Tinmith-evo5 - An Architecture for Supporting Mobile Augmented Reality Environments

Wayne Piekarski and Bruce H. Thomas
Wearable Computer Laboratory
School of Computer and Information Science
University of South Australia
Mawson Lakes, SA, 5095, Australia
{wayne, thomas}@cs.unisa.edu.au

Abstract

This paper presents a summary of a new software architecture we have developed, known as Tinmith-evo5, which is designed as one possible methodology for writing complex AR applications. While software for 2D environments is very mature, in the 3D case there is still missing software support that we are attempting to address.

1 Introduction

Over the last few years, we have developed a number of augmented reality applications, such as the example Tinmith-Metro application [3] shown in Figure 1. This paper presents an overview of the architecture we have developed for supporting mobile augmented reality environments. We have brought together many ideas from a variety of different systems in order to implement Tinmith-evo5, and used this to implement complex user interfaces for virtual environments.

Simple AR and VR applications require support for 3D tracking devices and head mounted displays for input/output, and a renderer with a scene graph for the graphics. However, writing complex user interfaces and modelling applications requires more complex architecture, and so we desired a system with the following features: objects passing values using a data flow methodology, distributed processing and information sharing, libraries of reusable objects to implement high level applications, abstraction objects for trackers and other hardware, design for research and not novice programmers, rapid prototyping of code, and architecture that does not pay performance penalties for unused features.

For traditional 2D desktop environments, many stable design methodologies, toolkits, libraries, and input devices are available. Since virtual environments, and augmented reality in particular, are relatively new fields, the same support is not as mature for implementing applications in these areas. A number of toolkits exist for the implementation of VEs, such as Alice, World Toolkit, Coterie, and VR Juggler, which are discussed extensively in [1]. However, these toolkits do not focus on all the aspects required to implement a VE, leaving much to the programmer to decide on. Low level libraries, such as Java3D and Open Inventor, provide 3D rendering and scene graphs, but no higher level support. Figure 2 shows an example of the levels of functionality provided in Tinmith-evo5, and the previously mentioned systems only cover a subset of these. We desire much higher level frameworks that provide more human-oriented functionality, such as user interaction techniques. We propose our system as one possible solution, tailored toward our requirements outlined previously.

2 Tinmith-evo5 architecture

With our previous work, such as [2], we discovered that having large procedural software processes communicating exclusively using IPC was wasteful of resources, and so we set about producing a new more efficient design. This design contains features we know are important from our previous work, while making the other expensive features optional.

The overall design of the system is based on a data flow model, with objects processing some data and then making it available to others, similar to an Observer/Observable pattern. We decompose the application down into hundreds of simple objects that perform a specific task. The architecture was implemented with high performance being a major design requirement, as portable computers tend to have limited resources.

2.1 Low level architecture implementation

In order to implement the architecture, we identified a number of layers to support high level applications, as shown in Figure 2. Objects were written for this structure, and can be connected together to form other system components and applications.

At the lowest level, the data flow model is implemented using dynamic callback pointers that can be added and removed during execution. Higher level code supports speci-

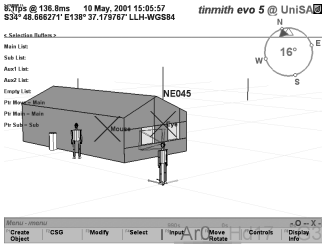

Figure 1 – External view of building model, showing situational awareness gadgets, with user interface menus and controls

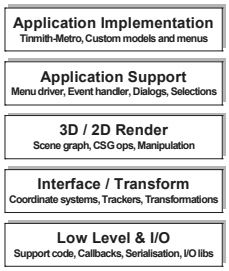

Figure 2 - Tinmith-evo5 library layering

In ISAR2001, 2nd International Symposium on Augmented Reality
October 29-30, 2001 - New York, NY, USA - Copyright (C) 2001 IEEE
Please visit http://www.tinmith.net for more information

fying connections between objects by name, and the ability to change these transparently. When the code executes a source object commit method, the callback methods are executed, allowing the destination object to process this new value. Since we use function calls and pointers by default for object communications, there is little overhead when connecting many objects.

