ARTICLE

Alice and DIVER: a software architecture for building environments

RANDY F PAUSCH, University of Virginia, Charlottesville, VA, United States

MATTHEW JOHN CONWAY, University of Virginia, Charlottesville, VA, United States

ROBERT A DELINE, University of Virginia, Charlottesville, VA, United States

RICH GOSSWEILER, University of Virginia, Charlottesville, VA, United States

STEVE MIALE, University of Virginia, Charlottesville, VA, United States

Published: 01 April 1993

INTERCHI93: Conference on Human Factors in Computing
April 24 - 29, 1993
Amsterdam, The Netherlands

Conference Sponsors:
SIGCHI

https://doi.org/10.1145/259964.259976
ISBN: 0897915747

Alice and DIVER: A Software Architecture for Building Virtual Environments

Randy Pausch, Matthew Conway, Robert DeLine, Rich Gossweiler, and Steve Miale
University of Virginia, School of Engineering
Department of Computer Science
Pausch@Virginia.EDU, 804/982-2211

ABSTRACT
We are developing a rapid prototyping system built on an object-oriented, interpreted language which allows small interdisciplinary teams to quickly create and modify three-dimensional interactive simulations. Like other systems, we separate the simulation and presentation frame rates, but unlike existing systems, we do so in such a way that the application-level programmer need not understand the multi-process architecture. The system has been used for building perceptual psychology experiments, for replicating techniques developed by other researchers, and for experimenting with novel three-dimensional interaction techniques.

KEYWORDS
virtual reality, virtual environments, head-mounted display, rapid prototyping, graphical simulation, object oriented programming.

INTRODUCTION
We are developing a rapid-prototyping environment for creating immersive synthetic environments, or "virtual reality." Our rapid prototyping system is composed of two parts: Alice is the run-time system supporting application programs, and DIVER is a low-level component that provides a graphics database and manages I/O devices. Alice applications are written in an object-oriented, interpreted programming environment based on the Python language[3].

SYSTEM ARCHITECTURE
Many existing virtual environment/graphics systems, such as Sense8's WorldToolkit [4] and IRIS Inventor[6], work within a single process:

```
WHILE NOT done DO
    get input device (tracker) information
    compute a "tick" of the simulation
    update graphics to the user
END
```

This couples the graphics rendering rate (based on the movement of the user's head) to the underlying simulation's computation speed. If one were simulating a billiard table, for example, and the collision detection module were only capable of calculating 2 frames/second, then the user's visual system would be forced to update at 2 frames/second as well. If we separate the animation frames from rendering frames, we gain the ability to *render* scenes in real-time, even when the higher-level *animation* computations become complicated.

Other systems, such as the University of Alberta MR Toolkit [5] and IBM's Veridical User Environment System [2], separate simulation from rendering and interaction, but take the approach that simulation is a computation process adjunct to either a main process (in MR), or a UIMS (in the IBM system). This approach may make it possible to separate rendering and simulation, but it does not make it easy in practice. For example, MR programmers must deal with the complexity of starting the separate simulation process and managing communication with it via shared data.

We believe that application programmers should be able to take advantage of this separation without having to explicitly manage inter-process communication. In our system, Alice application programmers write code in an interpreted, object-oriented language. Changes to the state of objects are passed to the DIVER process, as shown in Figure 1. DIVER maintains enough three-dimensional geometry information to render the scene based on the user's current viewpoint. Alice programmers are not really aware of DIVER, or of using a multiple-process architecture.

The rendering and computational frame rates are separated because the input devices, such as six degree of freedom trackers, are read by the DIVER processes. In this way, the user's point of view can be updated by DIVER as quickly as DIVER can render the graphics. Alice commands arrive asynchronously over the network and update the locations of objects being altered by the Alice computation. The user's hand (driven by a glove input device) can also be rendered independently of the computation rate, as it can be updated based on the tracker information. As with hardware support for mouse cursor position, smooth, low-level operations can be performed in real time, whether or not the application is currently busy. But higher-level, more explicit events, such as mouse clicks or glove gestures, require the attention of the application.

