Reusable Modular Software Interfaces for Outdoor Augmented Reality Applications in Engineering

Amir H. Behzadan[1] and Vineet R. Kamat[2]

[1]A.M.ASCE, Ph.D. Candidate, Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI 48109; PH (734) 763-6825; FAX (734) 764-4292; email: abehzada@umich.edu
[2]M.ASCE, Assistant Professor, Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI 48109; PH (734) 764-4325; FAX (734) 764-4292; email: vkamat@umich.edu

Abstract

An Augmented Reality (AR) application is essentially a platform that integrates the selected hardware components via their corresponding software interfaces. In the context of this research, a software interface is a piece of code implemented to establish communication channels between a hardware device and the AR application platform. The implementation of such an interface for each piece of hardware in an AR application requires a significant amount of time and coding effort. In all engineering and scientific applications of AR developed thus far, although they largely integrate and communicate with the same set of hardware components, the effort invested in implementing the required software interfaces has been repeatedly duplicated. The objective of the presented research has been to remedy this situation by designing and implementing an extensible software framework that allows application developers to communicate with typical AR hardware components. This paper describes the design of a software interface that enables generic communication with peripheral positioning and orientation devices that are typical components in all scientific and engineering AR applications.

Introduction

Augmented Reality (AR) is a powerful visualization technique with potential applications in diverse scientific and engineering fields. Integration of AR technology 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 (Barfield and Caudell 2001).

Visualization of medical information projected onto a patient's body on the other hand, is also one of the established applications of AR technology. Traditional Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) images provide physicians with information on a totally detached display from the patient. Using AR displays allow MRI and CT images to be superimposed over the patient's anatomy which can assist in tasks such as the planning of surgical procedures (Barfield and Caudell 2001). AR has also been used in military applications. For instance, the Battlefield Augmented Reality System (BARS) is an AR system that can network multiple dismounted war fighters together with a command center. BARS also supports information gathering and human navigation for situation awareness in urban settings (Livingston et al. 2002).

Previous studies have also explored AR for a number of architecture and construction applications. For example, Webster et al. (1996) presented a system that shows locations of columns behind finished walls, and rebars inside columns. Roberts et al. (2002) used AR to overlay locations of subsurface electrical, telephone, gas, and water lines onto real world views. Both applications demonstrated the potential of AR in helping maintenance workers avoid buried infrastructure and structural elements.

Webster et al. (1996) also presented an AR system to guide workers through assembly of a space frame. Hammad et al. (2004) augmented contextual information on real views of bridges to help inspectors conduct inspections more effectively. Thomas et al. (1999) and Klinker et al. (2001) explored AR to visualize designs outdoors. Wang and Dunston (2006) have studied the potential of AR as an assistant viewer for computer-aided drawing. Kamat and El-Tawil (2005) also used AR to study the extent of horizontal displacements sustained by structural elements due to extreme loading conditions.

Every AR-based application developed and implemented in any engineering or scientific field requires the integration of a certain set of basic hardware components to work. These include video capturing, positioning, orientation tracking, user input, and display devices (Behzadan and Kamat 2005). For each of these functions, the corresponding pieces of hardware can be interactively selected from many available choices depending on the application area and desired quality of accuracy and performance (Behzadan and Kamat 2007). An AR application running on a computer is essentially a platform that integrates the selected hardware components via their corresponding software interfaces. Each interface is responsible for extracting appropriate data coming through a corresponding device using a specific data transmission protocol, and interpreting the received data for the intended purpose of the AR application.

The implementation of a software interface for each piece of hardware in an AR application requires a significant amount of time and coding effort. In all engineering and scientific applications of AR developed thus far, the implemented software interfaces have been unique to those applications. In today's era of object-oriented programming, this symbolizes a lost opportunity for significant code reusability because AR application developers have to repeatedly spend time on implementing low-level interfaces to hardware components, and correspondingly lesser time on solving higher-level domain problems using AR. In this paper, a novel approach for designing and implementing a modular, reusable software framework that allows application developers (in any field) to communicate with typical AR hardware components is presented by the authors.

