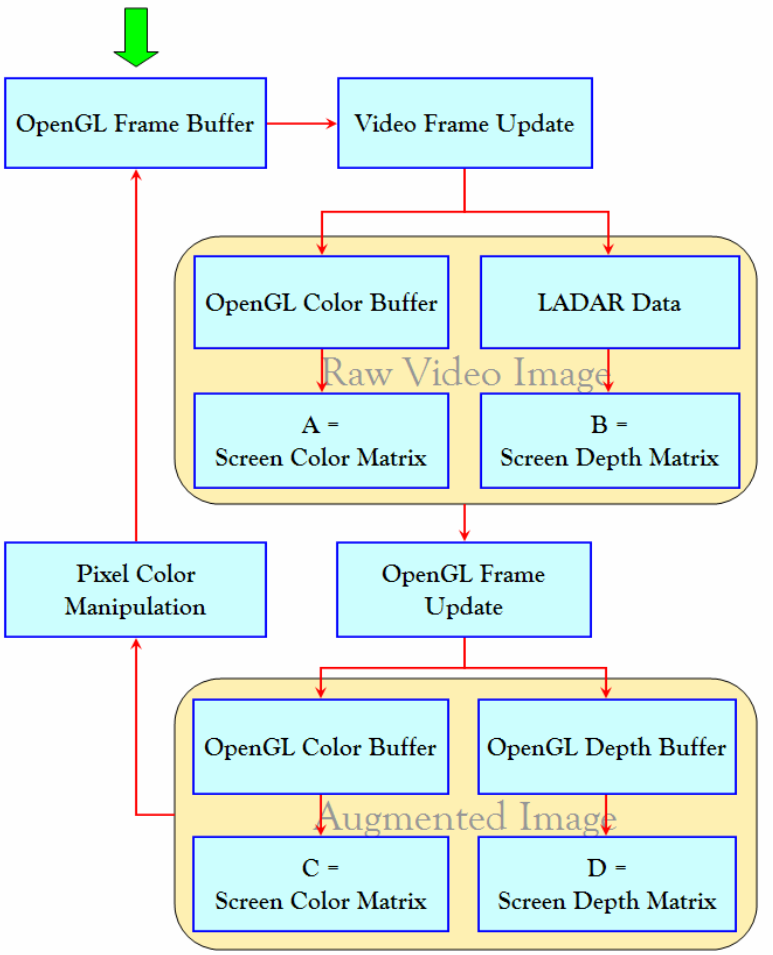


Figure 5.14 – Constructing Four Distinct Color and Depth Matrices

Figure 5.15 shows the frame buffer manipulation algorithm designed in the presented research which uses these four matrices in order to construct a final screen matrix (i.e. matrix **E** in this Figure). Screen matrix **E** contains correct pixel colors after resolving all incorrect occlusion cases. In this Figure, for every pixel on the screen, the corresponding color and depth value is read from the four matrices described above (shown as ***a***, ***b***, ***c***, and ***d*** in Figure 5.15). The real and virtual depth values are than compared. If the depth of the pixel in the real world is less than its depth in the virtual world, its color is changed to the color read from the matrix representing real world colors (i.e. matrix **A**). This represents the case in which a real object is occluding a virtual object. The other situation occurs when the depth of the pixel in the virtual world is less than its depth in the real world. This represents the case in which a virtual object is occluding a real object and hence its color is changed to the color read from the matrix representing virtual world

colors (i.e. matrix **C**). The correct pixel color (shown as ***e*** in Figure 5.15) is then stored in the screen matrix **E** which will be later used when the OpenGL frame buffer is updated to show the correct occlusion effect.

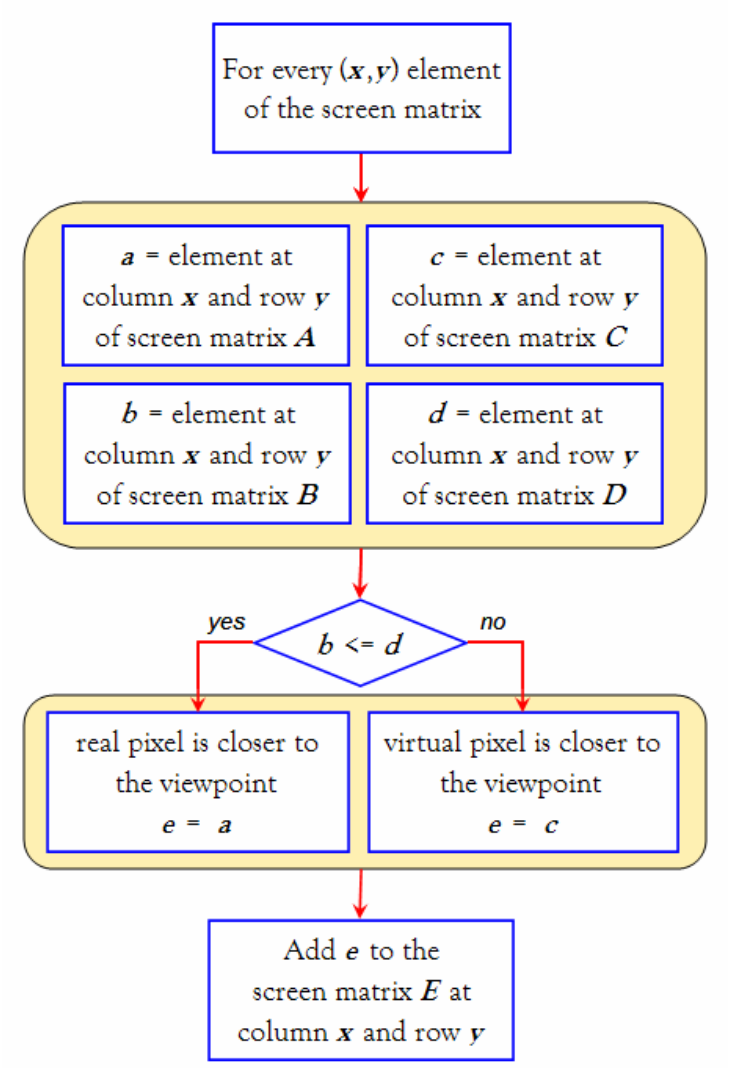


Figure 5.15 – Designed Frame Buffer Manipulation Algorithm

5.7. Summary and Conclusions

3D visualization of construction projects has gained a lot of credibility over the past few years. Several researchers have investigated the application of Virtual Reality (VR) to animate simulated construction operations in order to verify and validate the results of the

underlying simulation model [12,13,14,15,16,17]. In order to create realistic VR displays of a simulated process, detailed data about the process as well as the environment in which it takes place has to be obtained. Such data must be able to describe the simulation, 3D CAD models, facility under construction, and terrain topography (Model Engineering). As the size and complexity of the operation increases, data collection also becomes time and resource consuming. This directly translates into loss of project financial and human resources which could otherwise be saved and used more productively. In an effort to remedy this situation, the author has successfully introduced and developed an alternate approach to create dynamic animations of simulated operations. Following this approach, AR is used to create mixed views of real existing facilities on the jobsite and virtual CAD objects under construction.

The application of AR in animating simulated construction operations has great potential in reducing the Model Engineering and data collection tasks and at the same time leads to a more visually convincing output. As a tradeoff, however, the animation has to be capable of handling two distinct groups of objects: virtual and real. This is very essential in AR as the observer of the animation expects to see a mixed scene in which both groups of objects operate and interact in a realistic manner. This introduces a number of challenges unique to creating AR animations. One of these challenges is occlusion handling. Incorrect occlusion effect occurs when a real object blocks the user's view of a virtual object. In almost all AR-based animations, the real world serves as the background of the AR scene and it has to be "drawn" before the computer generated contents are added to the foreground. As a result, although a real object may be closer to the observer, it can be visually blocked by a CAD object that is intended to be farther away. The main reason for this is that there is no established method in AR to obtain the depth of real objects and as a result, incorrect occlusion effects cannot be automatically handled.

In order to take into account all the complexities and uncertainties involved in a dynamic simulated process such as a construction operation, the method described in this Chapter uses depth and frame buffer manipulation techniques at the pixel level of detail to

automatically detect and correctly resolve visual occlusion cases in dynamic augmented environments. The developed approach is unique since it can be easily integrated into a mobile platform which allows the observer of the AR animation to walk freely in the scene looking at the ongoing operations from different perspectives. The presented method is capable of automatically resolving occlusion cases in real time to produce a visually convincing final output.

5.8. References

[1] Breen, D. E., Rose, E., and Whitaker, R. T. (1995), “Interactive Occlusion and Collision of Real and Virtual Objects in Augmented Reality”, Technical Report ECRC-95-02, European Computer-Industry Research Center.

[2] Fuhrmann, A., Hesina, G., Faure, F., and Gervautz, M. (1999), “Occlusion in Collaborative Augmented Environments”, Journal of Computers and Graphics, 23(6), Elsevier Science, New York, NY, 809-819.

[3] Fortin, P. A., and Hebert, P. (2006), “Handling Occlusions in Real-Time Augmented Reality: Dealing with Movable Real and Virtual Objects”, In Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV’06), Quebec City, QB, Canada, 54.

[4] Lepetit, V., and Berger, M. O. (2000), “A Semi-Automatic Method for Resolving Occlusion in Augmented Reality”, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Hilton Head Island, SC, 2225–2230.

[5] Fischer, J., Regenbrecht, H., and Baratoff, G. (2003), “Detecting Dynamic Occlusion in Front of Static Backgrounds for AR Scenes”, In Proceedings of the Workshop on Virtual Environments, Zurich, Switzerland, 153-161.

[6] Feng, Y., Du, W., Guan, X., Gao, F., Chen, Y. (2006), “Realization of Multilayer Occlusion between Real and Virtual Scenes in Augmented Reality”, In Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design, Nanjing, China, 1-5.

[7] Wloka, M. M., and Anderson, B. G. (1995), “Resolving Occlusion in Augmented Reality”, In Proceedings of the Symposium on Interactive 3D Graphics, Monterey CA, 5-12.

[8] Behzadan, A. H., Timm, B. W., and Kamat, V. R. (2008), “General Purpose Modular Hardware and Software Framework for Mobile Outdoor Augmented Reality Applications in Engineering”, Journal of Advanced Engineering Informatics, 22(1), Elsevier Science, New York, NY, 90-105.

[9] Teizer, J., Caldas, C. H., and Haas, C. (2007), “Real-Time Three-Dimensional Occupancy Grid Modeling for the Detection and Tracking of Construction Resources”, Journal of Construction Engineering and Management, 133(11), American Society of Civil Engineers (ASCE), Reston, VA, 880-888.

[10] Teizer, J., Liapi, K., Caldas, C. and Haas, C. (2005), “Experiments in Real-Time Spatial Data Acquisition for Obstacle Detection”, In Proceedings of the Construction Research Congress (CRC), San Diego, CA, 107-116.

[11] Teizer, J., Kim, C., Bosche, F., Caldas, C. H., and Haas, C. T. (2005), “Real-Time 3D Modeling for Accelerated and Safer Construction Using Emerging Technology”, In Proceedings of the 1st International Conference on Construction Engineering and Management, Seoul, Korea, 539-543.

[12] Op den Bosch, A. (1994), “Design/Construction Process Simulation in Real-Time Object-Oriented Environments”, PhD Dissertation, Georgia Institute of Technology, Atlanta, GA.

[13] Barnes, M. R. (1997), “An Introduction to QUEST”, In Proceedings of Winter Simulation Conference (WSC), IEEE, Atlanta, GA, 619-623.

[14] Bishop, J. L., and Balci, O. (1990), “General Purpose Visual Simulation System: A Functional Description”, In Proceedings of the Winter Simulation Conference (WSC), IEEE, New Orleans, LA, 504-512.

[15] Rohrer, M. W. (2000), “Seeing Is Believing: The Importance of Visualization in Manufacturing Simulation”, In Proceedings of the Winter Simulation Conference (WSC), IEEE, Orlando, FL, 1211-1216.

[16] Rohrer, M. W., and McGregor, I. W. (2002), “Simulating Reality Using AUTOMOD”, In Proceedings of the Winter Simulation Conference (WSC), IEEE, San Diego, CA, 173-181.

[17] Kamat, V. R. (2003), “VITASCOPE: Extensible and Scalable 3D Visualization of Simulated Construction Operations”, PhD Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Chapter 6

Validation of ARVISCOPE Animation Language

6.1. Introduction

The information presented in this Chapter presents the validation exercises for each of the research challenges described in individual Chapters of this dissertation that each focus on a specific aspect of the research. In addition to validating each individual aspect of the research during its progress, the ability of the ARVISCOPE animation authoring language to create smooth and continuous Augmented Reality (AR) animations of simulated construction processes was also validated at several stages of the work. In particular, the designed language and the implemented software and hardware framework were critically evaluated using the following criteria:

- Geometric and spatial accuracy in animations
- Accuracy in 3D spatial user tracking
- Ability of the AR framework to support geometric instancing
- Accuracy in visual occlusion handling

Discrete Event Simulation (DES) models of several simulated construction operations were used to automatically generate ARVISCOPE animation trace files as they ran. Although the execution of a DES model typically takes a few seconds, the generated animation trace file can contain several thousand lines depending on the duration of the

operation, the number of activities to be performed, and the level of detail desired in animating each activity in ARVISCOPE. As described in Chapter 2, the resulting trace file is a sequence of ARVISCOPE language statements ordered by their time of execution. The trace files were then used to create AR animations in order to investigate the extent to which the simulated (and communicated) operations can be recreated in the 3D augmented environment of the construction site. Each of the following Subsections describe the details of an outdoor test performed using the UM-AR-GPS-ROVER mobile computing apparatus (Chapter 3) and the ARVISCOPE animation authoring language (Chapter 2).

The degree of accuracy in animations was observed to be a function of the amount of detail communicated by the driving process (i.e. the DES models in this research), and the level of precision of the tracking devices. The fact that the entire content of an animation trace file is created by an external process (i.e. a running DES model) enables animated sensitivity analysis for each case simply by manipulating the quantitative and/or logical simulation parameters of the driving process model. These parameters include the number and type of resources (e.g. number of trucks in an earthmoving operation, type of crane to used in a steel erection, space for temporary storage of materials), the rules under which the different tasks that compose the operations are performed, and random variables to describe the duration of individual tasks. The modified models can be promptly rerun to produce alternate animated scenarios that can then be processed and visualized in AR by ARVISCOPE for evaluation.

6.2. Validation of Geometric and Spatial Accuracy in Animations: Offshore Concrete Delivery Operation

The ARVISCOPE animation language was able to accurately describe and depict graphical 3D representations of the communicated operations, and superimpose them on top of live video streams of the surrounding environment. Each of the conducted validation experiments was designed carefully considering several factors such as

operational logic, test location, and CAD object selection and manipulation. First, a sketch of each test was created showing different resources used in the operation together with their maneuvering range and movement directions. STROBOSCOPE was then used to create a DES input file of the operation. STROBOSCOPE is a programmable and extensible simulation system designed for modeling complex construction operations in detail and for the development of special purpose simulation tools [1]. The equivalent ARVISCOPE animation trace file was generated automatically by instrumenting the DES model using ARVISCOPE scene construction, dynamic, and control statements (Chapter 2 and Appendix A). The animation was created in a 640x480 resolution screen using a refresh rate of 30 frames per second.

In order to ensure geometric and spatial accuracy in the animations of modeled operations, the output of ARVISCOPE at each stage was compared side by side to a corresponding 3D animation of the same operation generated in Virtual Reality (VR) using the VITASCOPE visualization system. VITASCOPE is a user extensible 3D animation language designed specifically for visualizing simulated construction operations in smooth, continuous, animated 3D virtual worlds [2]. The same instrumentation procedure was used to create VITASCOPE animation trace files using its own language statements. Each resulting trace file was then loaded and run inside the VITASCOPE environment and several snapshots were taken at the exact simulation time instants as those of ARVISCOPE snapshots. This facilitated side by side comparison of the two sets of animations (i.e. VR based and AR based) for each validation scenario.

Figure 6.1 shows the aerial view of an outdoor experiment conducted in this research. The objective of this experiment was to create an animation of an offshore concrete delivery operation. The animation invovled two 3D virtual barges, each carrying a virtual concrete truck from a virtual batch plant on the shore to an offshore pier located in the middle of a river. Real time views of the Huron river in Ann Arbor, Michigan were used for the real backgound. Each barge spent a certain amount of time on the shore while its concrete truck was loaded at the batch plant. It then traveled to the offshore pier to unload

the concrete. The empty concrete truck then returned to the shore on the barge. This cycle was repeated several times before the concrete placement operation ended.

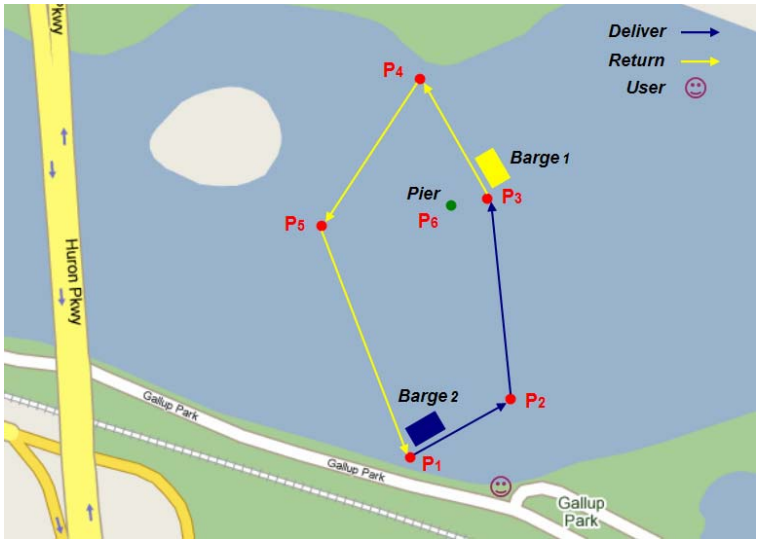


Figure 6.1 – Aerial View of the Offshore Concrete Delivery Operation Experiment

The global coordinates of points P_1 through P_6 in Figure 6.1 were carefully measured in terms of longitude, latitude, and altitude and used inside the corresponding animation trace file to define delivery and returning routes as well as the location of the offshore pier. The following assumptions were made to create the AR animation of this simulated operation:

(1) The actual concrete loading operation of the empty concrete truck was not animated on the batch plant located on the shore. To account for this part of the process, each barge waited a certain amount of time at the batch plant station.

(2) The actual concrete placement operation from the full concrete truck was not animated on the offshore location. To account for this part of the process, each barge waited a certain amount of time at the pier station.

Figure 6.2 shows the timeline of the AR animation. Each barge waited for 5 time units at the batch plant station, traveled for 15 time units to the offshore pier, waited for 5 time units at the offshore location, and returned to the batch plant station in 10 time units. Note that since this is a cyclic operation, the entire process repeated every 40 time units. In this experiment, all durations were deterministically set.

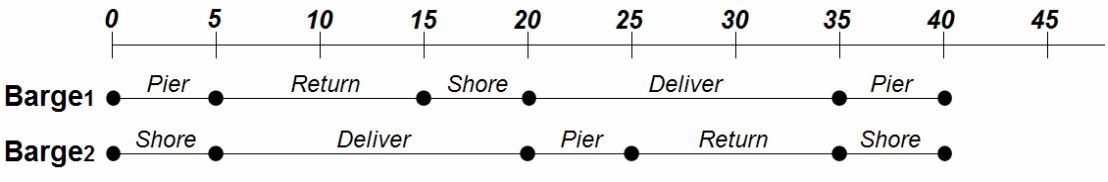


Figure 6.2 – Timeline of the Offshore Concrete Delivery Operation Experiment

Figure 6.3 illustrates the AR animation created in ARVISCOPE compared to the VR animation of the same operation created using VITASCOPE, and shows that the two outputs are identical in terms of geometric and spatial accuracy of the animation.

Figure 6.3 – Animated Offshore Concrete Delivery in ARVISCOPE and VITASCOPE

6.3. Validation of Accuracy in 3D Spatial Tracking: Earthmoving Operation

In order to validate the ability of the AR application to update the contents of the augmented animation based on the user's latest position and head orientation data, the user was allowed to navigate in the animation, approach the CAD objects, and change the head orientation. The objective of this part of the validation exercise was to ascertain that the tracking devices are fully capable of obtaining the user's real time position and head orientation, and that the application is robust enough to update the contents of the user's augmented viewing frustum based on these values. Continuous communication with the Global Positioning System (GPS) and head orientation tracking devices to obtain real time Six Degree-of-Freedom (6DOF) spatial data of the user was very critical since the augmented viewing frustum had to be constantly reconstructed in front of the user's eyes during the course of an animation. The GPS position measurements were obtained using the Wide Area Augmentation System (WAAS) service provided at no cost to the GPS users. Although the incoming GPS data inherited some error due to parameters such as the clarity of lines of sight to the GPS satellites, whether conditions, and existing ambient noise, the positional error level stayed constant and stable over the area in which the experiments were conducted as long as the number of visible GPS satellites was fixed. In order to achieve more precise GPS data, more precise technologies such as Differential GPS (DGPS) and Real Time Kinematics (RTK) can also be used. As discussed in Chapter 3, the tracking algorithms developed in this research, have been designed as generic as possible so that they are ready without modification to be used with DGPS, RTK or any other GPS data extraction technology [3,4].

Figure 6.4 shows the aerial view of an outdoor experiment conducted in this research. The objective of this experiment was to create an AR animated excavation operation. There were two virtual dump trucks and one virtual excavator involved in the AR animation. Real time views of a parking lot in Ann Arbor, Michigan were used for the real backgound. Each dump truck waited to be loaded by the excavator before it traveled

to the dumping area. After it dumped the soil, it returned to the loading area for the next loading cycle. The loading cycle repeated several times before the operation ended. The global coordinates of points P_1 through P_6 in Figure 6.1 were carefully measured in terms of longitude, latitude, and altitude and used inside the corresponding animation trace file to define haul and return routes as well as the location of the excavator.