When possible, the system is executed as a single monolithic Unix process, with no internal threads. In the optimal case, sensor data enters the system and is processed sequentially, as most objects are waiting for others to complete. For tasks requiring parallel processing, these objects can be offloaded to a separate process or thread, and then use traditional IPC to synchronise the data, with a performance penalty. Therefore, a process is a container for objects to execute within, and it is easy to move objects from single to multiple execution containers.

To connect remote processes, a serialisation object can be transparently inserted between two separated objects, converting the binary C++ object into a neutral network format. The objects are unaware the data is being transported over a network or shared memory, as they are just talking to another plug-in object. The code for serialisation and callbacks is generated using a custom program called TCC, which parses C++ class definition files – in contrast to other systems that generate code from definition files. Communications can be performed using fast UDP, or reliable TCP, depending on the requirements, and is efficient for small systems. Large scale distribution with thousands of clients (such as DIS) would require multicasting, which we do not currently support. One important feature is that there is no single marshalling point, which some systems require, and are a major performance bottleneck and single point of failure.

To facilitate easy connection of objects and referencing them in other parts of the system, we implement an object repository that allows objects to be stored and retrieved based on a key similar to a Unix hierarchical path name. The high level object code uses these path names rather than pointers to facilitate much of the dynamic nature of the system. Each object in the system is stored with a reference name, so global system-wide pointers can be avoided.

The object store is used throughout, including the run-time configuration system. Each object can be serialised to and from disk, and so at system start up these objects are created automatically and made available before the application starts. The run-time system stores values such as gadget colours and locations, strings, and debugging controls, with the ability to edit them during execution, and the program immediately reading the changes, all without the support of an interpreted language. While interpreted languages allow some flexibility in changing trivial code during execution, many architectural or other major changes still require a restart of the system. We feel our solution is useful because we have the speed benefits of a compiled language, while allowing some run time flexibility.

2.2 High level application support

Using the previously described low level architecture, it is possible to use these features to build components for the implementation of complex user applications.

Since tracker devices are an important part of VE systems, we have implemented inherent support for these. We differentiate between relative and absolute devices, combining them together using operators to resolve locations of complex articulated structures. Internally, values are stored using LLH, UTM, and ECEF coordinate systems in a variety of datums, as there is no single standard or appropriate type of coordinates to use. Internal processing is performed to allow the system to operate over large distances and varying locations without compromising the accuracy of the values.

One major component is the rendering system, which is a full hierarchical modelling system similar to Open Inventor or Java3D, with a scene graph of objects and transformation nodes, all handled by the object repository. It also contains an integrated real-time constructive solid geometry engine, allowing arbitrarily complex shapes to be constructed and carved from simple input primitives. Using the scene graph, it is possible to easily interface with other standard protocols such as DIS, allowing our system to share data with others, as demonstrated with our previous system in [2].

Currently, we have developed user interface components (known as Tinmith-Hand) that allow users to control the CSG modelling system using a pair of 3D tracked gloves. This interface is designed to allow users to walk outside and construct models, in an intuitive fashion, which match real world structures. The gloves are used to control a finger-mapped menu, with the structure contained inside the run-time object store. The model in Figure 1 was completely generated using these techniques, implemented in the Tinmith-Metro application [3].

3 Conclusion

This paper has introduced the Tinmith-evo5 design, an architecture that provides a uniform approach to implementing complex virtual environment applications. This paper demonstrates one possible method of implementing simple object oriented applications that perform a variety of complex tasks, such as user interaction techniques and 3D modelling.

4 References

[1] Bierbaum, A. and Just, C. Software Tools for Virtual Reality Application Development. In *SIGGRAPH98 Course 14 - Applied Virtual Reality,* Orlando, Fl, July 19-24, 1998.

[2] Piekarski, W., Gunther, B., and Thomas, B. Integrating Virtual and Augmented Realities in an Outdoor Application. In *2nd Int'l Workshop on Augmented Reality,* pp 45-54, San Francisco, Ca, Oct 1999.

[3] Piekarski, W. and Thomas, B. Tinmith-Metro: New Outdoor Techniques for Creating City Models with an Augmented Reality Wearable Computer. In *5th Int'l Symposium on Wearable Computers,* Zurich, Switzerland, Oct 2001.