The designed software interfaces are based on industry supported standard data communication formats (e.g. NMEA for GPS data), and allow users to create new AR applications without having to re-implement (i.e. recode) the underlying low-level software interfaces. The software interfaces are designed to readily support classes of hardware components (e.g. any GPS receiver that supports NMEA) instead of a specific type or model. The interfaces can be easily implemented as Dynamically Linked Libraries (DLLs) in the form of Microsoft .NET components that can be readily integrated and extended inside any AR application.

The implemented interfaces have been validated in two applications. The first one is the visualization of simulated construction processes in outdoor AR, and the second is the automated detection of contextual objects inside a user's field of view.

Description of the AR system

The AR-based visualization platform in this research consists of a set of hardware components communicating with the core application through a number of software interfaces. Figure 1 is a schematic illustration of the AR system in this research. The main hardware components as shown in this figure are a Head Mounted Display (HMD), a video camera, a GPS receiver, and a head orientation tracker. These devices are typically connected to the user in order to track and send real time changes in his or her state to the AR application.

The visualization process, as shown in this figure, starts with a real time video stream of the surrounding environment captured by the video camera and sent to the video compositor module of the AR application. Additionally, the position and orientation of the user in the field are obtained and sent to the perspective scene generating module at the same time. This module is responsible for creating a virtual viewing frustum with the user's eye located at the center point of the frustum. When the frustum is built, the application calculates the relative distance and heading angle between the user and each of the virtual CAD objects to which it has access via a computer folder (Kamat and Behzadan 2006). It then uses a set of transformation matrices (i.e. translation, rotation, and scaling) to place CAD objects in the user's viewing frustum. Furthermore, the entire virtual frustum is superimposed on top of the video stream of the real world. This entire process takes place at a very high refresh rate that the final result is seen through the HMD at real time by the user.

Required Software Interfaces

The major software interfaces required to perform the real time visualization task are those communicating with the GPS and head orientation tracking devices as well as the video camera. The HMD itself comprises of two miniature displays that play the role of a secondary monitor when connected to the computer and as a result is automatically detected by the system. The focus of this paper is on the first two software interfaces (i.e. GPS and orientation tracker).

Since the initial objective of this research work was to produce a generic and reusable software interface that can be reliably used to communicate with as many hardware components as possible, the most commonly accepted data transfer protocol for GPS devices was used. This protocol best known as National Marine Electronics Association (NMEA) standard serves as a common ground for almost all GPS devices available in market (Bennett 2006). Under this standard, a GPS device parses out real time data streams to a receiving device in a format shown in figure 2. Since the AR application only uses the positional data to build the perspective viewing frustum, the data stream as shown in figure 2 should be extracted first into its components and the numerical values of longitude, latitude, and altitude of the user should be stored in corresponding variables.

The head orientation tracker used in this research, on the other hand, sends orientation data in form of a binary packet. Figure 3 shows the structure of a data packet. The fact that the contents of each packet comes in a binary format over a serial connection makes it 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 packet data have been affected while being transferred and hence are 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, the main value for each orientation angle is obtained by converting the content of *Payload* to its numerical equivalent. In order to figure out which angle this value corresponds to, the *Frame ID* needs to be extracted. This is because each of the individual orientation angles has their corresponding *Frame ID*.

Software Interface Design

There are two important steps in communicating with each of the abovementioned hardware components: 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 a serial port. The original library was substantially modified by the authors to derive two more specific classes used for establishing port communication with the GPS receiver and orientation tracker. These classes were designed and implemented in C++. Figure 4 shows the main functionalities provided within each 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 and its main responsibility is to keep track of the events happening in the GPS thread. As soon as it receives an *EOL* (End of Line), it realizes that a complete data line has been received (Figure 2). 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.

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. 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. Figure 5 shows data transmission loops for the head orientation tracker and the GPS receiver.