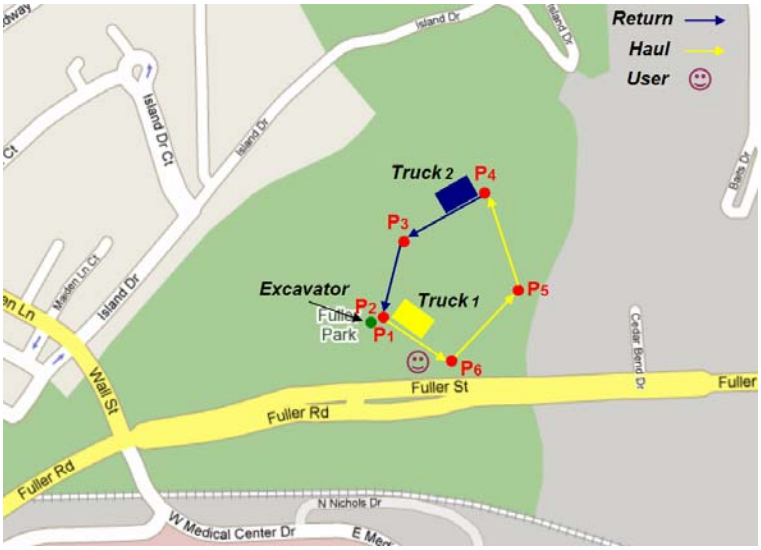


Figure 6.4 – Aerial View of the Earthmoving Operation Experiment

The following assumptions were made in this experiment:

(1) The excavator's interaction with the soil was not modeled or visualized. Only the kinematic motion of the excavator was animated.

(2) The unloading process was modeled only by the truck bucket's upward and downward motion. The interaction of the soil was not modeled or animated.

Figure 6.5 shows the timeline of the animation. Each truck waited for 5 time units in the loading area, took 15 time units to be loaded, traveled for 35 time units to the dumping area, maneuvered for 15 time units to get into the exact dumping position, unloaded the soil for 15 time units, returned to the loading area in 25 time units, and waited there for 5 time units before being placed in the loading zone. The total truck cycle was 110 time units. The excavator, on the other hand, was idle for the first 5 time units, loaded one truck at a time for 15 time units, waited for 10 time units, returned to its initial position in

10 time units, and remained idle for 15 time units before the next loading cycle started. Note that since this was a cyclic operation, the entire process repeated every 55 time units. In this case also, the durations were deterministically set.

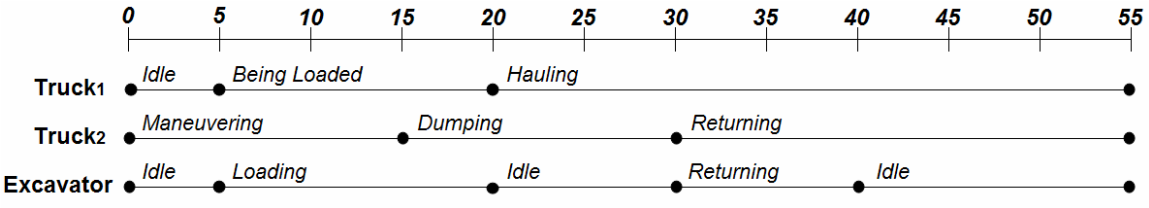


Figure 6.5 – Timeline of the Earthmoving Operation Experiment

Figure 6.6 shows the AR animation created using the ARVISCOPE language while the user was navigating inside the augmented scene. The contents of the augmented scene were continuously updated based on the latest global position of the user coming through the GPS receiver while the user was moving towards the virtual excavator.

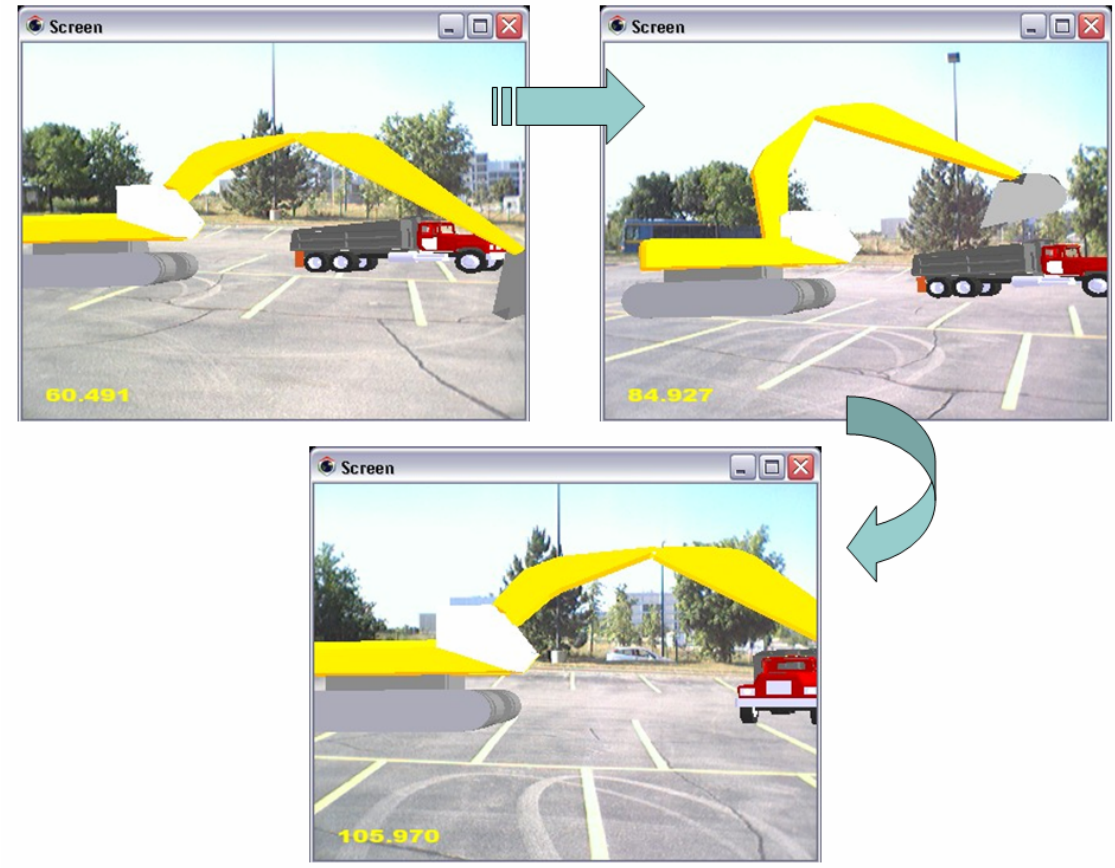


Figure 6.6 –Earthmoving Operation with Continuous Change in User's Global Position

Figure 6.7 shows the same AR animation created using the ARVISCOPE language while the user's head orientation was continually changing. The contents of the augmented scene were continuously updated based on the latest 3DOF head orientation data coming through the head orientation tracker while the user was observing the AR animation.

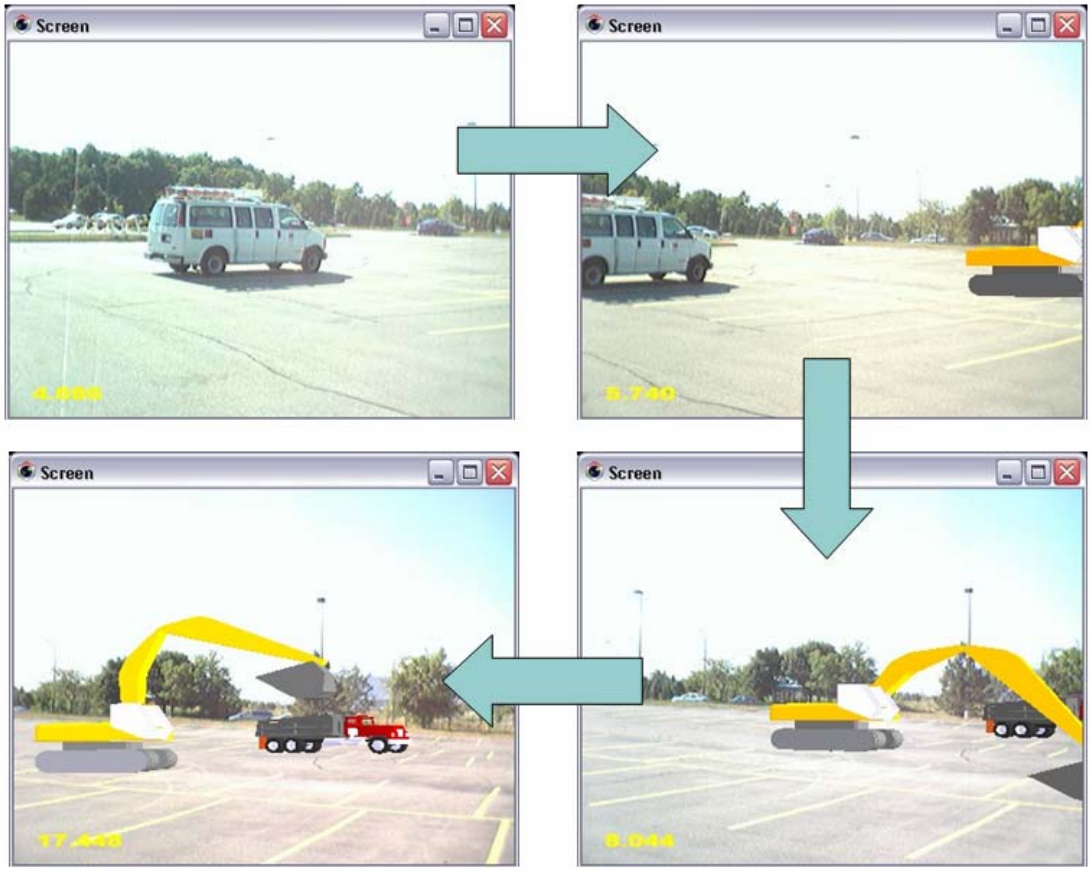


Figure 6.7 –Earthmoving Operation with Change in User's Head Orientation

In both cases shown in Figures 6.6 and 6.7, the ARVISCOPE animation was fully responsive to changes in the user's global position and head orientation, and the CAD objects inside the augmented viewing frustum were continuously updated based on the latest tracked spatial characteristics (i.e. position and orientation).

6.4. Validation of Ability of the AR Framework to Support Geometric Instancing: Structural Steel Erection Operation

As discussed in Chapter 4, geometric instancing in the context of this research is defined as the ability of the AR visualization engine to construct complex scenes that potentially consist of a large number of CAD objects, and to maintain performance levels as the size of the operation increases. Recalling Figure 4.14, geometric instancing allows the creation of very complex scenes such as the erection of an entire structural steel frame consisting of several beams and columns by loading only a few CAD models of steel sections, and placing them repeatedly at appropriate locations using multiple transformation nodes in the corresponding AR-based scene graph (Chapter 4).

In order to validate this aspect of the developed AR platform, a structural steel erection operation was selected. The AR animation of this process was created using the ARVISCOPE language, and a multi storey steel structure was completely modeled and animated in the augmented scene using CAD models of only a few steel sections. In order to create different section sizes from a standard steel cross section, the `SIZE` statement was used.

The operation consisted of a virtual tower crane that would pick up steel sections and install them in their appropriate locations on the steel structure. Figure 6.8 shows the AR animation created using the ARVISCOPE language and compares it to the VR animation of the same operation created using VITASCOPE.

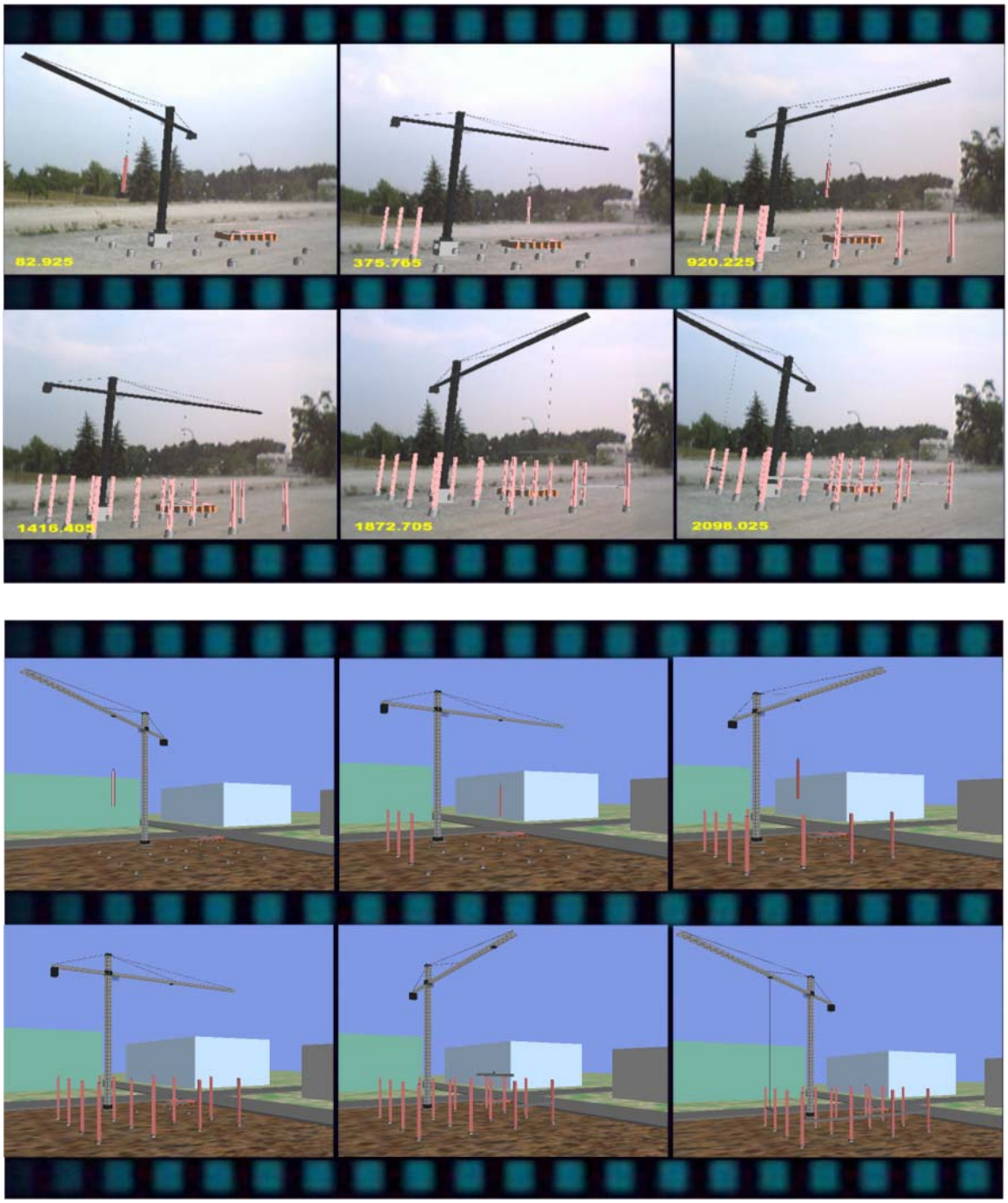


Figure 6.8 – Animated Structural Steel Erection in ARVISCOPE and VITASCOPE

6.5. Validation of Accuracy in Occlusion Handling: Miniature Scale Indoor Experiments

Several proof-of-concept experiments were conducted to validate the functionality of the frame buffer manipulation algorithm developed in this research. Animation trace files of small scale construction operations were created using the ARVISCOPE language. Miniature models of actual construction equipment and materials were used to create the real background of the augmented animation. Small scale CAD models were then superimposed on the real background to create the final augmented view of the operations. In each experiment, the distance between the virtual CAD objects and the user was set to be greater than that of the real construction objects in order to verify that the developed algorithm was capable of detecting and handling incorrect occlusion cases.

The objective of the experiments was to validate that the designed occlusion handling algorithm was capable of detecting and resolving visual occlusion cases in a real time AR animation, and produce visually convincing output representing the modeling operation. The depth data of real objects in these experiments was assumed to be available to the AR application, and was manually input from physical measurements taken around the layout of the physical objects.

Figures 6.9 through 6.12 show results of four indoor proof-of-concept experiments conducted in this research by superimposing the CAD models of construction equipment and machinery on top of miniature construction environments consisting of real scaled construction models. In each experiment, virtual models were placed in the augmented scene in a way that they were completely or partially occluded by real objects. Figure 6.9 depicts a virtual concrete truck occluded by a real brick wall. Figure 6.10 shows a virtual excavator occluded by a real structure. In Figure 6.11, a virtual dozer is partially occluded by a real tower crane. The virtual forklift in Figure 6.12 is occluded by a real container. All occlusion cases were successfully detected and corrected using the occlusion handling algorithm developed in this research (Chapter 5).

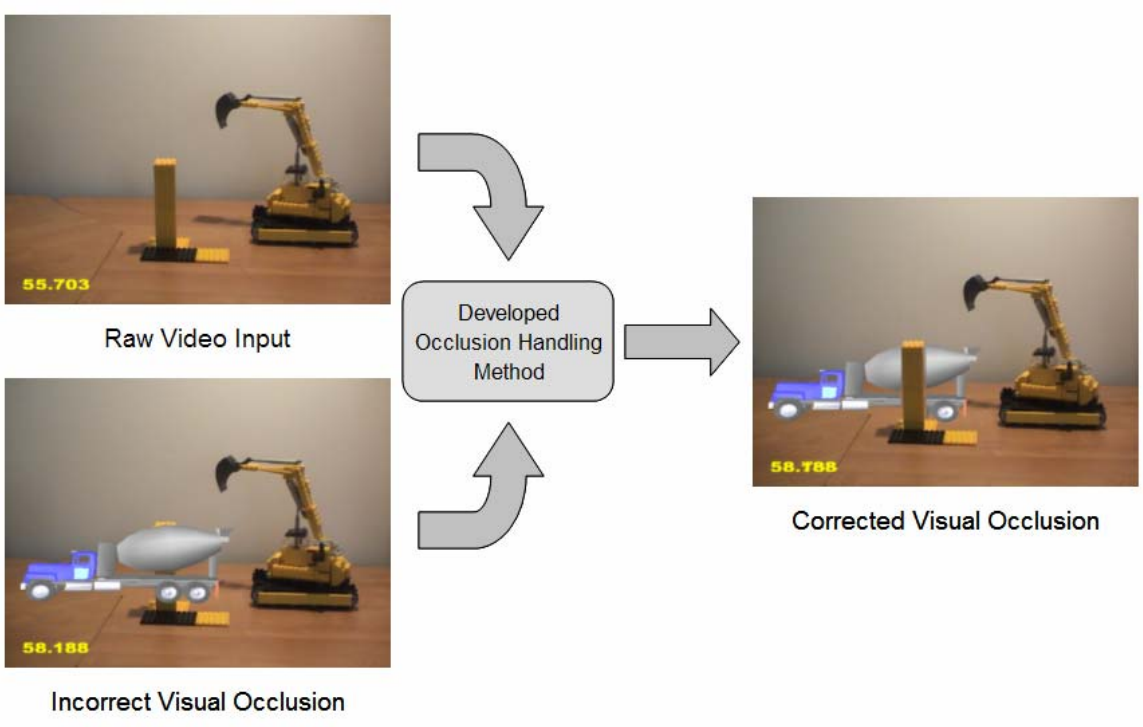


Figure 6.9 – Correcting Occlusion between a Virtual Truck and a Real Brick Wall

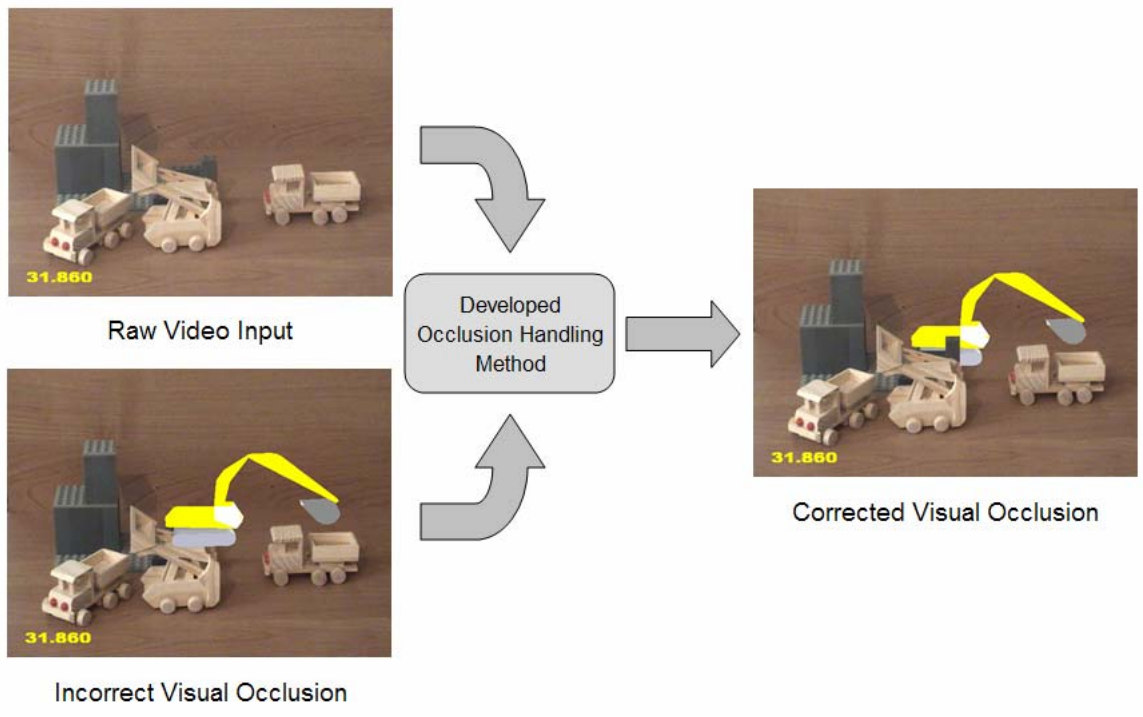


Figure 6.10 – Correcting Occlusion between a Virtual Excavator and a Real Structure

