

Cover Page

Submission Category: Short Paper

Title: Alice: A Rapid Prototyping System for Building Virtual Environments

Contributors:

Matthew Conway	(804) 982-2297	conway@virginia.edu
Randy Pausch	(804) 982-2221	pausch@virginia.edu
Rich Gossweiler	(804) 982-2297	rich@virginia.edu
Tommy Burnette	(804) 982-2297	tnb2d@virginia.edu

University of Virginia, School of Engineering
Department of Computer Science
Charlottesville, VA 22903

Type of Contribution: Description of new invention

Primary Contact: Randy Pausch

Keywords: virtual reality, rapid prototyping, interpreted languages

100 word abstract: Alice is a rapid prototyping system used to create three dimensional graphics simulations like those seen in virtual reality applications. Alice uses an interpreted language called Python to implement the semantics of user actions. This interactive development environment allows users to explore many more design options than is possible in a compiled language environment.

Number of Color Illustrations: 0

Preferred Presentation Format: 1 (strongly prefer verbal)

Machine Readable Text: sent by e-mail

Alice: A Rapid Prototyping System for Building Virtual Environments

Matthew Conway, Randy Pausch, Rich Gosswiler, and Tommy Burnette

University of Virginia, School of Engineering

Department of Computer Science

Pausch@Virginia.EDU, 804/982-2211

ABSTRACT

Alice is a rapid prototyping system used to create three dimensional graphics simulations like those seen in virtual reality applications. Alice uses an interpreted language called Python to implement the semantics of user actions. This interactive development environment allows users to explore many more design options than is possible in a compiled language environment.

KEYWORDS

virtual reality, rapid prototyping, interpreted languages

CURRENT NEED FOR RAPID PROTOTYPING

The current state of 3D user interface development is reminiscent of the early days of 2D GUI development. Then, as now, programmers see a huge new design space with opportunities for important strides in user interface development. Progress remains slow, unfortunately, because the development tools are still primitive and because most systems are still hampered by the edit-test-compile cycle imposed by C and C++.

To help alleviate this problem, we have created a programming environment called *Alice* which encourages programmers to experiment and to explore three-dimensional programming techniques in an interpreted, object-oriented environment. Alice uses an interpreted language called *Python* to shorten the time between development iterations.

A BRIEF DESCRIPTION OF PYTHON

Python is a language written by Guido van Rossum of CWI in Amsterdam [Rossum 92]. Python is extremely easy for C/C++ programmers to learn up due to its familiar programming paradigm and structure. Python has attributes which make it attractive as a rapid prototyping language:

- **Python is exceedingly easy to learn.** In syntax, Python is equivalent to an “interpreted Pascal.” Seasoned C/C++ programmers take only a few hours at most to master the language.
- **Python is extensible** in C or C++, which allows programmers to speed up time consuming operations, and to avoid most of the speed complaints levied against interpreted languages. Python’s standard extensions include modules for operating system calls, graphics and widget libraries, among others. Making existing C

libraries Python-callable is straightforward.

- **Python is powerful**, sporting modern programming language features like exception handling, multiple inheritance, polymorphism, namespace management, and object oriented programming support.

The programmer interacts with Alice from inside an Emacs editing session (see Figure 1). Programmers write Python programs that specify the behaviors of three dimensional objects and the semantics of user actions (glove gestures, voice commands, etc.). Programmers then use an emacs keybinding to evaluate their code. The three dimensional simulation is then reinitialized with this new code and the programmer can evaluate the new program. Turnaround time is typically less than two seconds. With such low turn around times (typically two orders of magnitude less than the delay incurred from a typical C compilation), Alice programmers can evaluate many more program designs and interface options. Once a simulation is loaded, the programmer can continue to evaluate Python code expressions incrementally to “tweak” the simulation in any arbitrary way. In this way, the programmer can experiment in a “what-if” manner that is hard, if not impossible to do with a compiled language.