Exporting the Class Libraries as Managed Code

All class libraries in the presented work are designed in a way that they can be easily exported as managed libraries that use Microsoft's .NET architecture. The authors have studied the characteristics and main features of both unmanaged and managed code and concluded that using managed type code for exporting the class libraries is more convenient and reliable. Managed code based applications are known for their faster performance, optimized memory management, automatic lifetime control of objects, code access security, minimum or none buffer overruns, cross-language integration, and platform neutrality (Microsoft 2006).

Managed code is compiled for the .NET run-time environment. It runs in the Common Language Runtime (CLR), which is the heart of the .NET framework. The CLR provides services such as security, memory management, and cross-language integration. Managed applications, written to take advantage of the features of the CLR, perform more efficiently and safely. They benefit as well from the developers' existing expertise in languages that support the .NET Framework. To export classes as a managed library, the keyword `__gc` was added to the beginning of each class definition (Figure 4).

Implementation of the Designed Communication Interfaces

The implemented interfaces have been validated in two AR applications; one for the visualization of simulated construction processes in outdoor AR, and the second for location based retrieval of contextual project information for on-site decision-making on construction sites.

To validate the software interface for an outdoor AR visualization task, the two classes were integrated into an OpenGL based AR application called UM-AR-GPS-ROVER which was developed by the authors and written entirely in C++ (Behzadan and Kamat 2006). The AR application was tested at an outdoor location at the University of Michigan by placing a number of virtual CAD objects on top of a real time video stream of the surrounding environment while the user was moving. The designed classes were implemented successfully and UM-AR-GPS-ROVER was able to construct updated augmented view of the world for the user as the user was walking around and looking at a CAD model of a steel structure both from a distance and inside. Figure 6 shows a snapshot of the test results as was displayed to the user through the HMD that was being worn.

In a separate experiment, the GPS and orientation tracker communication classes were integrated into another C++ application based on OpenSceneGraph graphical libraries. The objective of this application which was also developed by the authors' research team was to determine whether or not a real object is in the user's field of view (and therefore in context) by comparing live video streams of the real world to the contents of the user's virtual viewing frustum. The user's viewing frustum was continuously updated using the incoming positional and orientation data of a mobile user.

Again, the two classes produced satisfactory results and all their functionalities were successfully tested and validated inside the application. Figure 7 shows two snapshots of the results as produced by this C++ application. In this figure, the CAD object represents the shell of an existing building at the University of Michigan campus (i.e. G.G. Brown building). The white wire framed pyramid is in fact a representation of the user's viewing frustum which is created using the incoming GPS and orientation tracker data. The snapshots show the intersection of the CAD object with the user's viewing frustum as the user is moving around the building and looking at it from different viewing angles. At each step, the application is capable of detecting which segment of the building the user is looking at by highlighting the name of that segment in the display.

Conclusion

Although all engineering and scientific AR applications developed thus far, largely integrate and communicate with the same set of hardware components, the effort invested in implementing the required software interfaces has been repeatedly duplicated. In this paper and in order to remedy this situation by reducing the amount of work required to redevelop the common AR software interface, the authors investigated and discussed the requirements for design and implementation of a reusable generic software interface to communicate with GPS and orientation tracking devices in an AR application.

The authors successfully developed managed C++ class libraries that can be easily integrated into any AR application that uses standard GPS and orientation tracker data formats. The developed methods were tested in two separate AR applications to validate the functionalities provided by the two communication libraries. In both cases, it was found that an AR application developer can readily integrate the developed software framework as an off-the-shelf product into their AR applications and receive positional and orientation data streams required for AR registration with straightforward instantiation of C++ classes and library function calls, without the need to re-implement low level code to communicate with positioning and orientation peripheral devices.

References

Barfield, W., and Caudell, T. [editors] (2001). *Fundamentals of Wearable Computers and Augmented Reality,* Lawrence Erlbaum Associates, Mahwah, NJ.