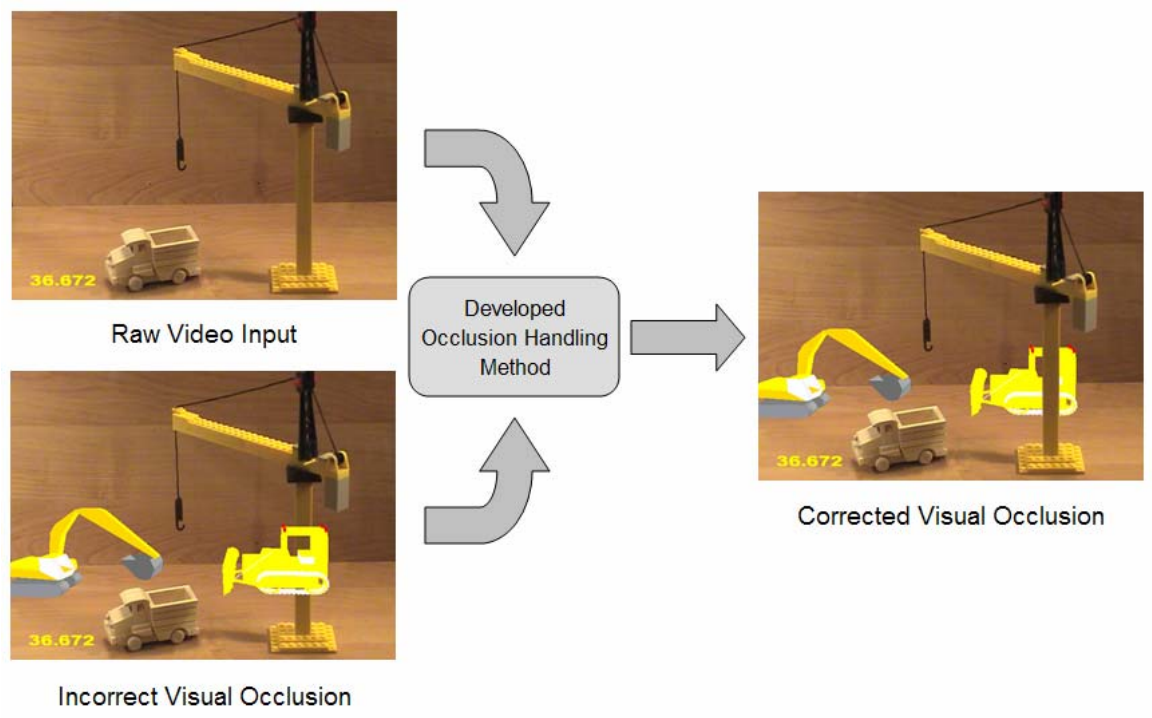


Figure 6.11 – Correcting Occlusion between a Virtual Dozer and a Real Tower Crane

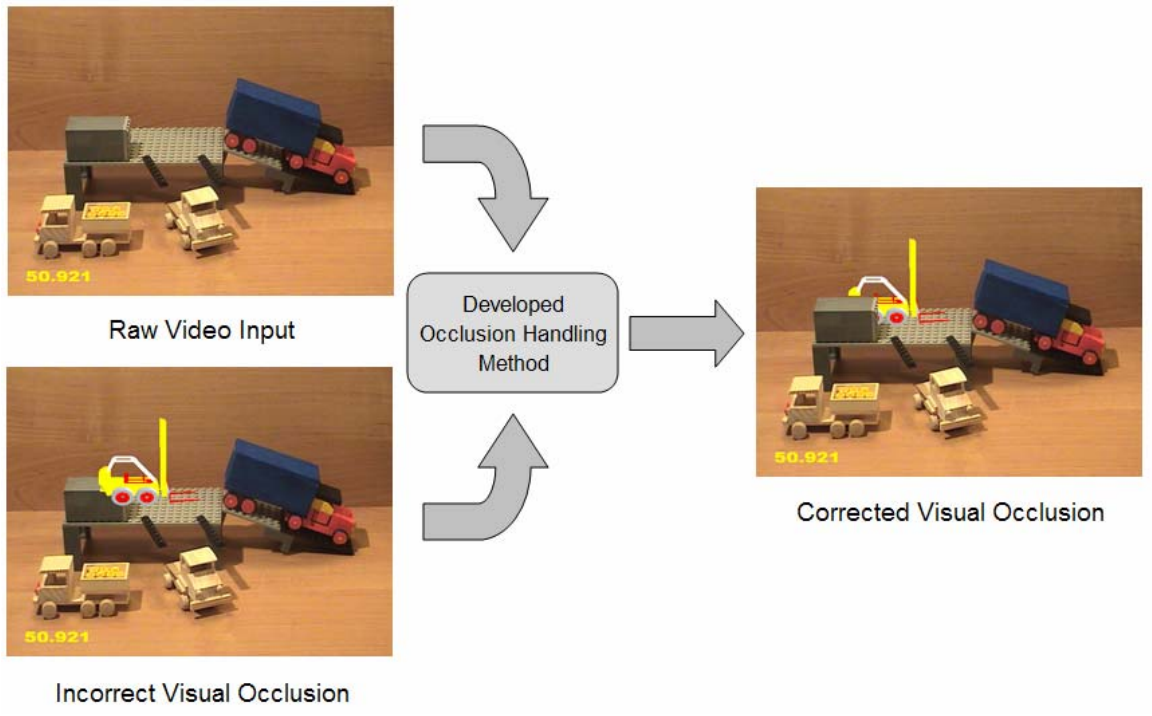


Figure 6.12 – Correcting Occlusion between a Virtual Forklift and a Real Container

6.6. Summary and Conclusions

Creating real time dynamic animations of simulated processes can help decision makers gain a better insight of planned or ongoing operations by examining the spatial and chronological details of the animated processes from different perspectives and at different operation times.

In this Chapter, the ability of both the ARVISCOPE animation authoring language and the UM-AR-GPS-ROVER mobile computing apparatus to create continuous dynamic AR animations of simulated construction operations was successfully validated. The validation parameters were grouped into four main categories including geometric and spatial accuracy in animations, accuracy in 3D spatial user tracking, ability of the AR framework to support geometric instancing, and accuracy in visual occlusion handling. Several validation experiments were conducted in each group to evaluate the robustness and effectiveness of the AR platform developed in this research.

6.7. References

[1] Martinez, J. C. (1996), “STROBOSCOPE: State and Resource Based Simulation of Construction Operations”, PhD Dissertation, University of Michigan, Ann Arbor, MI.

[2] Kamat, V. R. (2003), “VITASCOPE: Extensible and Scalable 3D Visualization of Simulated Construction Operations”, PhD Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[3] Behzadan, A. H., Timm, B. W., and Kamat, V. R. (2008), “General Purpose Modular Hardware and Software Framework for Mobile Outdoor Augmented Reality Applications in Engineering”, Journal of Advanced Engineering Informatics, 22(1), Elsevier Science, New York, NY, 90-105.

[4] Behzadan A. H., and Kamat V. R. (2007), “Reusable Modular Software Interfaces for Outdoor Augmented Reality Applications in Engineering”, International Workshop on Computing in Civil Engineering, American Society of Civil Engineers (ASCE), Pittsburgh, PA, 825-837.

Chapter 7

Conclusions

3D Visualization is an effective and convenient method to verify, validate, and communicate the results of simulated operations. Creating visualized scenes of modeled processes with an appropriate level of detail can help decision makers analyze and evaluate different operational scenarios, and preclude potential space and resource conflicts that would otherwise manifest during the performance of the real operations. This can lead to significant savings in project time and financial resources that would otherwise be expended in manual data analyses or trial and error experiments.

Existing visualization tools primarily used Virtual Reality (VR) environments to create animations from the results of a simulated process. However, as the size and complexity of an operation increases, and with the introduction of additional resources into a process, the task of collecting, managing, and manipulating graphical data to visualize the operation (Model Engineering) becomes time consuming, labor intensive, and often impractical. In an effort to remedy this situation, this research studied the shortcomings of existing visualization approaches and investigated possible ways to improve the current practice of 3D visualization of simulated engineering operations.

This dissertation documented the research that led to the design and implementation of an innovative approach to create continuous dynamic animations of simulation models using Augmented Reality (AR).

AR is a rapidly advancing technology that offers high potential for significant improvement in many scientific and engineering domains such as construction, industrial and mechanical engineering, medicine, aviation, and manufacturing. For example, using AR in construction can allow walking through an actual site and experiencing a virtual facility as it may be built in the future, or looking into the ground and “seeing” utility lines as they exist based on as-built CAD data. AR can also be applied to assembly lines, with the option of presenting the individual work steps to the assembler in an augmented visualization environment during training or actual assembly phases.

AR visualizations can also help product manufacturers in providing required training to production line staff about assembling different components in order to enhance productivity. In aviation, aircraft manufacturers can use AR technology to aid in assembly operations. Flow and inventory control of raw material and final products in a warehouse can also be conveniently animated using AR to minimize order lead times, product delivery times, and maintenance request waiting times. The applications and potential of AR are thus vast and applicable to several engineering domains.

The application of visualization techniques for planning, analysis, and design in construction and civil engineering is relatively new compared to the notable amount of AR related research conducted for diverse applications in fields such as manufacturing, medical operations, military, and gaming. As noted in Chapter 1, previous studies have also explored the application of AR as a state-of-the-art visualization technique for a limited number of architecture and construction applications.

All prior research and applications of AR, however, have focused on developing domain specific platforms intended for very specific purposes, thereby limiting their widespread applicability and reusability.

In addition, as noted in Chapter 1, prior research in AR:

- Did not incorporate arbitrary operations level complexity into AR visualizations.
- Did not consider dynamic objects evolving in an AR scene based on the results of discrete-event simulation.
- Did not afford unrestricted mobility to the observer of a dynamic AR scene.
- Did not consider the interaction between real and virtual objects in an AR scene.
- Did not consider problems associated with enabling AR visualization in outdoor unprepared environments.

This research successfully addressed the abovementioned limitations. In this dissertation, the design and development of AR-based visualization techniques, position and orientation tracking algorithms, and a powerful animation authoring language together with their practical implementation inside a mobile AR visualization tool were discussed. The designed animation language provides a convenient method to automatically author animations of any length and complexity at the operations level of detail using an external software process such as a running Discrete Event Simulation (DES) model. The results of the research have been validated by animating several simulated construction operations in outdoor AR. Each Chapter in this dissertation described the details of individual scientific questions successfully addressed, major challenges involved, and algorithms and methods studied, adapted, developed, and implemented in achieving a particular research objective.

The following list summarizes the individual research challenges successfully addressed and described in the preceding Chapters:

- Visual representation of simulation models (Chapter 2): Continuously communicating with the simulation model of an engineering operation to extract required elements to describe the modeled processes with chronological and spatial accuracy.
- Accurate registration (Chapter 3): Establishing and maintaining a spatial and logical link between the objects of the virtual world and the scenes of the real

environment using state-of-the-art Global Positioning System (GPS) and Three Degree-of-Freedom (3DOF) orientation tracking devices.

- Augmented scene construction, management, and manipulation (Chapter 4): Creating a flexible CAD object hierarchy that enables fast and dynamic handling and manipulation of the graphical contents of an animation.
- Resolution of visual discrepancies in real time (Chapter 5): Handling incorrect visual occlusion cases in which a virtual object inside the animation is partially or completely blocked by a real object that is intended to be closer to the observer of the augmented scene.

In Chapter 2, the details of an animation authoring language called ARVISCOPE designed to visualize simulated processes in dynamic, georeferenced AR environments were presented. Although the main focus of this Chapter was on construction processes, most of the findings of this research are generic and widely applicable to other fields of science and engineering where the need to animate and communicate simulated operations is as important as that in construction. ***The primary contribution of the research presented in Chapter 2 was an AR animation authoring language that enables modelers to automatically create augmented reality animations of construction operations of arbitrary length and complexity simulated using a DES tool.***

In Chapter 3, the details of registration algorithms necessary to track the global position (i.e. longitude, latitude, altitude) and 3D orientation (i.e. yaw, pitch, and roll) of the user's viewpoint were discussed. In addition, methods to reconcile the tracked information with the known global position and orientation of CAD objects to be superimposed in a user's view were described. The results of these computations were used to calculate the relative translation and axial rotations between the user's eyes and the CAD objects at each animation frame during an AR visualization. The relative geometric transformations were then applied to the CAD objects to generate an augmented outdoor environment where superimposed CAD objects stay fixed to their real world locations as the user moves freely in an animation. ***The primary contributions of the research presented in Chapter 3 were the designed UM-AR-GPS-ROVER mobile hardware apparatus, and software***

methods in the AR framework that implement the developed georeferenced registration algorithms to accurately superimpose virtual CAD objects into an AR animation. The hardware apparatus and software methods have been designed to be conveniently extended and reused by other AR application developers simply by importing the designed class libraries into their code, thereby saving significant time and effort that would otherwise be required to reimplement low level hardware interfacing software.

In Chapter 4, a scene graph based CAD object management scheme was introduced. Effective CAD object management was identified as a crucial step in creating realistic visual output in AR animations of simulated processes. Robust CAD object manipulation is essential in creating the illusion that both groups of objects (real and virtual) coexist in a single scene and operate in a seamless manner. The high variability involved in a simulated process as dynamic as a construction operation requires the graphical engine of the AR application to create, load, and handle a large amount of 3D objects in each animation frame. As the size of the operation increases, CAD object manipulation becomes a memory intensive task that can reduce the animation speed and the ability to interact with the animated process in real time. ***The primary contribution of the research presented in Chapter 4 was a flexible and dynamic methodology to facilitate the tasks of constructing and managing the contents of the AR animation of a simulation model by continuously loading, rendering, and displaying the virtual CAD models and real surrounding environment in which the operation takes place.***

In Chapter 5, the design of a pixel level of detail occlusion handling algorithm was discussed. The presented method takes into account all the complexities and uncertainties involved in a dynamic simulated process such as a construction operation by manipulating depth and frame buffers to automatically detect and correctly resolve visual occlusions in dynamic augmented environments. The developed approach presented in this Chapter is unique since it can be easily integrated into a mobile platform which allows the observer of an AR animation to walk freely in the scene observing the ongoing operations from different perspectives. It is capable of automatically resolving incorrect occlusion effects in real time to produce a visually convincing final output. ***The primary***

contribution of the research presented in Chapter 5 was a general purpose occlusion handling algorithm integrated into the core AR platform capable of detecting and correctly resolving occlusion effects in real time.

In Chapter 6, validation exercises evaluating each of the individual research components described in individual Chapters of this dissertation were presented. In addition to validating each individual aspect of the research during its progress, the ability of the ARVISCOPE animation authoring language to create smooth and continuous AR animations of simulated construction processes was also validated at several stages of the work. In particular, the designed language and the implemented software and hardware framework were critically evaluated from the point of view of geometric and spatial accuracy in animations, accuracy in 3D spatial user tracking, ability of the AR framework to support geometric instancing, and accuracy in visual occlusion handling.

In summary, this dissertation has led to two major final products: 1) the ARVISCOPE animation authoring language capable of graphically describing the detailed processes in a simulated engineering operation within a 3D AR environment; and 2) the UM-AR-GPS-ROVER mobile computing apparatus capable of tracking the current global position and head orientation of the scene observer via tracking devices, and continuously computing and displaying the contents of the augmented animation in real time based on this data. When used together in an integrated AR platform, ARVISCOPE and UM-AR-GPS-ROVER can describe, create, and display real time augmented scenes of simulated operations by georeferencing and registering virtual CAD objects on top of live video backgrounds of the real world. The user is embedded in the animation and can walk in the scene with minimum physical constraints to observe the ongoing augmented operations from different perspectives.

The primary contribution of this research to the body of knowledge is that the methods designed in this work improve upon the current practice of verifying, validating, and communicating the results of simulated operations. This was achieved by providing an effective visual means to gain a better understanding of the underlying processes in fully

immersive and interactive augmented environments. Such an environment allows the modeler to examine the dynamics of a simulated construction operation from different perspectives and with different levels of detail. The resulting AR animations are realistic to the extent that they create the illusion that the observer is fully immersed in the environment and as a result, can interact with the operation more effectively while studying an animated scene with the highest level of graphical and operational detail. The designed data communication methods between the tracking devices, underlying DES model, and AR visualization engine are generic and extensible, and can be readily modified and applied to other engineering and scientific domains such as manufacturing and transportation that also experience a need to create visual interfaces to underlying domain processes.

Appendices

Appendix A

ARVISCOPE Language Statements

In this Appendix, the basic ARVISCOPE animation authoring language statements together with their syntax are documented. As described in Chapter 2, there are three groups of statements: scene construction, dynamic, and control statements.

Scene Construction Statements

Syntax: `LOADMODEL <GroupName> <3DFileName>;`
Example: `LOADMODEL Truck CATTruck.3ds;`

This statement is used to assign a CAD file to a certain group of objects in the scene. The CAD file contains the 3D geometric data of the type of objects specified in the first argument. 3D CAD files can be selected from a variety of formats such as AC3D geometry (`.ac`), 3D Studio (`.3ds`), and Lightweight 3D Object (`.lwo`). Most CAD modeling tools can export created 3D models in various formats which facilitates their use in `LOADMODEL` statement.

Syntax: `ORIENTMODEL <GroupName> <Axis> <Degree>;`

Example: `ORIENTMODEL Truck Y 90;`

This statement is used to correct any misalignment between a CAD model's local coordinate axes and those of the global frame. Following ARVISCOPE conventions, X axis is always pointing from west to east and Y axis is pointing upwards. Using the right hand rule for vector operations, Z axis will be always pointing from the depth of the augmented scene towards the user's eyes. Figure A.1 shows how the ORIENTMODEL statement is used to align the local coordinate frames of a CAD model with the ARVISCOPE global axes.

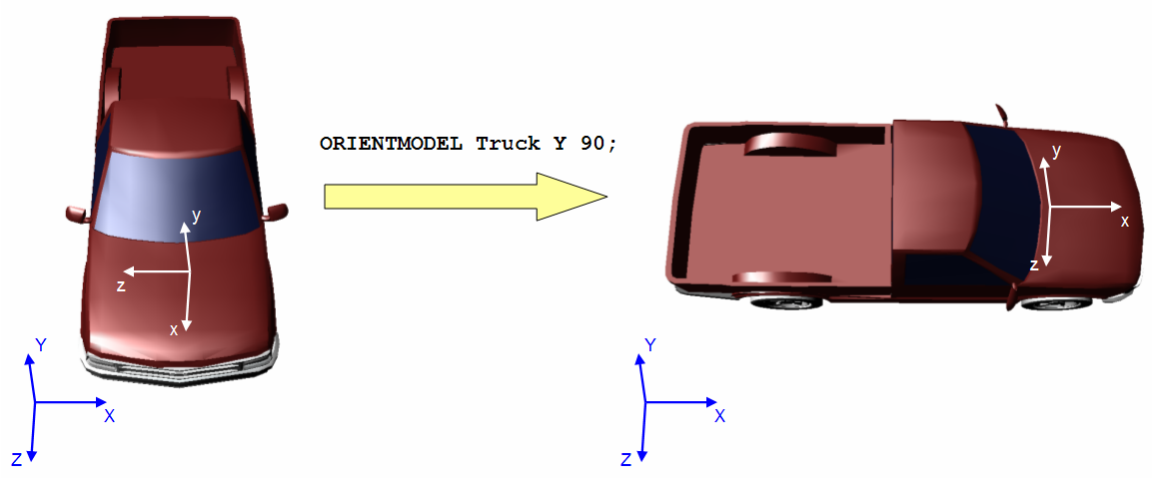


Figure A.1 – Aligning the Local Coordinate Frame of a CAD Model with Global Axes

In this Figure, the initial local coordinate frame of the red truck (located on the hood) is not appropriately aligned with the ARVISCOPE global coordinate frame. As a result, a rotation equal to 90 degrees about the Y axis is applied to the CAD model to fix this misalignment after the CAD model of the truck has been loaded.

Syntax: `CHANGEMODEL <ObjectName> <NewGroupName>;`
Example: `CHANGEMODEL SteelBeam150 BlueSections;`

This statement is used to change the 3D model assigned to a particular object in the scene. This is extremely useful when the appearance of a single object has to be changed during the course of the animation. For example, in order to distinguish between completed and uncompleted work, the modeler may decide that steel beams waiting to be picked up should be assigned the color red and the color of those picked up and installed should be changed to blue. This can be done by defining two distinct groups of objects (i.e. red and blue steal sections) and changing the group of beams from red sections to blue sections after they are installed.

Syntax: `OBJECT <ObjectName> <GroupName>;`
Example: `OBJECT DumpTruck Truck;`

This statement is used to create a single instance of a group of objects. Each call to this statement causes one new object to be created, which is of the type specified in `GroupName`.

Syntax: `REMOVE <ObjectName>;`
Example: `REMOVE DumpTruck;`

This statement is used to permanently remove an object from the augmented scene.

Syntax: `CONNECT <ChildName> <ParentName> <AtPoint>;`
Example: `CONNECT Boom Tower (0,35,0);`

This statement is used to create a meta-object from individual CAD objects. For example, to create a tower crane, individual parts (e.g. foundation, tower, boom, trolley, cable, and hook) can be connected to each other. This will result in an articulated piece of equipment that can be later manipulated at different joints (i.e. internal connection points). Figure A.2 shows how the 3D model of a tower crane can be created using separate CAD models of its components.

Syntax: `STICK <ChildName> <ParentName> <AtPoint>;`
Example: `STICK Hook Cable (0,-1,0);`

This statement is similar to the `CONNECT` statement. The only difference is that any change in the size (scale) of the parent node will not affect the size (scale) of the child node. For example, in the tower crane shown in Figure A.2, lowering and raising the cable is done through changing the size (scale) of the cable in the vertical direction. Since the hook is a child of the cable, connecting it to the cable using the `CONNECT` statement will result in the hook changing size as well. In order to avoid this situation, the `STICK` statement is recommended (as shown in this Figure).