For example, we have written a simple virtual reality simulation that involved a two-handed interaction that allows the user to grab objects and either move them or stretch them,

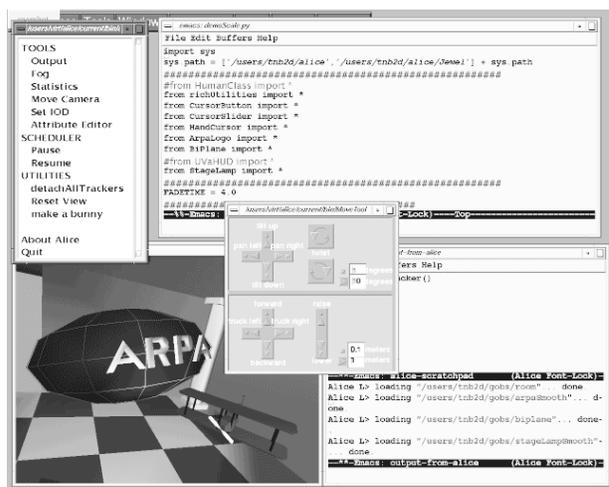


Figure 1: A typical Alice session. Shown here clockwise from upper left is the Alice main control panel, an Alice/Emacs buffer with Python code, Alice output through Emacs, the graphical output from the Alice simulation, and in the center, a 2D GUI control panel.

depending on whether the user grabs with one hand or both hands. This application took less than one day to write and the programmer explored over 100 different design options during its construction (e.g. whether the objects should scale uniformly or only in one dimension?).

In addition to specifying Python code, programmers can interact with objects via Alice's two-dimensional GUI tools, as shown in Figure 1. There are tools for setting the camera position, activating and deactivating trackers and head mounted displays, and getting statistics back from the simulation (e.g. number of frames per second). Tools themselves can be written in either C/C++ or in Python.

MAINTAINING EFFICIENCY

While Python is not as computationally efficient as C code, the speed issue is not nearly as grave in the Alice system as it is in other systems due to the way in which our system divorces the frame rate of the simulation from the frame rate of the renderer.

Many existing virtual environment/graphics systems, such as Sense8's WorldToolkit [Sense] work within a single process; 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 rendering of these scene would be forced to update at 2 frames/second as well, even if the scene was simple enough to be rendered much faster if the animation were stopped. 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. The billiard balls remain slow, but the user can look around as fast as the complexity of the scene allows.

The Alice and DIVER systems perform this separation of frame rates. DIVER acts purely as a rendering engine and Alice as a simulation which asynchronously sends commands to change the contents or structure of the database being rendered. To the Alice programmer, the program is a simulation, with rendering that happens as a side-effect.

Other systems, such as the University of Alberta MR Toolkit [Shaw 92] and IBM's Veridical User Environment System [Bryan 91], also 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.

THE ALICE SIMULATION LOOP

Traditional GUI widgets are passive: if the user chooses not to interact with the system, the widgets perform no action. In Alice, however, objects can be active or passive: objects can have *scheduled functions* associated with them that Alice calls once per simulation loop. Even if the user chooses not to interact with the three-dimensional environment, the environment is still "active" in some sense. This paradigm gives Alice programs the quality of being *visual simulations*, with the user modelled as just another active object interacting with the scene. Alice objects move across the floor, bump into other objects, destroy each other, and create new objects. In short, these 3D entities can perform any action on their own that a user might otherwise request them to do.

The active objects in Alice are implemented as a base class from which other classes are derived in an object-oriented fashion. The base object in Alice supports the usual operations of moving, scaling, rotating, setting colors, and the like. Subclasses of this have been created that can respond to voice commands and that can perform the same kind of "slow in/slow out" animation techniques demonstrated in the Self programming system [Chang 93]. We have already found that this animation class is extremely useful in providing smooth state transitions of all kinds (position, color, opacity). In the last 18 months, we have created useful classes for rotoscoping, object selection, and 3D navigation.

We are currently continuing our work in this direction by determining the set of core classes and abstractions that are most useful to application programmers.

REFERENCES

1. [Chang 93] B. Chang and D. Ungar, From Cartoons to the User Interface, UIST '93 Proceedings.
2. [Rossum 92] Guido van Rossum, *Interactively Testing Remote Servers Using the Python Programming Language*. available via anonymous ftp from `ftp.cwi.nl`
3. [Shaw 92] 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.
4. [Bryan 91] 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.
5. [Sense] Sense8 Corporation; WorldToolKit: Virtual Reality Support Software. 4000 Bridgeway Suite 101, Sausalito, CA 94965, telephone : (415) 331-6318.