Behzadan, A. H., and Kamat, V. R. (2007). "Georeferenced Registration of Construction Graphics in Mobile Outdoor Augmented Reality." *J. Computing in Civil Engineering,* ASCE (in press).

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

Behzadan, A. H., and Kamat, V. R. (2005). "Visualization of Construction Graphics in Outdoor Augmented Reality." *Proceedings of the 2005 Winter Simulation Conference*, Orlando, FL.

Bennett, P. (2006). "National Marine Electronics Association – FAQ." *Edge of Space Science*, http://www.eoss.org/pubs/nmeafaq.htm>(Dec. 5, 2006).

Hammad, A., Garrett, J. H., and Karimi, H. (2004). "Location-based computing for infrastructure field tasks." *Telegeoinformatics: Location-based computing and services,* 287-314. CRC Press.

Kamat, V. R., and Behzadan, A. H. (2006). "GPS and 3DOF Tracking for Georeferenced Registration of Construction Graphics in Outdoor Augmented Reality." *Proceedings of the 13th Workshop of Intelligent Computing in Engineering and Architecture (EG-ICE)*, Ascona, Switzerland.

Kamat, V. R., and El-Tawil, S. (2005). "Evaluation of Augmented Reality for Rapid Assessment of Earthquake-Induced Building Damage." *J. Computing in Civil Engineering*, ASCE (in press).

Klinker, G., Stricker, D., and Reiners, D. (2001). "Augmented Reality for exterior construction applications." In W. Barfield & T. Caudell (Eds.), *Fundamentals of Wearable Computers and Augmented Reality*. Mahwah, NJ: Lawrence Erlbaum.

Livingston, M. A., Rosenblum, L. J., Julier, S. J., Brown, D., Baillot, Y., Swan II, J. E., Gabbard, J. L., and Hix, D. (2002). "An Augmented Reality system for military operations in urban terrain." *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference*, Orlando, FL.

Microsoft (2006). *C++ At Work: Managed Code in Visual Studio 2005*. MSDN Magazine, http://msdn.microsoft.com/msdnmag/issues/06/06/CAtWork/default.aspx>(Jan. 10, 2007).

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.” *FIG XXII International Congress*, Washington, D.C.

Thomas, B., Piekarski, W., and Gunther, B. (1999). “Using Augmented Reality to Visualize Architecture Designs in an Outdoor Environment.” *Design Computing on the Net (DCNET1999)*, Sydney, Australia.

Wang, X., and Dunston, P. S. (2006). “Potential of Augmented Reality as An Assistant Viewer for Computer-Aided Drawing.” *J. Computing in Civil Engineering*, ASCE, 20(6), 437-441.

Webster, A., Feiner, S., MacIntyre, B., Massie, W., and Krueger, T. (1996). “Augmented Reality in Architectural Construction, Inspection and Renovation.” *Proceedings of 3rd Congress on Computing in Civil Engineering*, ASCE, 913-919. Reston, VA.