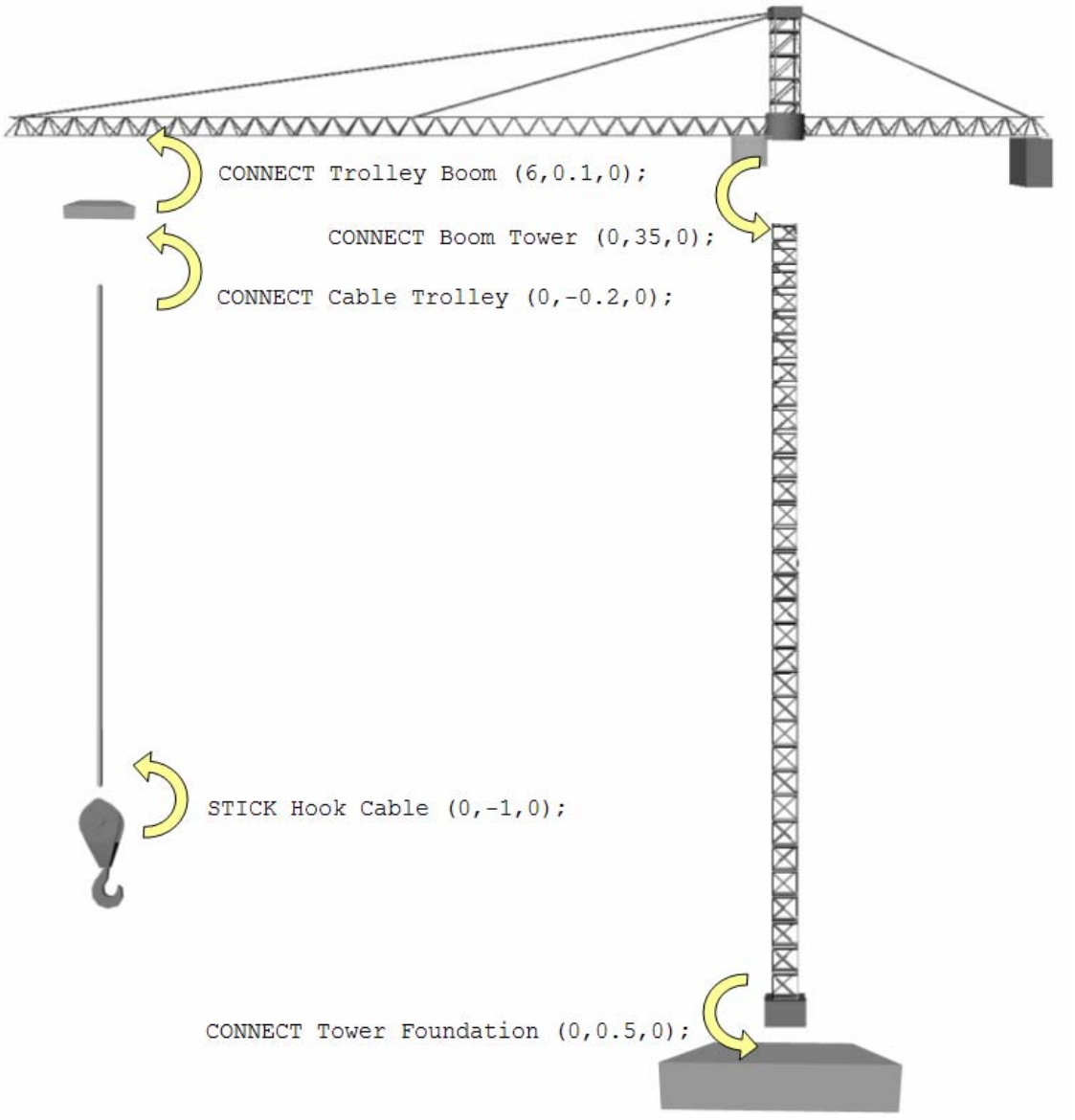


Figure A.2 – Creating a Meta-Object from Individual CAD Models

Syntax: `DISCONNECT <ChildName> <ParentName>;`

Example: `DISCONNECT SteelBeam Hook;`

This statement is used to separate a child object from its parent object. For example, a steel beam previously picked up by a tower crane has to be separated from the hook and placed at its appropriate location. This can be done using the `DISCONNECT` statement.

Syntax: `ROUTE <RouteName> <Points_Array_XYZ>;`

Example: `ROUTE TruckRoad (-83.714569,42.298058,270.00)`
`(-83.714553,42.298092,270.00)`
`(-83.714553,42.298136,270.00)`
`(-83.714600,42.298186,270.00);`

This statement establishes a motion path by defining its beginning, intermediate, and end points. The coordinates of these points have to be expressed in global frame (i.e. longitude, latitude, and altitude). The first, second, and third coordinate values for each point are longitude, latitude, and altitude respectively. ARVISCOPE automatically constructs a route by connecting line segments defined by these points. It is also possible to define a route by specifying relative distances between its points and a reference point with known global position in terms of longitude, latitude, and altitude. This is described in `ROUTE REL` statement.

Syntax: `ROUTE REL <Reference> <RouteName> <Points_Array_XYZ>;`

Example: `ROUTE REL Flag TruckRoad (+17.3997,1.00,-17.3348)`
`(+17.3997,1.00,-17.3348)`
`(+23.7191,1.00,-27.9968)`
`(+19.8434,1.00,-33.5500);`

This statement is similar to the previous statement and it also establishes a motion path by defining its constituent points. The difference is that the coordinates of the beginning, intermediate, and end points have to be expressed relative to a reference point (the coordinate of which has been previously defined in the global space). The required steps to create a reference point in ARVISCOPE are described in Appendix B. The first, second, and third coordinate values for each point are its relative distance along X, Y, and Z axes to the reference point. ARVISCOPE automatically constructs a route by calculating global coordinate values of each point and connecting the corresponding line segments. It is also possible to define a route by specifying absolute global positions of

its points in terms of longitude, latitude, and altitude. This is described in `ROUTE` statement.

Syntax: `POSITION <ObjectName> AT <Point_XYZ>;`
Example: `POSITION DumpTruck AT (-83.714569,42.298058,270.00);`

This statement is used to place a CAD object in the augmented scene by defining its global position in terms of longitude, latitude, and altitude values. It is also possible to place a CAD object in the scene by specifying relative distances between the object and a reference point with known global position in terms of longitude, latitude, and altitude. This is described in `POSITION REL` statement. In addition, an object can be placed on a predefined route in the scene. This is described in `POSITION ON` statement.

Syntax: `POSITION <ObjectName> ON <RouteName>;`
Example: `POSITION DumpTruck ON TruckRoad;`

This statement is used to place a CAD object on a predefined route. By default, the object will be placed at the beginning of the route. It is also possible to place a CAD object in the scene by specifying relative distances between the object and a reference point with known global position in terms of longitude, latitude, and altitude. This is described in `POSITION REL` statement. In addition, an object can be placed on a specific point with known global position in terms of longitude, latitude, and altitude. This is described in `POSITION AT` statement.

Syntax: `POSITION <ObjectName> REL <Reference> <Point_XYZ>;`
Example: `POSITION DumpTruck REL Flag (+17.40,1.00,-17.34);`

This statement is used to place a CAD object relative to a reference point, the coordinate values of which have been previously defined in global space. The required steps to create a reference point in ARVISCOPE are described in Appendix B. It is also possible to place a CAD object on a specific point with known global position in terms of longitude, latitude, and altitude. This is described in `POSITION AT` statement. In addition, an object can be placed on a predefined route in the scene. This is described in `POSITION ON` statement.

Syntax: `ADJUST GROUP <GroupName> RGP <Value>;`
Example: `ADJUST GROUP Truck RGP 6.5;`

This statement is used to assign a Rear Guide Point (RGP) value to a group of objects. RGP is an imaginary point behind the Local Origin (LO) of a CAD object which always remains on the route (just as the local origin of the CAD object does) when the CAD object is traveling along it. Assigning RGP to a CAD object leads to a more smooth motion on routes as it eliminates sharp changes in the object heading when it reaches intermediate points on a route. Figure A.3 shows how RGP affects the motion of a truck at an intermediate point of a route. In this Figure, the two main points are denoted by LO and RGP on the CAD object. As the CAD object approaches the intersection of two segments (Figure A.3.a), both RGP and LO points are on the route. Since LO is in front of RGP, it reaches the intersection first (Figure A.3.b). While LO remains on the route as the CAD object turns, RGP is clamped to the route (Figure A.3.c) until RGP also reaches the intersection (Figure A.3.d). At this point, the turn is complete and both LO and RGP points will be transferred to the next route segment (Figure A.3.e). The most important step in this Figure is part (c) in which LO is on the next route segment and RGP is located on the previous segment. This, in fact, has the primary effect on smooth motion as any sharp turn of the CAD object may violate the rule that both these points have to

remain on the route at any given time instant. Since this is done continuously and for all the points on a route segment, the final result is a smooth turn as opposed to a sudden change in the CAD object heading.

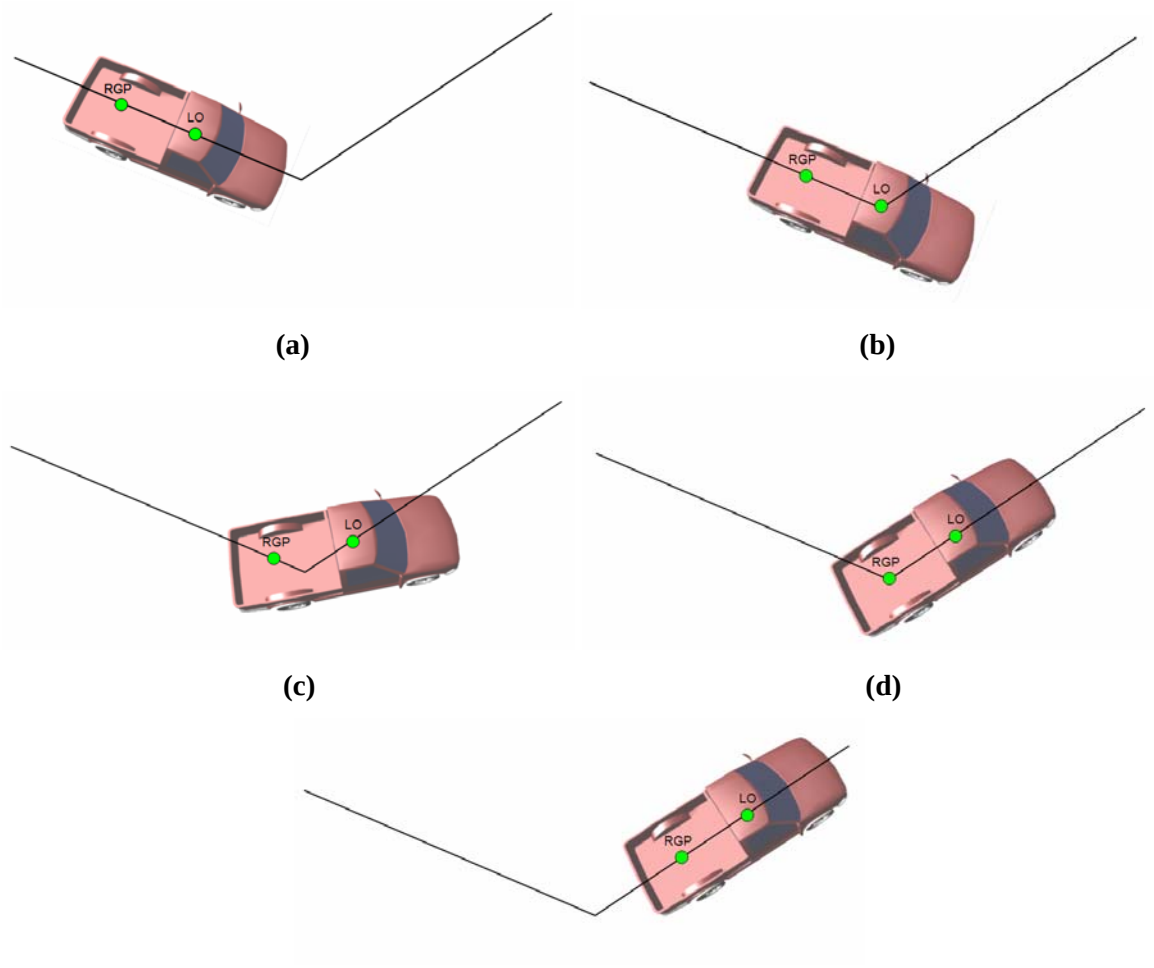


Figure A.3 – Using RGP to Depict Smooth Turns on a Route

Syntax: `ADJUST GROUP <GroupName> FORCLR <Value>;`
Example: `ADJUST GROUP Equipment FORCLR 3.0;`

This statement is used to define the fore clearance distance for a group of CAD objects. Fore clearance is defined as the distance between the local origin of a CAD object and the closest CAD object located in front of it on the same route.

Syntax: `ADJUST GROUP <GroupName> AFTCLR <Value>;`
Example: `ADJUST GROUP Equipment AFTCLR 8.0;`

This statement is used to define the after clearance distance for a group of CAD objects. After clearance is defined as the distance between the local origin of a CAD object and the closest CAD object located behind it on the same route. Fore and after clearance values are important especially for safety measures on a construction site and when visibility is limited in certain directions. For example, dump trucks should not get too close to each other or they should always keep a certain clearance distance from pieces of excavating equipment. Figure A.4 shows the definition of fore and after clearance values for an excavator.

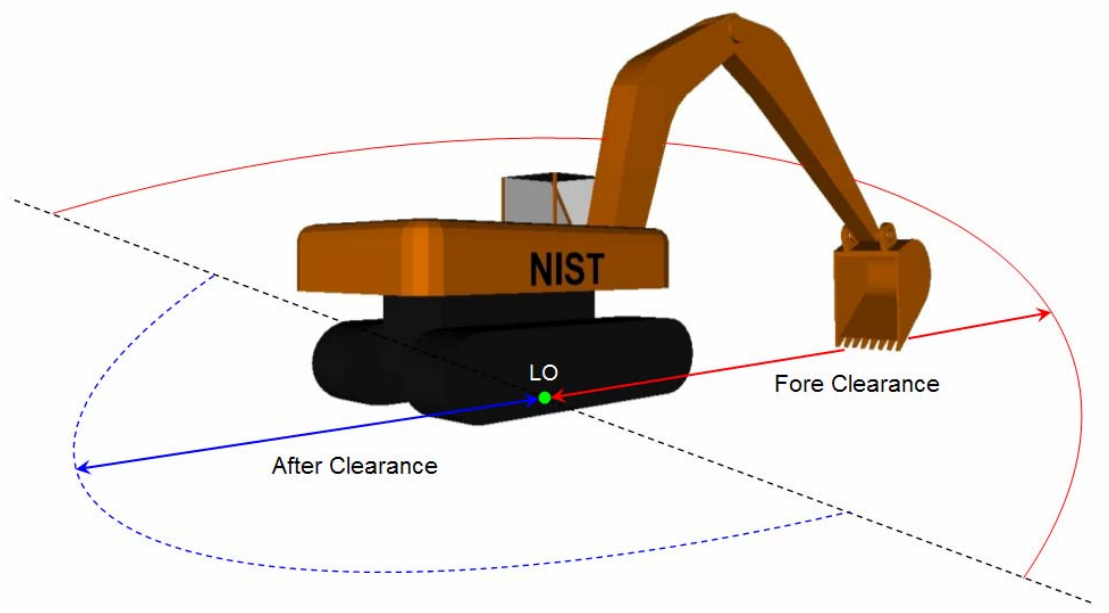


Figure A.4 – Definition of Fore and After Clearance Distances

Syntax: `ADJUST MODEL <ObjectName> RGP <Value>;`
Example: `ADJUST MODEL DumpTruck RGP 6.5;`

This statement is used to assign a RGP value to a single CAD object.

Syntax: `ADJUST MODEL <ObjectName> FORCLR <Value>;`
Example: `ADJUST MODEL Excavator FORCLR 3.0;`

This statement is used to define the fore clearance distance for a single CAD object.

Syntax: `ADJUST MODEL <ObjectName> AFTCLR <Value>;`
Example: `ADJUST MODEL Excavator AFTCLR 8.0;`

This statement is used to define the after clearance distance for a single CAD object.

Dynamic Statements

Syntax: `HRZORIENT <ObjectName> <YawValue>;`
Example: `HRZORIENT Plane 50;`

This statement is used to change the local horizontal orientation (heading) of a CAD model in the augmented scene. This is done by specifying the amount of yaw angle in the second parameter. Figure A.5 shows the definition of yaw, pitch, and roll angles for the blade of a dozer in ARVISCOPE coordinate system.

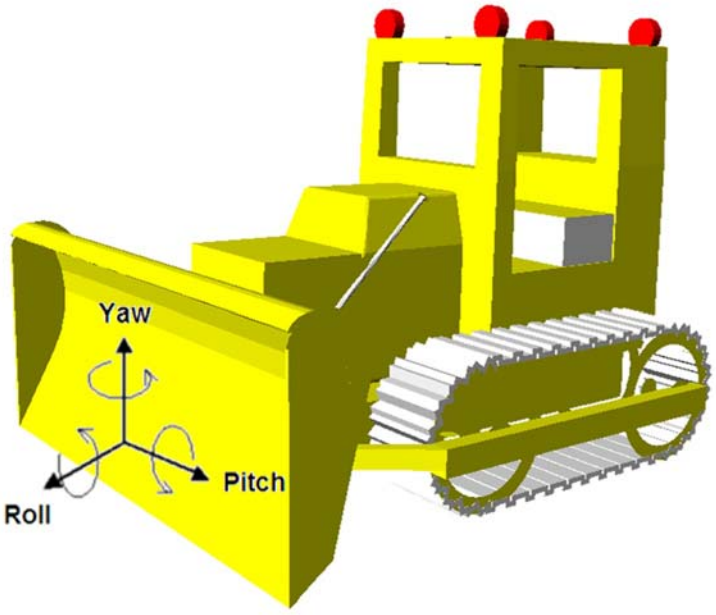


Figure A.5 – Definition of Yaw, Pitch, and Roll Angles

Syntax: `VRTORIENT <ObjectName> <PitchValue>;`
Example: `VRTORIENT Plane 30;`

This statement is used to change the local vertical orientation of a CAD model in the augmented scene. This is done by specifying the amount of pitch angle in the second parameter.

Syntax: `SIDEORIENT <ObjectName> <RollValue>;`
Example: `SIDEORIENT Plane 10;`

This statement is used to change the local side orientation of a CAD model in the augmented scene. This is done by specifying the amount of roll angle in the second parameter.

Syntax: `ORIENT <ObjectName> <Axis> <Value> <Duration>;`
Example: `ORIENT Boom Y 45 15;`

This statement is used to change the orientation of a CAD model about a certain plane by a certain value in a specified duration of animation time. Using the right hand rule, all clockwise rotations are considered negative and all counterclockwise rotations are positive.

Syntax: `ORIENTTO <ObjectName> <Axis> <Value> <Duration>;`
Example: `ORIENTTO Boom Y 30 15;`

This statement is used to change the orientation of a CAD model about a certain plane to a certain target value in a specified duration of animation time. This statement is similar to the `ORIENT` statement. The only difference is that the third argument provided to the `ORIENT` statement indicates the amount of change in orientation while the same argument in the `ORIENTTO` statement indicates the final target orientation value of the CAD object. Again, all clockwise rotations are considered negative and all counterclockwise rotations are positive.

Syntax: `TRAVEL <ObjectName> <RouteName> <Duration>;`
Example: `TRAVEL Truck TruckRoad 60;`

This statement is used to move a CAD model on a predefined route in a certain number of animation time units. Using a constant speed formula for a moving object, the speed of the CAD object is determined internally by dividing the length of the route (sum of the lengths of all its line segments) by the specified duration. The heading (yaw) of the moving CAD model is continuously changed to the direction of the route so that it is always aligned with the moving direction.

Syntax: `TRANSFER <ObjectName> <RouteName> <Speed>;`
Example: `TRANSFER Truck TruckRoad 45;`

This statement is used to move a CAD model on a predefined route at a certain constant speed. Using a constant speed formula for a moving object, the travel duration of the CAD object is determined internally by dividing the length of the route (sum of the lengths of all its line segments) by the specified speed. The heading (yaw) of the moving CAD model is continuously changed to the direction of the route so that it is always aligned with the moving direction.

Syntax: `SHIFT <ObjectName> <Vector> <Duration>;`
Example: `SHIFT Truck (10,0,5) 15;`

This statement is used to translate a CAD model along a certain vector in the 3D space. The translation value is determined in the vector provided as the second argument. Using a constant speed formula for a moving object, the speed of the CAD object is determined internally by dividing the length of the vector by the specified duration.

Syntax: `SHIFTTO <ObjectName> <Point_XYZ> <Duration>;`
Example: `SHIFTTO Truck (-83.714569,42.298058,270.00) 15;`

This statement is used to translate a CAD model to a certain global point, the position of which is provided in terms of longitude, latitude, and altitude. Using a constant speed formula for a moving object, the speed of the CAD object is determined internally by dividing the distance (between the current position of the CAD model and its new position) by the specified duration.

Syntax: `SIZE <ObjectName> <ScaleChange> <Duration>;`
Example: `SIZE Cable (0,50,0) 15;`

This statement is used to change the size (scale) of a CAD model by a certain amount along its local axes. The `ScaleChange` argument consists of a vector which expresses the change in scale (size) of the CAD model along its local X, Y, and Z axis respectively. A positive value indicates expansion and a negative value indicates shrinkage in size. For example, if the current scale of a CAD object is `(5,5,5)`, applying the `SIZE` statement with a scale factor of `(2,2,2)` will result in its final scale being `(7,7,7)`.