Consider the case where the user picks up an object being animated by Alice. The location of the object will update at the simulation frame rate, while the user's hand updates at the rendering frame rate. This gives the appearance of the user's hand moving more smoothly than the object it is grasping. In this important special case, we can solve the problem by temporarily making the grasped object a geo-

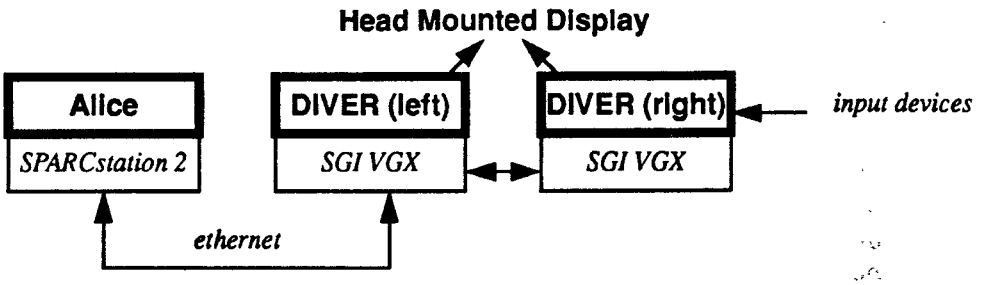

Figure 1: DIVER and Alice System Architecture

metric child of the glove/hand object, and DIVER will smoothly update both together. This technique does not extend to more general manipulation, such as having gravitational attraction between objects.

The system which is closest to our architecture is VPL's Body Electric, which has a similar process separation, but attaches the input devices to the application process, not the rendering process. This couples their rendering speed to their application processing.

SUPPORT FOR APPLICATION DEVELOPMENT

The Alice run-time system supplies a set of interactive GUI tools and a set of predefined classes which simplify application development. This portion of the system is very similar to IRIS Inventor's support library; it allows the application author to easily control text, lights, camera positions, and other common objects.

Alice supports voice input and sound output, as well as the ability to examine the direction of user gaze. As a measure of the system's power, a student was able to write a two page program where the user interacted with several independently animated objects and gave voice commands to them. The recipient of the command was determined by examining the user's gaze vector. The system has also been used to replicate the shadows and mirrors work from Brown University[1], and various locomotion techniques, including the point-of-interest logarithmic flying developed at Xerox PARC.

CONCLUSIONS AND FUTURE WORK

We have found it valuable to separate the simulation and rendering frame rates, mostly because application/simulation computation tends to have high variation in frame rate, and DIVER is able to maintain a more stable rendering rate to the user. We are pleased that undergraduates are able to create interactive, virtual environments in less than a day using our system. The presentation of this work will include several videotape examples of environments built by students.

REFERENCES

1. Kenneth P. Herndon, Robert C. Zeleznik, Daniel C. Robbins, D. Brookshire Conner, Scott S. Snibbe and Andries van Dam, *Interactive Shadows*, **Proceedings of the ACM Symposium on User Interface Software and Technology**, Monterey, California, November 15-18, 1992, pp. 1-6.

2. J. Bryan Lewis, Lawrence Koved, and Daniel T. Ling, *Dialogue Structures for Virtual Worlds*, **Proceedings of the ACM SIGCHI Human Factors in Computer Systems Conference**, April, 1991, pp 131-136.

3. Guido van Rossum, *Interactively Testing Remote Servers Using the Python Programming Language*. This paper and the Python programming language is available via anonymous ftp from `ftp.cwi.nl`.

4. Sense8 Corporation; WorldToolKit: *Virtual Reality Support Software*. 4000 Bridgeway Suite 101, Sausalito, CA 94965, telephone : (415) 331-6318.

5. Chris Shaw, Jiandog Liang, Mark Green, and Yungi Sun, *The Decoupled Simulation Model for Virtual Reality Systems*, **Proceedings of the ACM SIGCHI Human Factors in Computer Systems Conference**, May, 1992, Monterey, California, pp. 321-328.

6. Paul Strausss and Rikk Carey, *An Object-Oriented 3D Graphics Toolkit*, **Computer Graphics** (SIGGRAPH '92 Proceedings), 26:2, July 1992, pp. 341-350.