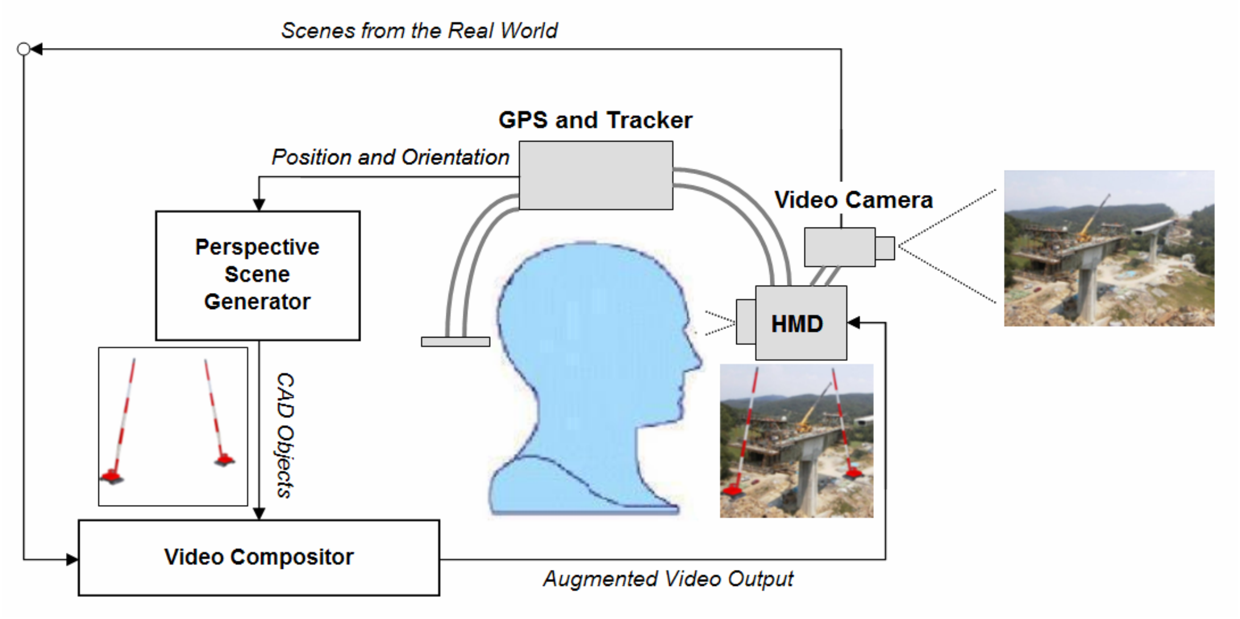


Figure 1. Main components of the AR application in the presented research

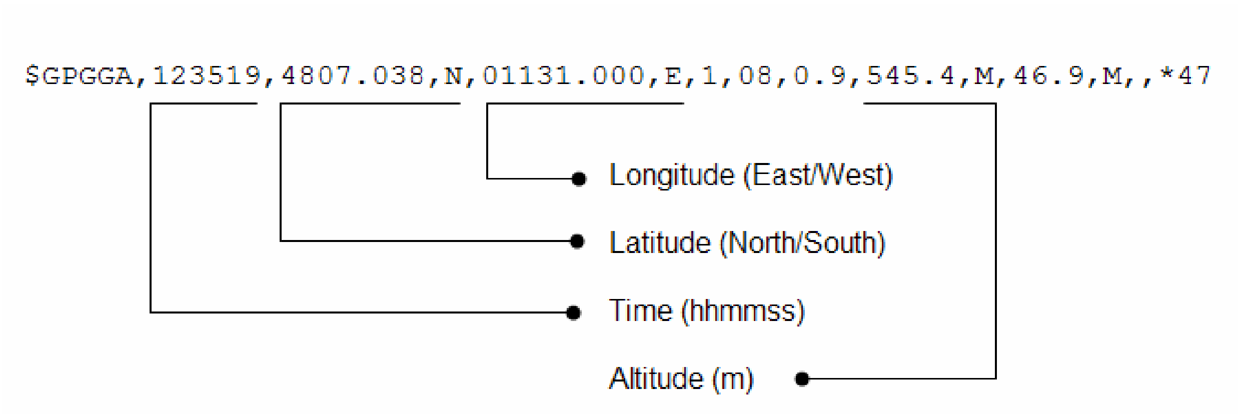