Syntax: `SIZETO <ObjectName> <ScaleTarget> <Duration>;`
Example: `SIZETO Cable (0,55,0) 15;`

This statement is used to change the size (scale) of a CAD model to a certain size (scale) along its local axes. The `ScaleChange` argument consists of a vector which expresses the target scale (size) of the CAD model along its local X, Y, and Z axes respectively. Again, a positive value indicates expansion and a negative value indicates shrinkage in size. This statement is similar to the `SIZE` statement. The only difference is that the second argument provided to the `SIZE` statement indicates the amount of change in the current size (scale) of the CAD model while the same argument in the `SIZETO` statement indicates the final size (scale) value of the CAD object. For example, if the current scale of a CAD object is `(5,5,5)`, applying the `SIZETO` statement with a scale factor of `(2,2,2)` will result in its final scale being `(2,2,2)`. Figure A.6 shows examples of the `SIZE` and `SIZETO` statements.

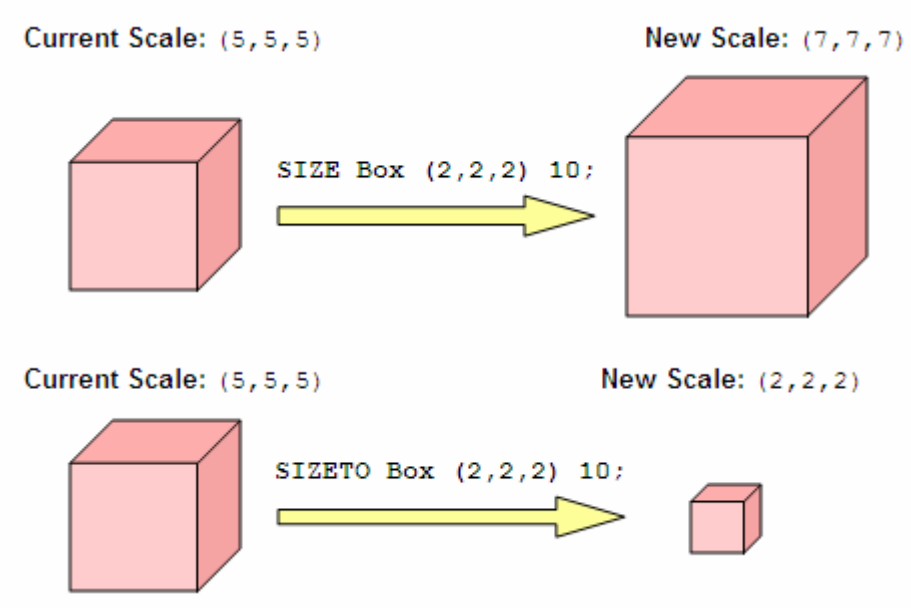


Figure A.6 – Changing the Scale of a CAD Model Using Two Different Statements

Control Statements

Syntax: `SIMTIME <TimeValue>;`

Example: `SIMTIME 125;`

When this statement is called, further reading of the animation trace file is suspended until the animation time reaches the time value specified as the argument. ARVISCOPE then processes the statements that follow it until the next `SIMTIME` statement is reached. After verifying that the `TimeValue` is greater than or equal to the current animated time, ARVISCOPE suspends the reading of any more lines from the trace file until the animation time specified by the `SIMTIME` statement has been reached or exceeded. When that happens, ARVISCOPE reads and processes the next line(s) in the trace file until another `SIMTIME` statement is encountered. This procedure repeats until the end of the trace file is reached. Figure A.7 shows how a sample trace file clustered using the `SIMTIME` statement is read and processed by ARVISCOPE in the course of an animation.

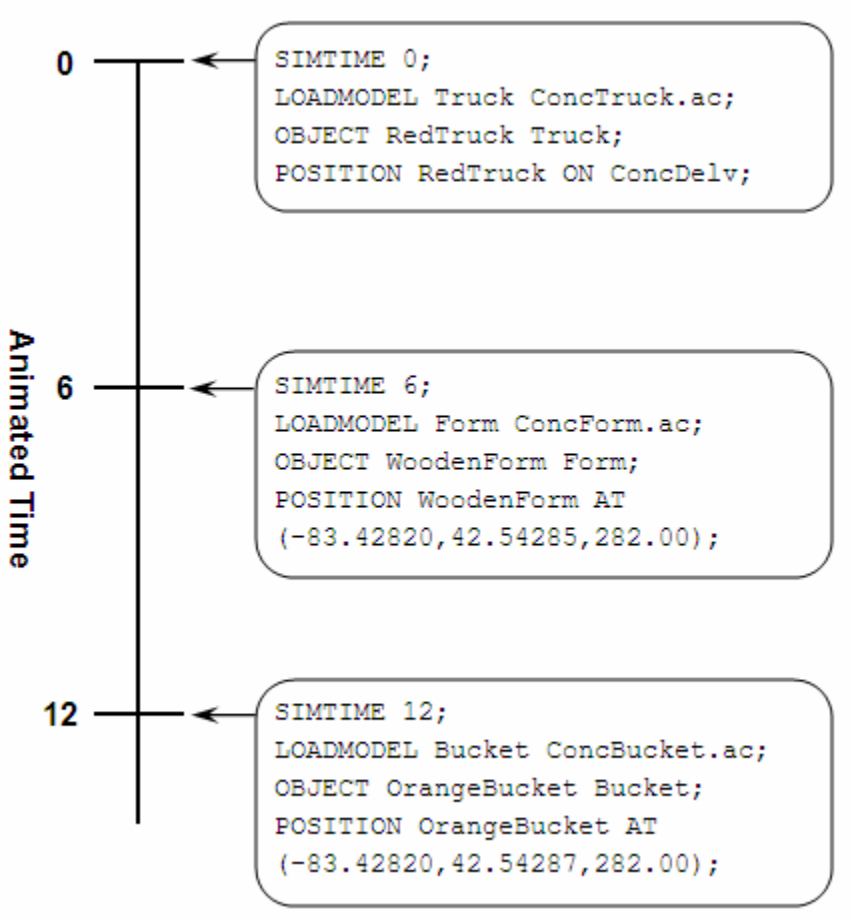


Figure A.7 – Clustering an Animation Trace File Using the `SIMTIME` Statement

By default, an AR animation runs at a 1:1 time ratio in ARVISCOPE. This essentially means that one real clock second corresponds to one simulation time unit. Thus, if a truck is set to move along a route for 2 minutes, its moving duration has to be set to 120 units inside the animation trace file. This guarantees real time user interaction with the scene by walking inside the animation at normal speed. A too low or too high animation speed keeps the user from being involved at a normal rate of walking or head rotation. The exception, however, is the case in which the user is only an observer of the animation with no real interaction with the scene. In this case, the animation speed can be set to be higher or lower than the normal rate to animate very slow or very fast operations respectively.

Appendix B

Defining a Global Reference Point

A global reference point is a point in the 3D augmented space with known longitude, latitude, and altitude values. In order to place 3D objects in the scene (`POSITION` statement) or defining points of a route (`ROUTE` statement) in ARVISCOPE, the first step is to obtain their global positions in terms of longitude, latitude, and altitude values to be used inside the animation trace file. This can become a very time consuming and labor intensive task as the size of the project and the number of points of interest increase. An intuitive approach to reduce the amount of site measurements for each point was to replace the global position of a point by its relative distances to a known global reference point (i.e. known benchmark). In most cases, this information is already provided to the modeler in the project documentation and survey maps. As discussed in Chapter 2, it is a common practice in drawing the site layout plans to specify local position of important objects relative to the positions of a number of known surveying benchmarks. Using the positions of these benchmarks as global reference points in creating ARVISCOPE animation trace files can result in significant time savings during the process of modeling and animating the corresponding simulated construction operation.

A global reference point in ARVICOPE is defined as an object placed in a known global position using the `POSITION` statement. The geometry of this object can be selected arbitrarily and does not effect the overall AR animation. In order to be consistent, 3D models of simple primitives (e.g. small colored spheres or boxes) were used in this

research to define a global reference point. After a global reference point is defined, it can be used by other statements inside the animation trace file to place more 3D objects in the augmented scene relative to the global reference point. As described earlier in this Appendix and in Chapter 2, `POSITION` and `ROUTE` are two primary ARVISCOPE statements that use global referencing. The details of how ARVISCOPE interprets each statement and calculates global position of each point or 3D object based on its relative distance to the global reference point is discussed in detail in Chapter 2 and Appendix D. Figure B.1 shows portions of an ARVISCOPE trace file.

```
[1]  /The global reference point
[2]  LOADMODEL Reference RedBox.ac;
[3]  OBJECT BM Reference;
[4]  POSITION BM AT (-83.707000,42.29560,265.00);
[5]
[6]  /The hauling route
[7]  ROUTE HaulRoad REL BM (20.000,0,-5.500)
[8]                        (112.125,0,44.100)
[9]                         (191.675,0,-56.450)
[10]                        (163.400,0,-142.900);
[11] /The excavator spot
[12] LOADMODEL ExcBase ExcBase.ac;
[13] LOADMODEL ExcCabin ExcCabin.lwo;
[14] LOADMODEL ExcBoom ExcBoom.lwo;
[15] LOADMODEL ExcStick ExcStick.lwo;
[16] LOADMODEL ExcBucket ExcBucket.ac;
[17] OBJECT ExcBase ExcBase;
[18] OBJECT ExcCabin ExcCabin;
[19] OBJECT Boom ExcBoom;
[20] OBJECT Stick ExcStick;
[21] OBJECT ExcBucket ExcBucket;
[22] CONNECT ExcCabin ExcBase (0,0,0);
[23] CONNECT Boom ExcCabin (3,4.48,0);
[24] CONNECT Stick Boom (10.5,7.25,0);
[25] CONNECT ExcBucket Stick (12.2,-9.3,0);
[26] POSITION ExcBase REL BM (5.00,0.00,-3.00);

     . . .
```

Figure B.1 – Using Global Referencing in an ARVISCOPE Animation Trace File

In this Figure, the 3D file of a red box is first loaded (line 2) and an instance of this model called `BM` is created as a global reference point (line 3). `BM` is then placed at a

known position in the global space (line 4). The route `HaulRoad` is then expressed be defining the coordinates of its end and mid points relative to `BM` (lines 7 through 10). A virtual model of an excavator is then created by loading and connecting individual 3D models for the base, cabin, boom, stick, and shovel (lines 12 through 25). The virtual excavator is then placed relative to `BM` (line 26).

Appendix C

Guide to Create an ARVISCOPE Animation

In this Appendix, the process of creating an ARVISCOPE animation trace file is discussed in more detail in order to explain how the statements inside the trace file describe an augmented scene and the motion of objects during the course of an Augmented Reality (AR) animation. In order to achieve this objective, an earthmoving operation (also discussed in Section 6.3) is visualized using the ARVISCOPE animation authoring language. This operation consists of two dump trucks and one excavator. Each dump truck waits to be loaded by the excavator before it travels to the dumping area. After it dumps the soil, it returns to the loading area for the next loading cycle. The loading cycle repeats several times before the operation ends.

The animated earthmoving operation in this Appendix is for illustrative purposes. As a result, in order to maintain clarity in explaining the example and the process of creating an ARVISCOPE animation trace file, activity durations have been assumed to be deterministic and with rounded and stepped values. In addition, the travel distances have been assumed to be relatively small so that both the loading and dumping areas are simultaneously visible in the animation snapshots shown in the subsequent Figures. The overall process described in this Appendix can, however, be used for different simulation models with arbitrary stochastic parameters and levels of detail of the underlying processes.

The cycle starts with dump truck 1 being loaded by the excavator while dump truck 2 unloads itself in the dumping area. As soon as dump truck 1 is loaded, it starts hauling to the dumping area while dump truck 2 returns to the loading area. The cycle continues with dump truck 1 dumping soil and dump truck 2 being loaded by the excavator after which the two dump trucks start moving towards the loading and dumping areas respectively. The entire dump truck cycle takes 110 animation time units. This value is 55 time units for the excavator since it loads two dump trucks consequently.

The following assumptions were made in this experiment:

(1) The excavator's interaction with the soil was not modeled or visualized. Only the kinematic motion of the excavator was animated ffdf.

(2) The unloading process was modeled only by the truck bucket's upward and downward motion. The interaction of the soil was not modeled or animated.

Figure C.1 shows the layout of the jobsite where the earthmoving operation takes place. As shown in this Figure, two separate routes have been defined: `Haul` which is defined by points 1 through 4 and used by a loaded dump truck to haul the soil from the loading area to the dumping area, and `Return` which is defined by points 4, 5, and 1, and used by an empty dump truck to return from the dumping area to the loading area. The Excavator is also positioned in the scene such that its local origin is exactly placed on the global reference point `BM` the coordinates of which (in terms of longitude, latitude, and altitude) are known.

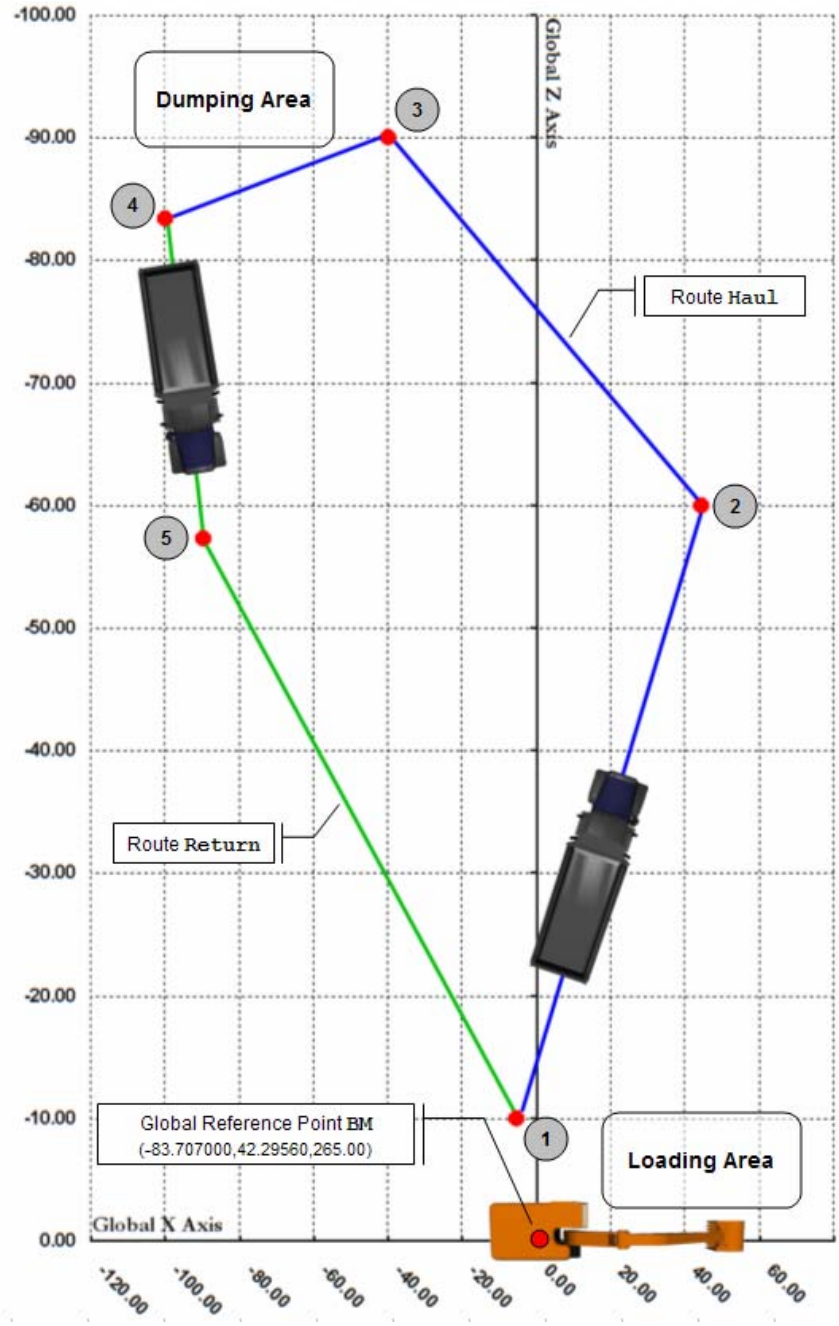


Figure C.1 – Layout of the Earthmoving Operations Jobsite

Figure C.2 shows the beginning part of the ARVISCOPE animation trace file associated with the earthmoving operation illustrated above. As discussed in Chapter 2, the animation trace file is a list of ARVISCOPE statements from three different categories (scene construction, dynamic, and control statements) chronologically ordered to describe the scene and events that are happening in the augmented world at each animation frame.

```
[1] SIMTIME 0;

[2] LOADMODEL Reference BM.ac;
[3] OBJECT BM Reference;
[4] POSITION BM AT (-83.707000,42.29560,265.00);

[5] ROUTE Return REL BM (83.400,0.000,-100.000)
                        (57.250,0.000,-89.800)
                        (10.000,0.000,-5.500);
[6] ROUTE Haul REL BM (10.000,0.000,-5.500)
                      (60.000,0.000,44.100)
                      (90.000,0.000,-40.000)
                      (83.400,0.000,-100.000);

[7] /The Trucks
[8] LOADMODEL Truck RedMackBody.lwo;
[9] LOADMODEL TruckBucket RedMackBucket.lwo;
[10] ADJUST GROUP Truck AFTCLR 10;
[11] ADJUST GROUP Truck RGP 5;
[12] OBJECT Truck1 Truck;
[13] OBJECT TruckBucket1 TruckBucket;
[14] CONNECT TruckBucket1 Truck1 (-7,2.2,0);
[15] OBJECT Truck2 Truck;
[16] OBJECT TruckBucket2 TruckBucket;
[17] CONNECT TruckBucket2 Truck2 (-7,2.2,0);
[18] POSITION Truck1 ON Haul;
[19] POSITION Truck2 ON Return;

[20] /The Excavator
[21] LOADMODEL Base ExcBase.ac;
[22] LOADMODEL Cabin ExcCabin.lwo;
[23] LOADMODEL Boom ExcBoom.lwo;
[24] LOADMODEL Stick ExcStick.lwo;
[25] LOADMODEL Bucket ExcBucket.ac;
[26] OBJECT ExcBase Base;
[27] OBJECT ExcCabin Cabin;
[28] OBJECT ExcBoom Boom;
[29] OBJECT ExcStick Stick;
[30] OBJECT ExcBucket ExcBucket;
[31] CONNECT ExcCabin ExcBase (0,0,0);
[32] CONNECT ExcBoom ExcCabin (3,4.48,0);
[33] CONNECT ExcStick ExcBoom (10.5,7.25,0);
[34] CONNECT ExcBucket ExcStick (12.2,-9.3,0);
[35] POSITION ExcBase REL BM (0.00,0.00,0.00);
[36] ORIENT ExcCabin HOR -10 0.00;
```

Figure C.2 – Contents of the ARVISCOPE Animation Trace File (Part 1)

In Figure C.2, the numbers inside the parentheses at the beginning of each line of the trace file are not parts of the trace file and they have only been added to facilitate clarity in the upcoming discussion. The statement group shown in Figure C.2 is preceded by the control statement `SIMTIME` with an argument `0` (line 1), which means that all the subsequent statements in this group are executed immediately after the animation starts. The first set of statements shown in this Figure (lines 2, 3, and 4) describe the layout of the augmented scene by defining the global reference point `BM` (Appendix B) and placing it on the scene. The two routes (`Return` and `Haul`) are then defined by providing the position for their beginning, intermediate, and end points along the three major axes X, Y, and Z relative to the reference point `BM` (lines 5 and 6).

In lines 7 through 19 of Figure C.2, the two dump trucks are created and placed in the scene. Line 7 contains a comment statement and is not executable. In lines 8 and 9, CAD models for the dump truck chassis and bucket are loaded using the `LOADMODEL` statement. The After Clearance and Rear Guide Point (RGP) values for the dump trucks are then set in lines 10 and 11. As discussed in Appendix A, these two values can be optionally specified for a group of CAD objects or for certain individual CAD objects.

An instance of the dump truck chassis called `Truck1` is defined in line 12 followed by the definition of an instance of the dump truck bucket called `TruckBucket1` in line 13. `TruckBucket1` is then connected to `Truck1` to form a complete dump truck (line 14). The same set of statements is repeated in lines 15, 16, and 17 to create the second dump truck called `Truck2`. `Truck1` is then placed on route `Haul` (line 18) and `Truck2` is placed on route `Return` (line 19).

Line 20 contains a comment statement and is not executable. In lines 21 through 25, five separate CAD models for the excavator base, cabin, boom, stick, and bucket are loaded and stored as five distinct groups of CAD models with specified naming conventions. In lines 26 through 30, an instance of each of these groups is created for the excavator shown in Figure C.1. These instances are called `ExcBase`, `ExcCabin`, `ExcBoom`, `ExcStick`, and `ExcBucket`. In line 31, `ExcCabin` is connected to `ExcBase` as a

child object. `ExcBoom` is then connected to `ExcCabin` in line 32 and `ExcStick` is connected to `ExcBoom` is line 33. Finally, `ExcBucket` is connected to `ExcStick` which results in a complete excavator meta-object. In line 35, `ExcBase` is placed in the scene relative to the global reference point `BM`. Since the relative position vector (last argument in line 35) is `(0,0,0)` the excavator will be placed exactly on `BM`. In line 36, the initial horizontal orientation of the excavator cabin (yaw angle) is instantly changed by -10 degrees.

By the end of the execution of this set of statements, the initial layout of the augmented scene is defined and all CAD objects are positioned at their planned location inside the animation. Figure C.3 shows a view of the scene at animation time 0 immediately after line 36 shown in Figure C.2 is executed.