Figure 2. A sample GPS data stream following the NMEA standard

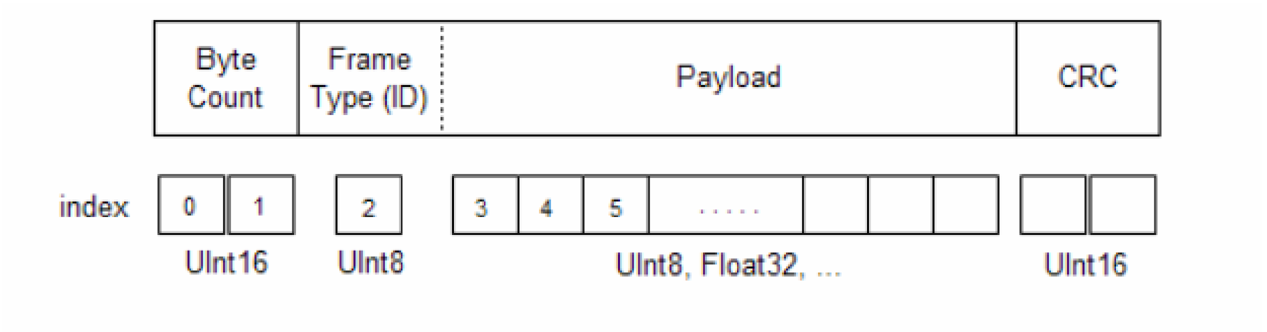
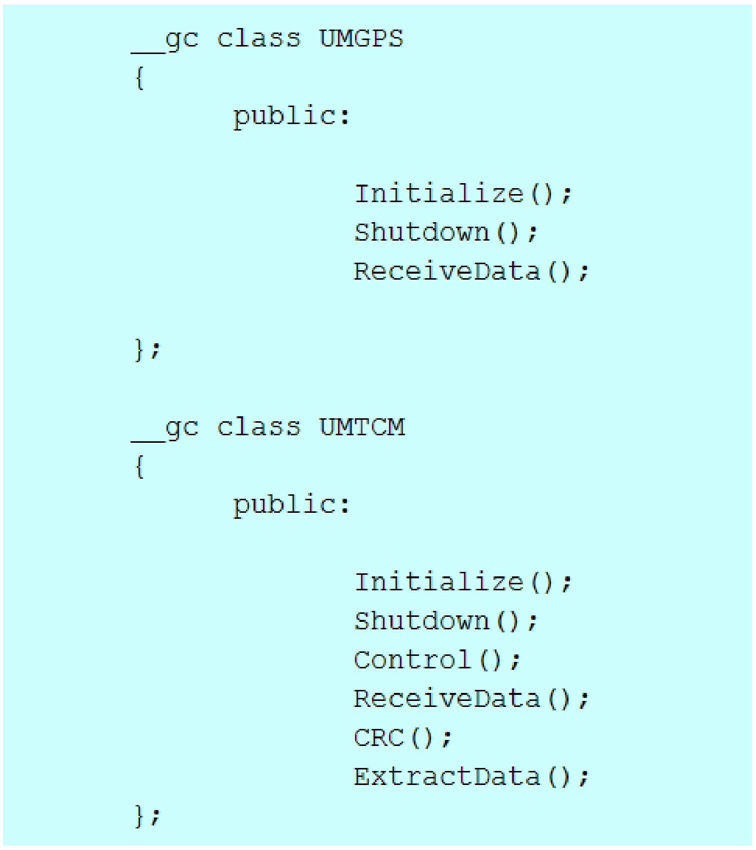
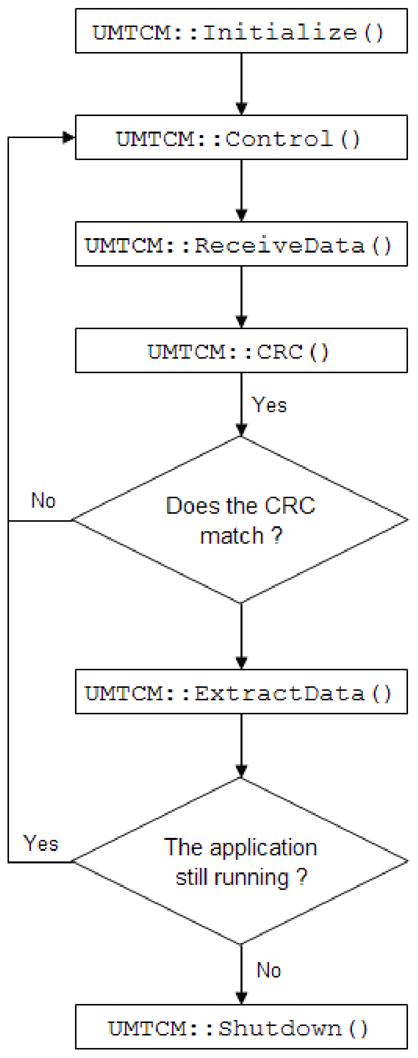


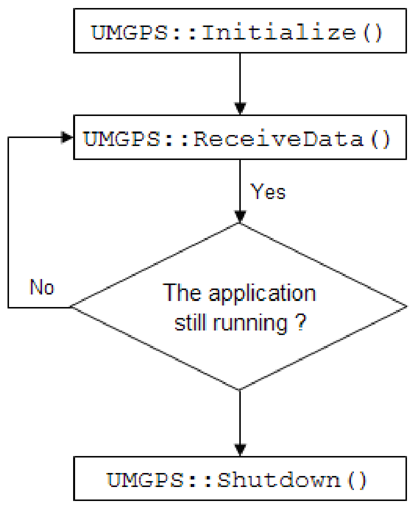
Figure 3. The structure of an orientation tracker binary data packet



```
__gc class UMGPS
{
      public:

             Initialize();
             Shutdown();
             ReceiveData();

};

__gc class UMTCM
{
      public:

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

Figure 4. GPS and orientation tracker communication classes in C++





Data Transmission Loop

Data flow diagrams for the head orientation tracker (left) and the GPS receiver (above)

Figure 5. GPS and orientation tracker data transmission loops

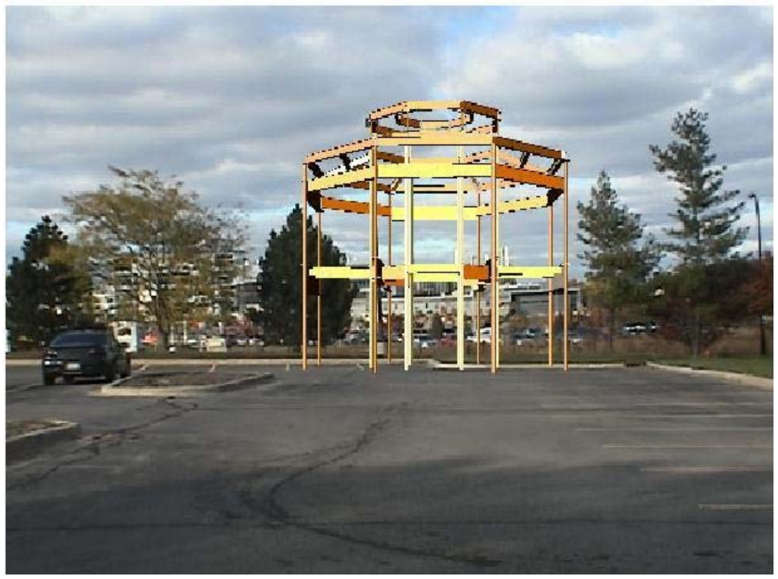
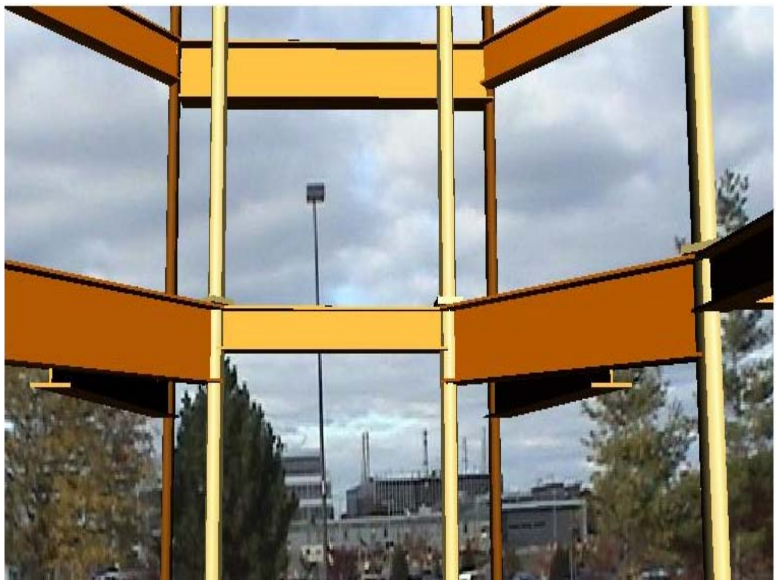
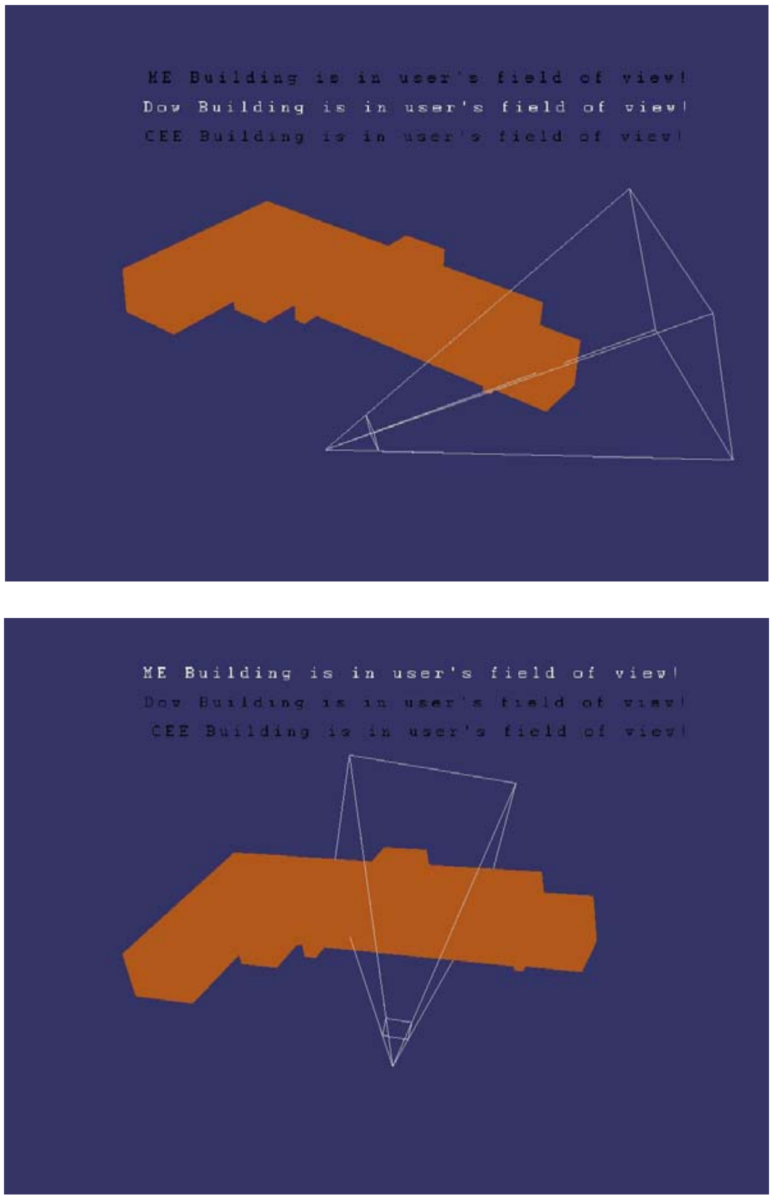



Figure 6. Application of interfaces in UM-AR-GPS-ROVER AR visualization



```
ME Building is in user's field of view!
Dow Building is in user's field of view!
CEE Building is in user's field of view!
```

```
ME Building is in user's field of view!
Dow Building is in user's field of view!
CEE Building is in user's field of view!
```

Figure 7. Application of interfaces in contextual project information retrieval