Figure C.3 – View of the AR Earthmoving Operation at Animation Time 0

Figure C.4 shows the remaining part of the ARVISCOPE animation trace file associated with the earthmoving operation. Similar to Figure C.2, the numbers inside the parentheses at the beginning of each line of the animation trace file are not parts of the trace file and they have only been added to facilitate clarity in the upcoming discussion.

```
[37] SIMTIME 5;

[38] ORIENT ExcStick VERT -20 5.00;
[39] ORIENT ExcBucket VERT -90 5.00;

[40] SIMTIME 12;

[41] ORIENT ExcBoom VERT 42 5.00;
[42] ORIENT ExcStick VERT -5 5.00;

[43] SIMTIME 19;

[44] ORIENT ExcCabin HOR 30 5.00;

[45] SIMTIME 25;

[46] ORIENT ExcBucket VERT 52 3.00;
[47] ORIENT TruckBucket2 VERT 55 3.00;

[48] SIMTIME 30;

[49] ORIENT ExcBucket VERT -52 2.00;
[50] ORIENT TruckBucket2 VERT -55 3.00;

[51] SIMTIME 32;

[52] ORIENT ExcCabin HOR -30 3.00;

[53] SIMTIME 35;

[54] ORIENT ExcBoom VERT -42 3.00;
[55] ORIENT ExcStick VERT 25 3.00;
[56] ORIENT ExcBucket VERT 90 3.00;
[57] TRAVEL Truck1 Haul 20;
[58] TRAVEL Truck2 Return 15;

[59] SIMTIME 55;

[60] POSITION Truck1 ON Return;
[61] POSITION Truck2 ON Haul;
[62] ORIENT ExcCabin HOR -10 0.00;
```

Figure C.4 – Contents of the ARVISCOPE Animation Trace File (Part 2)

Line 37 of Figure C.4 contains the control statement `SIMTIME` with an argument `5` which means that all the subsequent statements up to the next `SIMTIME` statement are executed at animation time 5. These lines are numbered as 38 and 39 in Figure C.4 and

they correspond to the rotation of the excavator stick (line 38) and the attached excavator bucket (line 39). Each of these two orientation changes takes 5 animation time units and as a result, the execution of these two lines will finish at animation time 10. In line 38, the initial vertical orientation of `ExcStick` (pitch angle) is changed -20 degrees which causes the excavator stick to lower from its initial position. In line 39, the initial vertical orientation of `ExcBucket` (pitch angle) is changed -90 degrees which causes the excavator bucket to rotate inwards and dig into the soil. Figure C.5 shows a view of the scene at animation time 10 (i.e. at the end of the execution of line 39).

Figure C.5 – View of the AR Earthmoving Operation at Animation Time 10

The execution of the ARVISCOPE animation trace file continues at line 40 which contains the control statement `SIMTIME` with an argument `12` which means that all the subsequent statements up to the next `SIMTIME` statement are executed at animation time 12. These lines are numbered as 41 and 42 in Figure C.4 and they correspond to the rotation of excavator boom (line 41) and excavator stick (line 42). Each of these two orientation changes takes 5 animation time units and as a result, the execution of these two lines will finish at animation time 17. In line 41, the initial vertical orientation of

ExcBoom (pitch angle) is changed 42 degrees which causes the excavator boom to rise from its initial position. In line 42, the current vertical orientation of ExcStick (pitch angle) is changed -5 degrees which causes the excavator stick to rotate further inward. Figure C.6 shows a view of the scene at animation time 17 (i.e. at the end of the execution of line 42).

Figure C.6 – View of the AR Earthmoving Operation at Animation Time 17

Line 43 of the ARVISCOPE animation trace file in Figure C.4 contains another control statement SIMTIME with an argument 19 which means that the following statement in line 44 is executed at animation time 19. The statement in line 44 is ORIENT which will cause the excavator cabin to change its current orientation by 30 degrees in 5 animation time units in order for the excavator bucket to reach to the dump truck at animation time 24. Figure C.7 shows a view of the scene at animation time 24 and by the end of the execution of line 44.

Figure C.7 – View of the AR Earthmoving Operation at Animation Time 24

In line 45, control statement `SIMTIME` is used again with an argument `25`. As a result, the following lines 46 and 47 will be executed immediately at animation time 25. In line 46, the excavator bucket is rotated vertically by 52 degrees in 3 animation time units in order to load the empty dump truck in the loading area. In line 47, the full dump truck, assumed to be located at the dumping area at the beginning of the visual simulation, starts unloading soil by raising its bucket by 55 degrees in 3 animation time units. Both these operations end at animation time 28. Figure C.8 shows a view of the scene at animation time 28 (i.e. at the end of the execution of line 47).

Figure C.8 – View of the AR Earthmoving Operation at Animation Time 28

Line 48 of the ARVISCOPE animation trace file in Figure C.4 contains a control statement `SIMTIME` with an argument `30`. As a result, the two subsequent lines 49 and 50 will be executed as soon as the animation clock reaches 30. In line 49, the excavator bucket changes its orientation by rotating 52 degrees inwards in 2 animation time units. This operation ends in animation time 32. Line 50 corresponds to the dump truck in the dumping area which has just finished unloading the soil and is ready to lower its bucket. The dump truck bucket starts lowering by 55 degrees which takes 3 animation time units. As a result, this operation finishes at animation time 33. Figure C.9 shows a view of the scene at animation time 32 in which the excavator bucket has already changed its orientation and the dump truck bucket is in the process of being lowered.

Figure C.10 shows a view of the scene at animation time 33 in which the bucket of the empty dump truck has been completely lowered and the dump truck is ready to leave the dumping area.

Figure C.9 – View of the AR Earthmoving Operation at Animation Time 32

Figure C.10 – View of the AR Earthmoving Operation at Animation Time 33

In line 51 of the ARVISCOPE animation trace file of Figure C.4, another `SIMTIME` statement with an argument `32` is called. This essentially means that the statement in line 52 will be executed at animation time 32. In line 52, the current orientation of the excavator cabin changes by -30 degrees which takes 3 animation time units. Figure C.11 shows a view of the scene at animation time 35.

Line 53 contains another `SIMTIME` statement with an argument `35`. As a result, lines 54 through 58 will be executed at animation time 35. Lines 54, 55, and 56 correspond to three separate orientation changes for the excavator boom, excavator stick, and excavator bucket. Each of these orientation changes take 3 animation time units. In line 57, the full dump truck is set to haul on the `Haul` route for 20 animation time units and in line 58, the empty dump truck in the dumping area is set to return to the loading area on the `Return` route in 15 time units. Figure C.12 shows a view of the scene at animation time 38. Note that in this Figure, the dump trucks are still in the process of moving from the loading area to the dumping area and vice versa.

Figure C.11 – View of the AR Earthmoving Operation at Animation Time 35

Figure C.12 – View of the AR Earthmoving Operation at Animation Time 38

As shown in line 58 of Figure C.4, the returning process takes 15 animation time units. As a result, at animation time 50, the empty dump truck has completed its trip from the dumping area to the loading area and the full dump truck is still on its way to the dumping area since the hauling process takes 20 time units and finishes at animation time 55 (line 57 in Figure C.4). Figure C.13 shows a view of the scene at animation time 50. Note that in this Figure, the empty dump truck is already at the end of the route `Return` waiting to be placed on the route `Haul` and loaded by the excavator, while the full dump truck is still in the process of moving from the loading area to the dumping area.

Figure C.13 – View of the AR Earthmoving Operation at Animation Time 50

The last group of statements in the ARVISCOPE animation trace file shown in Figure C.4 starts with a `SIMTIME` statement with an argument `55`. As a result, lines 60, 61, and 62 will be executed at animation time 55. In line 60, the full dump truck (which has now completed its trip to the dumping area) is placed on route `Return` before it can unload the soil. In line 61, the empty dump truck is placed on route `Haul` in order to be loaded by the excavator. Finally, in line 62, the current orientation of the excavator cabin is changed by -10 degrees to get the excavator to the correct position to start a new loading cycle. Figure C.14 shows a view of the scene at animation time 55. Note that this view is identical to the view shown in Figure C.3 which corresponds to animation time 0 as the underlying earthmoving operation is cyclic. As a result, the entire cycle can repeat itself for several consecutive 55 animation time unit intervals to animate a longer earthmoving operation. In order to simplify the explanation of the ARVISCOPE animation language, the stochastic nature of the process is not considered in this example. However, if the underlying DES model includes uncertainties in parameters such as activity durations, such uncertainties can be conveniently reflected in the process of animation generation and taken into account in the generated ARVISCOPE animation trace file by changing

the duration of the individual tasks inside the AR animation trace file either manually or automatically (as described later in this Appendix).

Figure C.14 – View of the AR Earthmoving Operation at Animation Time 55

Figure C.15 shows the contents of the ARVISCOPE animation trace file if another earthmoving cycle is to be animated. The statements shown in this Figure are identical to those of Figure C.4. There are two major differences between the statements in Figures C.4 and C.15. First, 55 animation time units have been added to the original argument values of the `SIMTIME` statements in Figure C.4. In addition, the original numbering used for the two dump trucks has been reversed to reflect the fact that they have switched position during the first earthmoving cycle.

```
[63] SIMTIME 60;

[64] ORIENT ExcStick VERT -20 5.00;
[65] ORIENT ExcBucket VERT -90 5.00;

[66] SIMTIME 67;

[67] ORIENT ExcBoom VERT 42 5.00;
[68] ORIENT ExcStick VERT -5 5.00;

[69] SIMTIME 74;

[70] ORIENT ExcCabin HOR 30 5.00;

[71] SIMTIME 80;

[72] ORIENT ExcBucket VERT 52 3.00;
[73] ORIENT TruckBucket1 VERT 55 3.00;

[74] SIMTIME 85;

[75] ORIENT ExcBucket VERT -52 2.00;
[76] ORIENT TruckBucket1 VERT -55 3.00;

[77] SIMTIME 87;

[78] ORIENT ExcCabin HOR -30 3.00;

[79] SIMTIME 90;

[80] ORIENT ExcBoom VERT -42 3.00;
[81] ORIENT ExcStick VERT 25 3.00;
[82] ORIENT ExcBucket VERT 90 3.00;
[83] TRAVEL Truck2 Haul 20;
[84] TRAVEL Truck1 Return 15;

[85] SIMTIME 110;

[86] POSITION Truck2 ON Return;
[87] POSITION Truck1 ON Haul;
[88] ORIENT ExcCabin HOR -10 0.00;
```

Figure C.15 – Contents of the ARVISCOPE Animation Trace File (Part 3)

Automated Generation of the Animation Trace File

As discussed in Chapter 2, an ARVISCOPE animation trace file corresponding to a simulation model can be automatically created during a simulation run. Manual generation of an animation trace file is typically not practical except in the case of simple demonstrative examples of short animated duration. Automatic generation of a trace file is the intended method since it requires less time and produces accurate results (provided that the underlying simulation model is in turn accurate itself). Automatic generation of an ARVISCOPE animation trace file requires instrumentation of a simulation model (i.e. including additional code in a simulation model). In this Section, STROBOSCOPE is used to automatically create the contents of an ARVISCOPE animation trace file as the DES model runs. STROBOSCOPE, as introduced in Chapter 2, is a programmable and extensible simulation system designed for modeling complex construction operations in detail and for the development of special purpose simulation tools [1]. Figure C.16 shows the Activity Cycle Diagram (ACD) for the earthmoving operation introduced and discussed earlier in this Appendix.

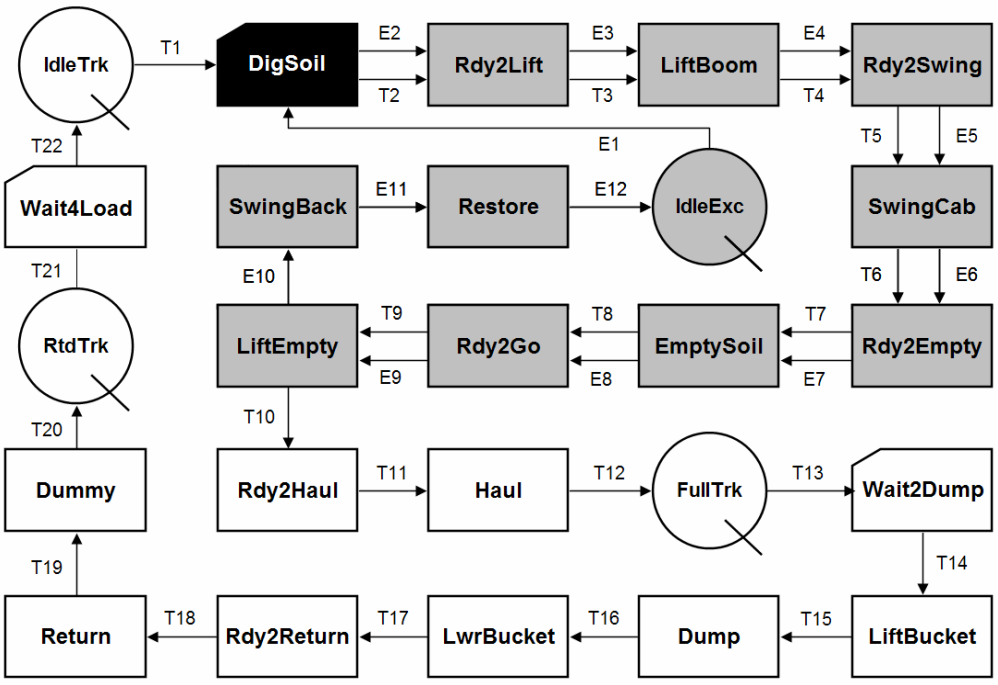


Figure C.16 – ACD for an Earthmoving Operation

The illustrated operation in this Figure is largely self-explanatory and consists of typical load-haul-dump-return cycles involving the transport of soil from the loading area to the dumping area (Figure C.1). In Figure C.16, the excavator cycle is shown in activities shaded gray and the dump truck cycle is shown in activities with a white background. Each cycle begins with the excavator digging the soil (which is shown in the activity shaded black). The intention of this Section of the Appendix is to describe the process of visualizing modeled operations in ARVISCOPE and not to describe the process of modeling itself or to design earthmoving operations using DES tools. As such, the productivity data and the system parameters of the model, such as the volume of work, required productivity, available equipment, and steps required to maximize productivity, are excluded from the discussion. The reader is referred to the STROBOSCOPE documentation introduced in Chapter 2 [1] for discussion on the elements of a DES model.

Figures C.17 through C.21 show the contents of the STROBOSCOPE file associated with the ACD of Figure C.16. The details of the STROBOSCOPE simulation language and the syntax of its statements are not described here for brevity. Readers unfamiliar with this language should refer to the STROBOSCOPE documentation introduced in Chapter 2 [1] in order to comprehend the following discussion and Figures.

```
/Definition of characterized resources.

CHARTYPE         DumpTruck              ID;
SUBTYPE          DumpTruck  DumpTruck1 1;
SUBTYPE          DumpTruck  DumpTruck2 2;

/Definition of generic resources.

GENTYPE          Excavator;

/Statements that define the queues.

QUEUE      IdleExc          Excavator;
QUEUE      RtdTrk           DumpTruck;
QUEUE      IdleTrk          DumpTruck;
QUEUE      FullTrk          DumpTruck;
```

Figure C.17 – Contents of the STROBOSCOPE Simulation Input File (Part 1)

```
/Initialize the queues.

INIT  IdleExc          1;
INIT  FullTrk          1     DumpTruck2;
INIT  RtdTrk           1     DumpTruck1;

/Statements that define the combi activities.

COMBI      DigSoil;
COMBI      Wait2Dump;
COMBI      Wait4Load;

/Statements that define the normal activities.

NORMAL     Rdy2Lift;
NORMAL     LiftBoom;
NORMAL     Rdy2Swing;
NORMAL     SwingCab;
NORMAL     Rdy2Empty;
NORMAL     EmptySoil;
NORMAL     Rdy2Go;
NORMAL     LiftEmpty;
NORMAL     SwingBack;
NORMAL     Restore;

NORMAL     Rdy2Haul;
NORMAL     Haul;
NORMAL     LiftBucket;
NORMAL     Dump;
NORMAL     LwrBucket;
NORMAL     Rdy2Return;
NORMAL     Return;
NORMAL     Dummy;
```

Figure C.18 – Contents of the STROBOSCOPE Simulation Input File (Part 2)

```
/Statements that define the links.

LINK  T1    IdleTrk    DigSoil;
LINK  T2    DigSoil    Rdy2Lift   DumpTruck;
LINK  T3    Rdy2Lift   LiftBoom   DumpTruck;
LINK  T4    LiftBoom   Rdy2Swing  DumpTruck;
LINK  T5    Rdy2Swing  SwingCab   DumpTruck;
LINK  T6    SwingCab   Rdy2Empty  DumpTruck;
LINK  T7    Rdy2Empty  EmptySoil  DumpTruck;
LINK  T8    EmptySoil  Rdy2Go     DumpTruck;
LINK  T9    Rdy2Go     LiftEmpty  DumpTruck;
LINK  T10   LiftEmpty  Rdy2Haul   DumpTruck;
LINK  T11   Rdy2Haul   Haul       DumpTruck;
LINK  T12   Haul       FullTrk;
LINK  T13   FullTrk    Wait2Dump;
LINK  T14   Wait2Dump  LiftBucket DumpTruck;
LINK  T15   LiftBucket Dump       DumpTruck;
LINK  T16   Dump       LwrBucket  DumpTruck;
LINK  T17   LwrBucket  Rdy2Return DumpTruck;
LINK  T18   Rdy2Return Return     DumpTruck;
LINK  T19   Return     Dummy      DumpTruck;
LINK  T20   Dummy      RtdTrk;
LINK  T21   RtdTrk     Wait4Load;
LINK  T22   Wait4Load  IdleTrk;

LINK  E1    IdleExc    DigSoil;
LINK  E2    DigSoil    Rdy2Lift   Excavator;
LINK  E3    Rdy2Lift   LiftBoom   Excavator;
LINK  E4    LiftBoom   Rdy2Swing  Excavator;
LINK  E5    Rdy2Swing  SwingCab   Excavator;
LINK  E6    SwingCab   Rdy2Empty  Excavator;
LINK  E7    Rdy2Empty  EmptySoil  Excavator;
LINK  E8    EmptySoil  Rdy2Go     Excavator;
LINK  E9    Rdy2Go     LiftEmpty  Excavator;
LINK  E10   LiftEmpty  SwingBack  Excavator;
LINK  E11   SwingBack  Restore    Excavator;
LINK  E12   Restore    IdleExc;
```

Figure C.19 – Contents of the STROBOSCOPE Simulation Input File (Part 3)

```
/Set attributes of the network elements.

DURATION DigSoil      '5';
DURATION Rdy2Lift     '2';
DURATION LiftBoom     '5';
DURATION Rdy2Swing    '2';
DURATION SwingCab     '5';
DURATION Rdy2Empty    '1';
DURATION EmptySoil    '3';
DURATION Rdy2Go       '2';
DURATION LiftEmpty    '2';
DURATION SwingBack    '3';
DURATION Restore      '3';
DURATION Rdy2Haul     '3';
DURATION Haul         '20';
DURATION Wait2Dump    '25';
DURATION LiftBucket   '3';
DURATION Dump         '2';
DURATION LwrBucket    '3';
DURATION Rdy2Return   '2';
DURATION Return       '15';
DURATION Wait4Load    '5';
DURATION Dummy        '5';
```

Figure C.20 – Contents of the STROBOSCOPE Simulation Input File (Part 4)

```
/Simulate the operations for 2 loading cycles.

SIMULATEUNTIL 'SimTime>110';
REPORT;
```

Figure C.21 – Contents of the STROBOSCOPE Simulation Input File (Part 5)

The simulation input file shown in Figures C.17 through C.21 must be instrumented to generate ARVISCOPE animation commands during a simulation run. As shown in Figure C.22, the output file name and path for the ARVISCOPE animation trace file have to be determined first.

```
/Set the output path for the animation trace file.
OUTFILE ARVI "..\Trace.txt";
```

Figure C.22 – Instrumenting the STROBOSCOPE Simulation Input File (Part 1)

The next step is to write the parts of the ARVISCOPE animation trace file that are independent of the activities occurring in a STROBOSCOPE simulation run. Those parts include the definition of the global reference point and routes, as well as statements required to load 3D models for the CAD objects inside the AR animation and if necessary, connecting them together to form meta-objects. As shown in Figure C.23, these statements are written exactly as they appear at the beginning of the ARVISCOPE animation trace file. Note that the term `\059` in Figure C.23 and subsequent Figures is the ASCII code for "*;*" and will be replaced by "*;*" in the ARVISCOPE animation trace file.

Finally, the lines that have to be added to the ARVISCOPE animation trace file as a result of each activity occurring within a simulation run are added to the STROBOSCOPE simulation input file. For example, the statement shown in Figure C.24, instructs STROBOSCOPE to add three lines of text to the ARVISCOPE animation trace file every time the excavator starts digging. As shown in Figure C.25, three lines will be added to the ARVISCOPE animation trace file as soon as the first instance of `DigSoil` occurs as a result of the statement shown in Figure C.24. If the duration of the activity in context is nondeterministic, each time an instance of that activity starts in the STROBOSCOPE simulation run, a different value for the activity duration will be used in

the simulation and added to the corresponding lines of the ARVISCOPE animation trace file.

```
PRINT ARVI
"SIMTIME 0.00\059

/Global Reference Point
LOADMODEL Reference BM.ac\059
OBJECT BM Reference\059
POSITION BM AT (-83.707000,42.29560,265.00)\059
/The Routes
ROUTE Return REL BM (83.400,0.000,-100.000)
                    (57.250,0.000,-89.800)
                    (10.000,0.000,-5.500)\059
ROUTE Haul REL BM (10.000,0.000,-5.500)
                  (60.000,0.000,44.100)
                  (90.000,0.000,-40.000)
                  (83.400,0.000,-100.000)\059
/The Trucks
LOADMODEL Truck RedMackBody.lwo\059
LOADMODEL TruckBucket RedMackBucket.lwo\059
ADJUST GROUP Truck AFTCLR 10\059
ADJUST GROUP Truck RGP 5\059
OBJECT Truck1 Truck\059
OBJECT TruckBucket1 TruckBucket\059
CONNECT TruckBucket1 Truck1 (-7,2.2,0)\059
OBJECT Truck2 Truck\059
OBJECT TruckBucket2 TruckBucket\059
CONNECT TruckBucket2 Truck2 (-7,2.2,0)\059
/The Excavator
LOADMODELExcBase ExcBase.ac\059
LOADMODELExcCabin ExcCabin.lwo\059
LOADMODELExcBoom ExcBoom.lwo\059
LOADMODELExcStick ExcStick.lwo\059
LOADMODELExcBucket ExcBucket.ac;
OBJECT ExcBase ExcBase\059
OBJECT ExcCabin ExcCabin\059
OBJECT ExcBoom ExcBoom\059
OBJECT ExcStick ExcStick\059
OBJECT ExcBucket ExcBucket\059
CONNECT ExcCabin ExcBase (0,0,0)\059
CONNECT ExcBoom ExcCabin (3,4.48,0)\059
CONNECT ExcStick ExcBoom (10.5,7.25,0)\059
CONNECT ExcBucket ExcStick (12.2,-9.3,0)\059
POSITION ExcBase REL BM (0.00,0.00,0.00)\059
ORIENT ExcCabin HOR -10 0.00\059\n";
```

Figure C.23 – Instrumenting the STROBOSCOPE Simulation Input File (Part 2)

```
ONSTART DigSoil PRINT ARVI
"SIMTIME %.2f\059
ORIENT ExcStick VERT -20 %.2f\059
ORIENT ExcBucket VERT -90 %.2f\059\n"
SimTime DigSoil.Duration DigSoil.Duration;
```

Figure C.24 – Instrumenting the STROBOSCOPE Simulation Input File (Part 3)

```
SIMTIME 5.00;
ORIENT ExcStick VERT -20 5.00;
ORIENT ExcBucket VERT -90 5.00;
```

Figure C.25 – Lines Added to the ARVISCOPE Animation Trace File for `DigSoil`

The statement shown in Figure C.26 instructs STROBOSCOPE to add another three lines of text to the ARVISCOPE animation trace file every time the excavator lifts its boom. As shown in Figure C.27, three lines will be added to the ARVISCOPE animation trace file as soon as the first instance of `LiftBoom` occurs as a result of the statement shown in Figure C.26.

```
ONSTART LiftBoom PRINT ARVI
"SIMTIME %.2f\059
ORIENT ExcBoom VERT 42 %.2f\059
ORIENT ExcStick VERT -5 %.2f\059\n"
SimTime LiftBoom.Duration LiftBoom.Duration;
```

Figure C.26 – Instrumenting the STROBOSCOPE Simulation Input File (Part 4)

```
SIMTIME 12.00;
ORIENT ExcBoom VERT 42 5.00;
ORIENT ExcStick VERT -5 5.00;
```

Figure C.27 – Lines Added to the ARVISCOPE Animation Trace File for `LiftBoom`

As shown in Figures C.28 and C.29, the same process should be repeated for all other activities of the ACD depicted in Figure C.16. As a result, the ARVISCOPE animation trace file will contain other lines of text that will be written out when other parts of the

modeled operation take place. Thus, the time ordered sequence of animation statements written out by all the activities in the model during a simulation run constitutes the trace file required to visualize the modeled operations in ARVISCOPE.

```
ONSTART SwingCab PRINT ARVI
"SIMTIME %.2f\059
ORIENT ExcCabin HOR 20 %.2f\059\n"
SimTime SwingCab.Duration;

ONSTART EmptySoil PRINT ARVI
"SIMTIME %.2f\059
ORIENT ExcBucket VERT 52 %.2f\059\n"
SimTime EmptySoil.Duration;

ONSTART LiftEmpty PRINT ARVI
"SIMTIME %.2f\059
ORIENT ExcBucket VERT -52 %.2f\059\n"
SimTime LiftEmpty.Duration;

ONSTART SwingBack PRINT ARVI
"SIMTIME %.2f\059
ORIENT ExcCabin HOR -35 %.2f\059\n"
SimTime SwingBack.Duration;

ONSTART Restore PRINT ARVI
"SIMTIME %.2f\059
ORIENT ExcBoom VERT -42 %.2f\059
ORIENT ExcStick VERT 25 %.2f\059
ORIENT ExcBucket VERT 90 %.2f\059
ORIENT ExcCabin HOR 5 %.2f\059\n"
SimTime Restore.Duration Restore.Duration
Restore.Duration Restore.Duration;

ONSTART Haul PRINT ARVI
"SIMTIME %.2f\059
TRAVEL Truck%.0f Haul %.2f\059\n"
SimTime Haul.DumpTruck.ID Haul.Duration;

ONSTART LiftBucket PRINT ARVI
"SIMTIME %.2f\059
ORIENT TruckBucket%.0f VERT 55 %.2f\059\n"
SimTime LiftBucket.DumpTruck.ID
LiftBucket.Duration;
```

Figure C.28 – Instrumenting the STROBOSCOPE Simulation Input File (Part 5)

```
ONSTART LwrBucket PRINT ARVI
"SIMTIME %.2f\059
ORIENT TruckBucket%.0f VERT -55 %.2f\059\n"
SimTime LwrBucket.DumpTruck.ID
LwrBucket.Duration;

ONSTART Return PRINT ARVI
"SIMTIME %.2f\059
TRAVEL Truck%.0f Return %.2f\059\n"
SimTime Return.DumpTruck.ID Return.Duration;

ONSTART Wait2Dump PRINT ARVI
"SIMTIME %.2f\059
POSITION Truck%.0f ON Return\059\n"
SimTime Wait2Dump.DumpTruck.ID;

ONSTART Wait4Load PRINT ARVI
"SIMTIME %.2f\059
POSITION Truck%.0f ON Haul\059\n"
SimTime Wait4Load.DumpTruck.ID;
```

Figure C.29 – Instrumenting the STROBOSCOPE Simulation Input File (Part 6)

The size of the generated ARVISCOPE animation trace files depends on the amount of detail modeled and the length of the simulation. The size of typical trace files will vary from a few hundred lines for simple models to several thousand lines for detailed and complex models that simulate operations over long periods of time. For example, the ARVISCOPE animation trace file for visualizing 50 cycles of the earthmoving operation discussed in this Appendix is more than 1,780 lines long. The STROBOSCOPE model required only 14 animation specific commands to do this. Since there is no limit on the size of the ARVISCOPE animation trace files that can be processed, this is not a constraining issue. Figure C.30 shows a larger portion of the ARVISCOPE animation trace file that could be generated during a simulation run.

```
SIMTIME 60.00;
ORIENT ExcStick VERT -20 5.00;
ORIENT ExcBucket VERT -90 5.00;
SIMTIME 67.00;
ORIENT ExcBoom VERT 42 5.00;
ORIENT ExcStick VERT -5 5.00;
SIMTIME 74.00;
ORIENT ExcCabin HOR 20 5.00;
SIMTIME 80.00;
ORIENT TruckBucket1 VERT 55 3.00;
SIMTIME 80.00;
ORIENT ExcBucket VERT 52 3.00;
SIMTIME 85.00;
ORIENT TruckBucket1 VERT -55 3.00;
SIMTIME 85.00;
ORIENT ExcBucket VERT -52 2.00;
SIMTIME 87.00;
ORIENT ExcCabin HOR -35 3.00;
SIMTIME 90.00;
TRAVEL Truck2 Haul 20.00;
SIMTIME 90.00;
ORIENT ExcBoom VERT -42 3.00;
ORIENT ExcStick VERT 25 3.00;
ORIENT ExcBucket VERT 90 3.00;
ORIENT ExcCabin HOR 5 3.00;
SIMTIME 90.00;
TRAVEL Truck1 Return 15.00;
SIMTIME 110.00;
POSITION Truck2 ON Return;
SIMTIME 110.00;
POSITION Truck1 ON Haul;
```

Figure C.30 – Larger Portion of the ARVISCOPE Animation Trace File

References

[1] Martinez, J. C. (1996), “STROBOSCOPE: State and Resource Based Simulation of Construction Operations”, PhD Dissertation, University of Michigan, Ann Arbor, MI.

Appendix D

Flowchart and Pseudo Code

In this Appendix, a flowchart describing the process of visualizing an AR animation created in ARVISCOPE is presented. Several pieces of pseudo code are also included to facilitate the discussion of the flowchart and to demonstrate how animation trace files written in the ARVISCOPE animation authoring language are processed to create 3D visualizations in Augmented Reality (AR).

The flowchart and pseudo code in this Appendix are not intended to be software development aids, and are only provided to supplement the discussion of the topics introduced in previous Chapters and Appendices. The presented information is also expected to help interested readers gain a better understanding of the internal processes and data flow in the developed AR platform, thus facilitating its reuse and/or modification for other application domains.

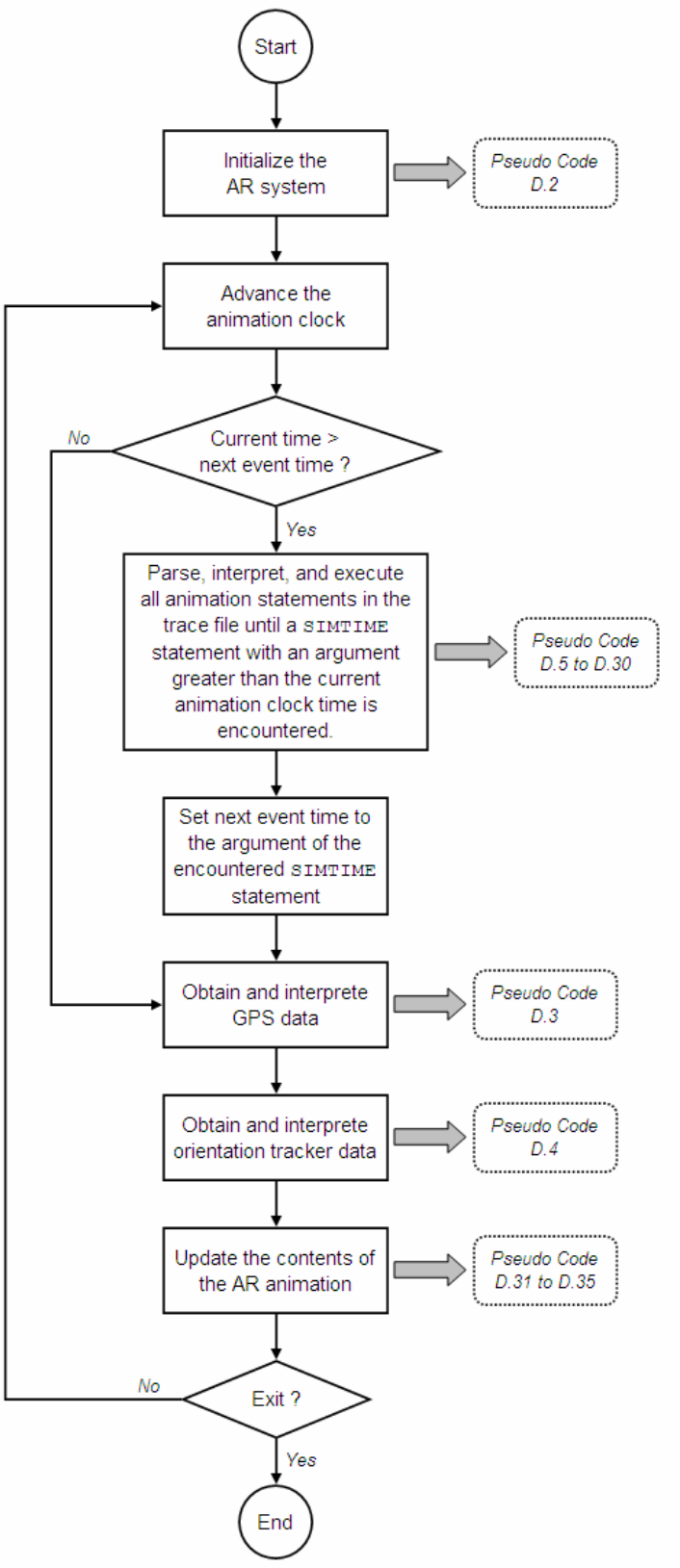


Figure D.1 – Main Loop for Visualizing an ARVISCOPE Animation

```
[1]   OPEN the serial port to which the GPS receiver is connected
[2]   SET the properties of the GPS receiver
[3]   OBTAIN and INTERPRET the first set of positioning data coming
      through the GPS receiver (Figure D.3)
[4]   OPEN the serial port to which the head orientation tracker is
      connected
[5]   SET the properties of the head orientation tracker
[6]   OBTAIN and INTERPRET the first set of head orientation data
      coming through the orientation tracker (Figure D.4)
[7]   SET the view ratio of the augmented reality animation
[8]   SET the value of “next event time” variable to 0
[9]   OPEN the ARVISCOPE animation trace file
[10]  START the video camera
```

Figure D.2 – Initializing the AR System

```
[1]   CREATE an empty list of character strings
[2]   OBTAIN and STORE a complete line of incoming GPS stream in the
      list
[3]   IF the list starts with $GPGGA
      [4]   EXTRACT element 3 of the list (latitude value)
      [5]   EXTRACT element 4 of the list (latitude direction)
      [6]   EXTRACT element 5 of the list (longitude value)
      [7]   EXTRACT element 6 of the list (longitude direction)
      [8]   EXTRACT element 10 of the list (altitude value)
      [9]   EXTRACT element 11 of the list (altitude unit)
      [10]  CONVERT extracted elements 3,5, and 10 to numbers
      [11]  IF extracted element 4 is "S"
            [12]  MULTIPLY element 3 by -1
      [13]  END IF
      [14]  IF extracted element 6 is "W"
            [15]  MULTIPLY element 5 by -1
      [16]  END IF
      [17]  IF extracted element 11 is "F"
            [18]  CONVERT element 10 from feet to meters
      [19]  END IF
      [20]  STORE elements 3,5, and 10 in a 3D vector
      [21]  OUTPUT the 3D vector
[22]  ELSE
      [23]  GOTO line [2]
[24]  END IF
```

Figure D.3 – Obtaining Longitude, Latitude, and Altitude from a GPS Receiver

```
[1]   SEND a data request to the tracker with the yaw angle ID
[2]   RECEIVE a binary data packet containing the yaw value
[3]   SEND a data request to the tracker with the pitch angle ID
[4]   RECEIVE a binary data packet containing the pitch value
[5]   SEND a data request to the tracker with the roll angle ID
[6]   RECEIVE a binary data packet containing the roll value
[7]   FOR each binary data packet received
      [8]   CALCULATE CRC value of the received data
      [9]   IF the calculated CRC does not match the received CRC
            [10]  IGNORE the binary data packet
            [11]  GOTO line [1]
      [12]  ELSE
            [13]  EXTRACT frame type (ID) of the received binary packet
            [14]  CONVERT the binary payload to numerical value
      [15]  END IF
[16]  END FOR
[17]  STORE yaw, pitch, and roll values in a 3D vector
[18]  OUTPUT the 3D vector
```

Figure D.4 – Obtaining Yaw, Pitch, and Roll from an Orientation Tracker

Syntax: ***LOADMODEL <GroupName> <3DFileName>;***

Example: ***LOADMODEL Truck CATTruck.3ds;***

```
[1]   READ all arguments following the LOADMODEL keyword
[2]   IF the number of arguments is not 2
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   ELSE
      [6]   CREATE a new entry in the map of model names vs. CAD file
            names for argument 1
      [7]   ASSIGN argument 2 to the new entry in the map of model
            names vs. CAD file names
[8]   END IF
```

Figure D.5 – Processing of LOADMODEL Statement

Syntax: ***ORIENTMODEL <GroupName> <Axis> <Degree>;***

Example: ***ORIENTMODEL Truck Y 90;***

```
[1]   READ all arguments following the ORIENTMODEL keyword
[2]   IF the number of arguments is not 3
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of model names vs. CAD
      file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 2 is not "X" OR "Y" OR "Z"
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  FIND the entry in the map of model names vs. CAD file names
      associated with argument 1
[15]  SET the local orientation of the CAD file asociated to the entry
      in the map of model names vs. CAD file names found in step [14]
      about the axis denoted in argument 2 to the value denoted by
      argument 3
```

Figure D.6 – Processing of `ORIENTMODEL` Statement

Syntax: ***CHANGEMODEL <ObjectName> <NewGroupName>;***

Example: ***CHANGEMODEL SteelBeam150 BlueSections;***

```
[1]   READ all arguments following the CHANGEMODEL keyword
[2]   IF the number of arguments is not 2
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   ELSE
      [6]   FIND the entry in the map of model names vs. CAD file names
            associated with argument 1
      [7]   ASSIGN argument 2 to the entry in the map of model names
            vs. CAD file names found in step [6]
[8]   END IF
```

Figure D.7 – Processing of `CHANGEMODEL` Statement

Syntax: ***OBJECT <ObjectName> <GroupName>;***

Example: ***OBJECT DumpTruck Truck;***

```
[1]   READ all arguments following the OBJECT keyword
[2]   IF the number of arguments is not 2
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   SEARCH the map of objects vs. pointers to CAD file names
[7]   IF argument 1 already exists in the map of objects vs. pointers
      to CAD file names
      [8]   OUTPUT ERROR
      [9]   TERMINATE
[10]  END IF
[11]  SEARCH the map of model names vs. CAD file names
[12]  IF argument 2 does not exist in the map of model names vs. CAD
      file names
      [13]  OUTPUT ERROR
      [14]  TERMINATE
[15]  END IF
[16]  CREATE a new entry in the map of object names vs. pointers to CAD
      file names for argument 1
[17]  ASSIGN argument 2 to the new entry in the map of object names vs.
      pointers to CAD file names
```

Figure D.8 – Processing of `OBJECT` Statement

Syntax: ***REMOVE <ObjectName>;***

Example: ***REMOVE DumpTruck;***

```
[1]   READ all arguments following the REMOVE keyword
[2]   IF the number of arguments is not 1
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   ELSE
      [6]   FIND the node in the AR scene graph associated with the
            object denoted by argument 1
      [7]   DELETE the node found in step [6] from the AR scene graph
[8]   END IF
```

Figure D.9 – Processing of REMOVE Statement

Syntax: ***CONNECT <ChildName> <ParentName> <AtPoint>;***

Example: ***CONNECT Boom Tower (0,35,0);***

```
[1]   READ all arguments following the CONNECT keyword
[2]   IF the number of arguments is not 3
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      pointers to CAD file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 2 does not exist in the map of object names vs.
      pointers to CAD file names
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  IF argument 1 is the same as argument 2
      [15]  OUTPUT ERROR
      [16]  TERMINATE
[17]  END IF
[18]  IF the child node denoted by argument 1 is already connected to
      the parent node denoted by argument 2 in the AR scene graph
      [19]  OUTPUT ERROR
      [20]  TERMINATE
[21]  END IF
```

Figure D.10 – Processing of CONNECT Statement (Part 1)

```
[22]  IF the object denoted by argument 1 exists in the map of object
      names vs. object positions
      [23]  REMOVE the object denoted by argument 1 from the map of
            object names vs. object positions
[24]  END IF
[25]  CREATE a child node for the object denoted by argument 1 to the
      parent node for the object denoted by argument 2 in the AR scene
      graph
[26]  UPDATE the transformation matrix of the created child node in the
      scene graph using the position vector of argument 3
```

Figure D.11 – Processing of `CONNECT` Statement (Part 2)

Syntax: ***DISCONNECT <ChildName> <ParentName>;***

Example: ***DISCONNECT SteelBeam Hook;***

```
[1]   READ all arguments following the DISCONNECT keyword
[2]   IF the number of arguments is not 2
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   SEARCH for argument 1 in the AR scene graph
[7]   IF argument 1 is a child node to te root node in the AR scene
      graph
      [8]   OUTPUT ERROR
      [9]   TERMINATE
[10]  END IF
[11]  CALCULATE the relative transformation of the object denoted by
      argument 1 to the root of the AR scene graph by multiplying the
      transformation matrices from the object to its immediate parent
      node all the way up to the node whose parent is the root node of
      the AR scene graph
[12]  REMOVE the object denoted by argument 1 from the AR scene graph
[13]  CREATE a child node to the root node of the AR scene graph for
      the object denoted by argument 1
[14]  UPDATE the transformation matrix of the created child node in the
      scene graph using the calculated value of step [11]
[15]  OBTAIN the current user position from the GPS receiver
[16]  CALCULATE the global position of the object denoted by argument 1
      using the calculated relative position from step [11] and the
      user position from step [15]
[17]  CREATE a new entry in the map of object names vs. object
      positions for argument 1
[18]  ASSIGN the calculated global position to the new entry in the map
      of object names vs. object positions
```

Figure D.12 – Processing of `DISCONNECT` Statement

Syntax: ***ROUTE <RouteName> <Points_Array_XYZ>;***

ROUTE REL <Reference> <RouteName> <Points_Array_XYZ>;

Example: ***ROUTE TruckRoad (-83.714569,42.298058,270.00)***
(-83.714553,42.298092,270.00)
(-83.714600,42.298186,270.00);

ROUTE REL Flag TruckRoad (+17.3997,1.00,-17.3348)
(+17.3997,1.00,-17.3348)
(+19.8434,1.00,-33.5500);

```
[1]   READ all arguments following the ROUTE keyword
[2]   IF the number of arguments is less than 2
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 is "REL"
      [7]   IF argument 2 does not exist in the map of object names vs.
            pointers to CAD file names
            [8]   OUTPUT ERROR
            [9]   TERMINATE
      [10]  END IF
      [11]  IF argument 3 already exists in the map of route names vs.
            lists of vectors
            [12]  OUTPUT ERROR
            [13]  TERMINATE
      [14]  END IF
      [15]  EXTRACT the current position of argument 2 from the map of
            object names vs. object positions
      [16]  FOR points on the route denoted by argument 4 to the last
            argument
            [17]  EXTRACT (X,Y,Z) values
            [18]  CALCULATE the global position of the point in
                  (Long,Lat,Alt) from the global position extracted in
                  step [15] and the relative position vector extracted
                  in step [17] using the MTM
```

Figure D.13 – Processing of ROUTE Statement (Part 1)

```
            [19]  INSERT the calcualted global position in a 3D vector
      [20]  END FOR
      [21]  CREATE a new entry in the map of route names vs. lists of
            vectors for argument 2
[22]  ELSE
      [23]  IF argument 1 already exists in the map of route names vs.
            lists of vectors
            [24]  OUTPUT ERROR
            [25]  TERMINATE
      [26]  END IF
      [27]  FOR points on the route denoted by argument 2 to the last
            argument
            [28]  EXTRACT (Long,Lat,Alt) values
            [29]  INSERT the extracted (Long,Lat,Alt) in a 3D vector
      [30]  END FOR
      [31]  CREATE a new entry in the map of route names vs. lists of
            vectors for argument 1
[32]  END IF
[33]  ADD all the calculated 3D vectors to a list of vectors
[34]  ASSIGN the created list of vectors to the new entry in the map of
      route names vs. lists of vectors
[35]  CALCULATE and STORE the length and orientation of all the line
      segments created by the vectors stored in the list of vectors
```

Figure D.14 – Processing of ROUTE Statement (Part 2)

Syntax: ***POSITION <ObjectName> AT <Point_XYZ>;***
POSITION <ObjectName> ON <RouteName>;
POSITION <ObjectName> REL <Reference> <Point_XYZ>;

Example: ***POSITION DumpTruck AT (-83.714569,42.298058,270.00);***
POSITION DumpTruck ON TruckRoad;
POSITION DumpTruck REL Flag (+17.40,1.00,-17.34);

```
[1]   READ all arguments following the POSITION keyword
[2]   IF the number of arguments is not 3 or 4
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      pointers to CAD file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  CREATE a child node to the root node of the AR scene graph for
      the object denoted by argument 1
[11]  IF argument 2 is "AT"
      [12]  CREATE a new entry in the map of object names vs. object
            positions for argument 1
      [13]  ASSIGN argument 3 to the new entry in the map of object
            names vs. object positions
[14]  ELSE IF argument 2 is "REL"
      [15]  SEARCH FOR argument 3 in the map of object names vs. object
            positions
      [16]  IF argument 3 does not exist in the map of object names vs.
            object positions
            [17]  OUTPUT ERROR
            [18]  TERMINATE
      [19]  END IF
      [20]  EXTRACT the current position of argument 3 from the map of
            object names vs. object positions
```

Figure D.15 – Processing of `POSITION` Statement (Part 1)

```
      [21]  CALCULATE the global position of the object denoted by
            argument 1 using the global position of argument 3 and the
            relative position vector denoted by argument 4
      [22]  CREATE a new entry in the map of object names vs. object
            positions for argument 1
      [23]  ASSIGN the calculated global position to the new entry in
            the map of object names vs. object positions
[24]  ELSE IF argument 2 is "ON"
      [25]  IF argument 3 does not exist in the map of route names vs.
            lists of vectors
            [26]  OUTPUT ERROR
            [27]  TERMINATE
      [28]  END IF
      [29]  FIND the position of the first point on the route from the
            list of vectors assigned to the route name denoted by
            argument 3
      [30]  CREATE a new entry in the map of object names vs. object
            positions for argument 1
      [31]  ASSIGN the global position of the first point on the route
            to the new entry in the map of object names vs. object
            positions
      [32]  OBTAIN the orientation of the first line segment on the
            route denoted by argument 3
      [33]  UPDATE the local orientation of the child node created in
            step [10] using the orientation values obtained in step
            [32]
[34]  CALCULATE the position of the object relative to the user using
      the MTM
[35]  UPDATE the translation field of the transformation matrix of the
      created child node created in step [10] using the calculated
      values of step [34]
[36]  END IF
```

Figure D.16 – Processing of `POSITION` Statement (Part 2)

Syntax: ***HRZORIENT <ObjectName> <YawValue>;***

Example: ***HRZORIENT Plane 50;***

```
[1]   READ all arguments following the HRZORIENT keyword
[2]   IF the number of arguments is not 2
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      pointers to CAD file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  SET the local orientation about Y axis of the node denoted by
      argument 1 in the scene graph equal to the value denoted by
      argument 2
```

Figure D.17 – Processing of HRZORIENT Statement

Syntax: ***VRTORIENT <ObjectName> <PitchValue>;***

Example: ***VRTORIENT Plane 30;***

```
[1]   READ all arguments following the VRTORIENT keyword
[2]   IF the number of arguments is not 2
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      pointers to CAD file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  SET the local orientation about X axis of the node denoted by
      argument 1 in the scene graph equal to the value denoted by
      argument 2
```

Figure D.18 – Processing of VRTORIENT Statement

Syntax: ***SIDEORIENT <ObjectName> <RollValue>;***

Example: ***SIDEORIENT Plane 10;***

```
[1]   READ all arguments following the SIDEORIENT keyword
[2]   IF the number of arguments is not 2
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      pointers to CAD file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  SET the local orientation about Z axis of the node denoted by
      argument 1 in the scene graph equal to the value denoted by
      argument 2
```

Figure D.19 – Processing of `SIDEORIENT` Statement

Syntax: ***ORIENT <ObjectName> <Axis> <Value> <Duration>;***

Example: ***ORIENT Boom Y 45 15;***

```
[1]   READ all arguments following the ORIENT keyword
[2]   IF the number of arguments is not 4
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      pointers to CAD file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 2 is not "X" OR "Y" OR "Z"
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  IF argument 4 is less than 0
      [15]  OUTPUT ERROR
      [16]  TERMINATE
[17]  END IF
[18]  CALCULATE the rotation speed of the object denoted by argument 1
      as the rotation angle denoted by argument 3 divided by the
      duration of orientation change denoted by argument 4
[19]  IF argument 2 is "X"
      [20]  ADD the object name to the list of vertical rotating
      objects
[21]  ELSE IF argument 2 is "Y"
      [22]  ADD the object name to the list of horizontal rotating
      objects
[23]  ELSE IF argument 2 is "Z"
      [24]  ADD the object name to the list of side rotating objects
[25]  END IF
```

Figure D.20 – Processing of ORIENT Statement

Syntax: ***ORIENTTO <ObjectName> <Axis> <Value> <Duration>;***

Example: ***ORIENTTO Boom Y 30 15;***

```
[1]   READ all arguments following the ORIENTTO keyword
[2]   IF the number of arguments is not 4
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      pointers to CAD file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 2 is not "X" OR "Y" OR "Z"
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  IF argument 4 is less than 0
      [15]  OUTPUT ERROR
      [16]  TERMINATE
[17]  END IF
[18]  IF argument 2 is "X"
      [19]  OBTAIN the vertical orientation of the object denoted by
            argument 1 from the rotation field of its transformation
            matrix
[20]  ELSE IF argument 2 is "Y"
      [21]  OBTAIN the horizontal orientation of the object denoted by
            argument 1 from the rotation field of its transformation
            matrix
[22]  ELSE IF argument 2 is "Z"
      [23]  OBTAIN the side orientation of the object denoted by
            argument 1 from the rotation field of its transformation
            matrix
[24]  END IF
```

Figure D.21 – Processing of `ORIENTTO` Statement (Part 1)

```
[25]  CALCULATE the rotation speed of the object denoted by argument 1
      as the difference in the target orientation denoted by argument 3
      and the orientation value obtained in steps [19] OR [21] OR [23]
      divided by the duration of orientation change denoted by argument
      4
[26]  IF argument 2 is "X"
      [27]  ADD the object name to the list of vertical rotating
      objects
[28]  ELSE IF argument 2 is "Y"
      [29]  ADD the object name to the list of horizontal rotating
      objects
[30]  ELSE IF argument 2 is "Z"
      [31]  ADD the object name to the list of side rotating objects
[32]  END IF
```

Figure D.22 – Processing of `ORIENTTO` Statement (Part 2)

Syntax: ***TRAVEL <ObjectName> <RouteName> <Duration>;***

Example: ***TRAVEL Truck TruckRoad 60;***

```
[1]   READ all arguments following the TRAVEL keyword
[2]   IF the number of arguments is not 3
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      object positions
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 2 does not exist in the map of route names vs. lists
      of vectors
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  IF argument 3 is less than 0
      [15]  OUTPUT ERROR
      [16]  TERMINATE
[17]  END IF
[18]  IF object denoted by argument 1 is currently in the either of the
      travelling, transferring, or shifting objects lists
      [19]  OUTPUT ERROR
      [20]  TERMINATE
[21]  END IF
[22]  CALCULATE the total length of the route denoted by argument 2
      using the map of route names vs. lists of vectors
[23]  OBTAIN the orientation of the first line segment of the route
      denoted by argument 2 using the map of route names vs. lists of
      vectors
```

Figure D.23 – Processing of TRAVEL Statement (Part 1)

[24] CALCULATE the speed of the object denoted by argument 1 as the total length of the route divided by the travel duration denoted by argument 3

[25] ADD the object name to the list of travelling objects

[26] ASSIGN the (Long,Lat,alt) of the first point on the route denoted by argument 2 to the entry in the map of object names vs. object positions associated with the object denoted by argument 1

[27] CALCULATE the position of the object relative to the user using the MTM

[28] UPDATE the translation field of the transformation matrix of the scene graph node denoted by argument 1 in the scene graph using the calculated relative position of step [27]

[29] UPDATE the local orientation of the scene graph node denoted by argument 1 in the AR scene graph using the calculated orientation of step [23]

Figure D.24 – Processing of TRAVEL Statement (Part 2)

Syntax: ***TRANSFER <ObjectName> <RouteName> <Speed>;***

Example: ***TRANSFER Truck TruckRoad 45;***

```
[1]   READ all arguments following the TRANSFER keyword
[2]   IF the number of arguments is not 3
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      object positions
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 2 does not exist in the map of route names vs. lists
      of vectors
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  IF argument 3 is less than 0
      [15]  OUTPUT ERROR
      [16]  TERMINATE
[17]  END IF
[18]  IF object denoted by argument 1 is currently in the either of the
      travelling, transferring, or shifting objects lists
      [19]  OUTPUT ERROR
      [20]  TERMINATE
[21]  END IF
[22]  CALCULATE the total length of the route denoted by argument 2
      using the map of route names vs. lists of vectors
[23]  OBTAIN the orientation of the first line segment of the route
      denoted by argument 2 using the map of route names vs. lists of
      vectors
[24]  EXTRACT and STORE the speed of the object denoted by argument 1
      from the value denoted by argument 3
```

Figure D.25 – Processing of TRANSFER Statement (Part 1)

```
[25]  ADD the object name to the list of transferring objects
[26]  ASSIGN the (Long,Lat,alt) of the first point on the route denoted
      by argument 2 to the entry in the map of object names vs. object
      positions associated with the object denoted by argument 1
[27]  CALCULATE the position of the object relative to the user using
      the MTM
[28]  UPDATE the translation field of the transformation matrix of the
      scene graph node denoted by argument 1 in the scene graph using
      the calculated relative position of step [27]
[29]  UPDATE the local orientation of the scene graph node denoted by
      argument 1 in the AR scene graph using the calculated orientation
      of step [23]
```

Figure D.26 – Processing of TRANSFER Statement (Part 2)

Syntax: ***SHIFT <ObjectName> <Vector> <Duration>;***

Example: ***SHIFT Truck (10,0,5) 15;***

```
[1]   READ all arguments following the SHIFT keyword
[2]   IF the number of arguments is not 3
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      object positions
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 3 is less than 0
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  IF object denoted by argument 1 is currently in either of the
      travelling, transferring, or shifting objects lists
      [15]  OUTPUT ERROR
      [16]  TERMINATE
[17]  END IF
[18]  CALCULATE the total length of the vector denoted by argument 2
[19]  CALCULATE the orientation of the vector denoted by argument 2
[20]  CALCULATE the speed of the object denoted by argument 1 as the
      total length of the vector calculated in step [18] divided by the
      travel duration denoted by argument 3
[21]  ADD the object name to the list of shifting objects
[22]  UPDATE the local orientation of the scene graph node denoted by
      argument 1 in the AR scene graph using the calculated orientation
      of step [19]
```

Figure D.27 – Processing of SHIFT Statement

Syntax: ***SHIFTTO <ObjectName> <POINT_XYZ> <Duration>;***

Example: ***SHIFTTO Truck (-83.714569,42.298058,270.00) 15;***

```
[1]   READ all arguments following the SHIFTTO keyword
[2]   IF the number of arguments is not 3
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      object positions
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 3 is less than 0
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  IF object denoted by argument 1 is currently in either of the
      travelling, transferring, or shifting objects lists
      [15]  OUTPUT ERROR
      [16]  TERMINATE
[17]  END IF
[18]  OBTAIN the current (Long,Lat,alt) of the object denoted by
      argument 1 from the map of object names vs. object positions
[19]  CALCULATE the shift vector using the MTM between the global
      position obtained in step [18] and the global point coordinates
      denoted by argument 2
[20]  CALCULATE the orientation of the shift vector calculated in step
      [19]
[21]  CALCULATE the speed of the object denoted by argument 1 as the
      total length of the shift vector divided by the travel duration
      denoted by argument 3
[22]  ADD the object name to the list of shifting objects
[23]  UPDATE the local orientation of the scene graph node denoted by
      argument 1 in the AR scene graph using the calculated orientation
      of step [20]
```

Figure D.28 – Processing of SHIFTTO Statement

Syntax: ***SIZE <ObjectName> <ScaleChange> <Duration>;***

Example: ***SIZE Cable (0,50,0) 15;***

```
[1]   READ all arguments following the SIZE keyword
[2]   IF the number of arguments is not 3
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      pointers to CAD file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 3 is less than 0
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  CALCULATE the speed of size change of the object denoted by
      argument 1 as the scale vector denoted by argument 2 divided by
      the duration of size change denoted by argument 3
[15]  ADD the object name to the list of scaling objects
```

Figure D.29 – Processing of SIZE Statement

Syntax: ***SIZETO <ObjectName> <ScaleTarget> <Duration>;***

Example: ***SIZETO Cable (0,55,0) 15;***

```
[1]   READ all arguments following the SIZETO keyword
[2]   IF the number of arguments is not 3
      [3]   OUTPUT ERROR
      [4]   TERMINATE
[5]   END IF
[6]   IF argument 1 does not exist in the map of object names vs.
      pointers to CAD file names
      [7]   OUTPUT ERROR
      [8]   TERMINATE
[9]   END IF
[10]  IF argument 3 is less than 0
      [11]  OUTPUT ERROR
      [12]  TERMINATE
[13]  END IF
[14]  OBTAIN the current size of the object denoted by argument 1 from
      the scale field of its transformation matrix
[15]  CALCULATE the speed of size change of the object denoted by
      argument 1 as the difference between the scale vector denoted by
      argument 2 and the scale vector obtianed in step [14] divided by
      the duration of size change denoted by argument 3
[16]  ADD the object name to the list of scaling objects
```

Figure D.30 – Processing of SIZETO Statement

```
[1]   FOR all the objects in the lists of travelling and transferring
      objects
      [2]   CALCULATE the elapsed travel time as the current animation
            time minus the time at which the moving object has started
            to move
      [3]   CALCULATE the length of the route already covered by the
            moving object as the product of the elapsed time and object
            speed
      [4]   FIND which line segment on the route the moving object is
            currently travelling on
      [5]   CALCULATE the length of the line segment already covered by
            the moving object as the total covered length of the route
            minus the sum of the lengths of all previous line segments
            on the same route
      [6]   CALCULATE the position of the moving object on the route by
            interpolating between the end points of the 3D vector of
            the line segment the object is moving on
      [7]   CALCULATE the (Long,Lat,Alt) of the moving object
      [8]   CALCULATE the position of the moving object relative to the
            user using the MTM
      [9]   OBTAIN the orientation of the line segment of the route the
            moving object is currently on using the map of route names
            vs. lists of vectors
      [10]  FIND the entry corresponding to the object in the map of
            moving objects vs. object positions
      [11]  ASSIGN the calculated (Long,Lat,Alt) to the moving object
            denoted by argument 1 in the map of moving objects vs.
            object positions
      [12]  UPDATE the translation field of the transformation matrix
            of the child node associated with the moving object in the
            AR scene graph using the calculated relative position of
            step [8]
      [13]  UPDATE the local orientation of the child node associated
            with the moving object in the AR scene graph using the
            calculated orientation of step [9]
```

Figure D.31 – Pseudo Code for the Update Process (Part 1)

```
      [14]  IF the moving object is at the end point of the route
            [15]  DELETE the object name from the list of travelling or
            transferring objects
      [16]  END IF
[17]  END FOR
[33]  FOR all the objects in the list of shifting objects
      [34]  CALCULATE the elapsed travel time as the current animation
            time minus the time in which the moving object has started
            to move
      [35]  CALCULATE the length of the shift vector already covered by
            the moving object as the product of shifting speed and the
            elapsed time
      [36]  CALCULATE the position of the moving object on the shift
            vector by interpolating between the end points of the 3D
            shift vector the object is moving on
      [37]  CALCULATE the (Long,Lat,Alt) of the moving object
      [38]  CALCULATE the position of the moving object relative to the
            user using the MTM
      [39]  FIND the entry in the map of moving objects vs. object
            positions
      [40]  ASSIGN the calculated (Long,Lat,Alt) to the moving object
            denoted by argument 1 in the map of moving objects vs.
            object positions
      [41]  UPDATE the translation field of the transformation matrix
            of the child node associated with the moving object in the
            scene graph using the calculated relative position of step
            [38]
      [42]  IF the moving object is at the end point of the shift
            vector
            [43]  DELETE the object name from the list of shifting
            objects
      [44]  END IF
[45]  END FOR
```

Figure D.32 – Pseudo Code for the Update Process (Part 2)

```
[46]  FOR all the objects in the list of vertical rotating objects
      [47]  CALCULATE the elapsed orientation change time as the
            current animation time minus the time in which the rotating
            object has started to change vertical orientation
      [48]  CALCULATE the current vertical orientation of the rotating
            object as the vertical orientation change speed times the
            elapsed time
      [49]  UPDATE the rotation field of the transformation matrix of
            the child node associated with the rotating object using
            the vertical orientation value calculated in step [48]
      [50]  IF the vertical rotating object has reached the final
            orientation
            [51]  DELETE the object name from the list of vertical
            rotating objects
      [52]  END IF
[53]  END FOR
[54]  FOR all the objects in the list of horizontal rotating objects
      [55]  CALCULATE the elapsed orientation change time as the
            current animation time minus the time in which the rotating
            object has started to change horizontal orientation
      [56]  CALCULATE the current horizontal orientation of the
            rotating object as the horizontal orientation change speed
            times the elapsed time
      [57]  UPDATE the rotation field of the transformation matrix of
            the child node associated with the rotating object using
            the horizontal orientation value calculated in step [56]
      [58]  IF the horizontal rotating object has reached the final
            orientation
            [59]  DELETE the object name from the list of horizontal
            rotating objects
      [60]  END IF
[61]  END FOR
```

Figure D.33 – Pseudo Code for the Update Process (Part 3)

```
[62]  FOR all the objects in the list of side rotating objects
      [63]  CALCULATE the elapsed orientation change time as the
            current animation time minus the time in which the rotating
            object has started to change side orientation
      [64]  CALCULATE the current side orientation of the rotating
            object as the side orientation change speed times the
            elapsed time
      [65]  UPDATE the rotation field of the transformation matrix of
            the child node associated with the rotating object using
            the side orientation value calculated in step [64]
      [66]  IF the side rotating object has reached the final
            orientation
            [67]  DELETE the object name from the list of side rotating
            objects
      [68]  END IF
[69]  END FOR
[70]  FOR all the objects in the list of scaling objects
      [71]  CALCULATE the elapsed size change time as the current
            animation time minus the time in which the object has
            started to change size
      [72]  CALCULATE the size change vector of the scaling object as
            the product of the elapsed size change time calculated in
            step [63] and the speed of size change
      [73]  CALCULATE the new size vector of the scaling object as the
            sum of the original object size vector and the calculated
            siza change vector in step [72]
      [74]  UPDATE the scale field of the transformation matrix of the
            child node associated with the scaling object using the
            calculated vector in step [73]
      [75]  IF the scaling object has reached the final size
            [76]  DELETE the object name from the list of scaling
            objects
      [77]  END IF
[78]  END FOR
```

Figure D.34 – Pseudo Code for the Update Process (Part 4)

```
[79]  OBTAIN user's global position in (Long,Lat,Alt) (Figure D.3)
[80]  FOR all the entries in tha map of object names vs. object
      positions
      [81]  EXTRACT the global position of the CAD object associated
            with the map entry
      [82]  CALCULATE the relative position of the user to the CAD
            object associated with the map entry
      [83]  UPDATE the translation field of the transformation matrix
            of the child node associated with the map entry using the
            calculated value in step [82]
[84]  END FOR
[85]  OBTAIN user's head orientation vector (Figure D.4)
[86]  CALCULATE the user's head orientation change vector as the
      orientation vector obtained in step [85] minus the previous
      user's head orientation vector
[87]  CALCULATE the AR scene orientation change vector as the user's
      head orientation change vector calculated in step [86] multiplied
      by -1
[88]  CALCULATE the AR scene orientation vector as the sum of the
      orientation change vector calculated in step [87] and the current
      orientation vector of the wrapper node in the AR scene graph
[89]  UPDATE the rotation field of the transformation matrix of the
      wrapper node in the AR scene graph using the orientation vector
      calculated in step [88]
```

Figure D.35 – Pseudo Code for the Update Process (Part 5)

Appendix E

Biography

Amir Hossein Behzadan was born in Tehran, Iran on Wednesday, January 23, 1980. He earned a Bachelor of Engineering (B.E.) degree in Civil Engineering at Sharif University of Technology (SUT), Tehran, Iran, in 2003 and a Master of Engineering (M.E.) degree in Civil and Environmental Engineering (majoring in Construction Engineering and Management) at the University of Michigan, Ann Arbor, MI, in 2005. He pursued his education in Construction Engineering and Management at the University of Michigan, Ann Arbor, MI and earned a Doctor of Philosophy (Ph.D.) degree in Civil and Environmental Engineering in 2008.