Interactive Exploration in Virtual Environments

Belleman, R.G.

Publication date
2003

Appendix A

The CAVE library

The CAVE Library (CAVELib) is a set of libraries designed as a base for developing virtual reality applications for spatially immersive displays [249]. CAVELib has been in development since the debut of the CAVE in 1992, and is widely used in research and academic areas. CAVELib abstracts from the hardware used in a VR device and therefore allows portable VE applications to be built. The operating systems on which CAVELib is supported is currently IRIX, Linux, Solaris, HPUX and Microsoft Windows. The graphics interfaces supported by CAVELib are OpenGL and Performer. A schematic diagram of a CAVELib application is shown in Figure A.1. CAVELib forks separate processes for each display (i.e. each wall). Communication between processes is done through shared memory which implies that semaphore locking operators are required to enforce mutual exclusion to shared data structures and to synchronize processes. The states of the trackers and controllers are also stored in shared memory. This tracker data is acquired from the hardware through a separate daemon process known as `trackd`. Several library functions are provided by CAVELib to access this data. Each display process is synchronized and automatically renders the correct perspective for each wall. Flexible configuration files make programs written with CAVELib portable to several display and input devices without the need to recompile. The library also provides functions to share information on a virtual environment over a network. This allows users on geographically different locations to collaborate in the same virtual environment.

A.1 Interprocess communication in a CAVELib application

After initialization, CAVELib applications spawn multiple processes; one "master process" that handles user input and performs the necessary calculations that define the behaviour of the virtual environment and one or more "display processes" that do the rendering, one process for each wall. From a software design standpoint the master process describes the process under investigation by filling in (writing) a data structure that is used (read) by the display processes for representation. Conceptually this

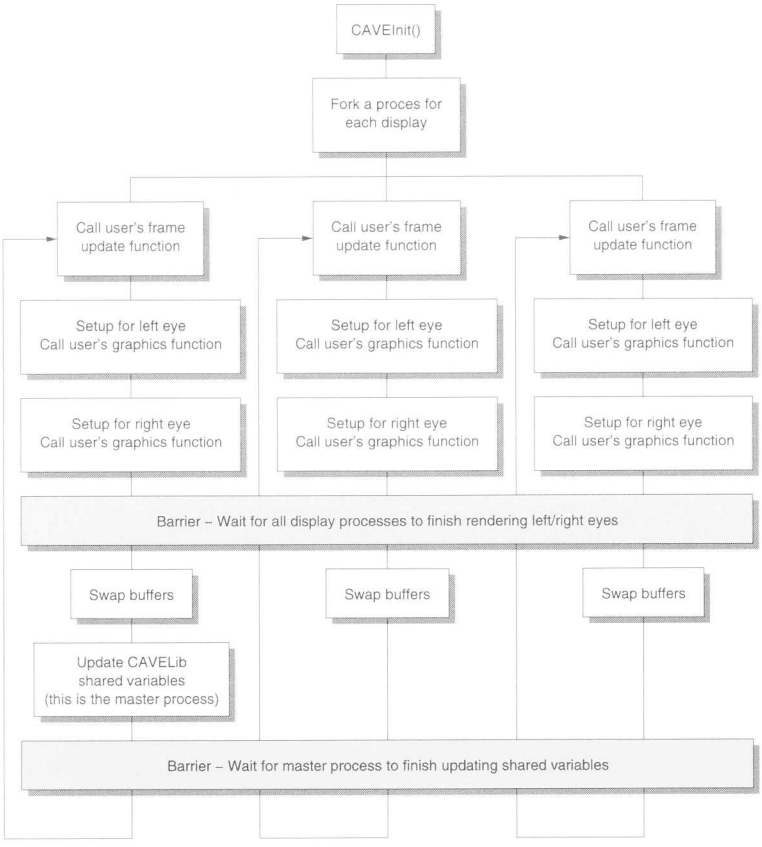

Figure A.1: Flowchart of a CAVELib program [249].

use of a shared data structure ("shared" in the sense that changes made by one process will also be available to other processes) is an easy to understand and sensible program structure to have two processes intercommunicate. Unfortunately, a design choice in CAVELib makes this a little harder than it should be.

The processes created by CAVELib are spawned using the `fork()` system call which means that, at least initially, each process is the same except for its process identifier and parent process identifier. Most UNIX operating systems optimize the creation of new processes by `fork()` by a "copy-on-write" mechanism [232]. This mechanism allows the kernel to create the processes in such a way that they can execute from the same memory areas as long as the processes perform only read accesses to this memory. When a process attempts a write access, the kernel first copies the memory areas before the process is allowed to continue. The implication of this for CAVELib applications is that the "shared" data structure proposed in the previous section will in practice not be shared between the processes at all. Instead, the main process

will have its own private copy from the moment it first writes to it while the display processes are reading from the initial, unchanged version.
The proposed method in CAVELib applications for solving this is through the use of "shared memory". As the name suggests, shared memory allows two or more processes to share a given region of memory [226]. A CAVELib application would, before any processes are spawned, allocate an area of shared memory and then pass a reference to this area to all processes that need access to it. While the main process is writing in the area, semaphores are used to prevent the display processes from accessing the area until the main process is done. The use of shared memory is a perfectly acceptable method for interprocess communications. However, shared memory is a restricted resource in some UNIX kernels. For example; on IRIX systems the maximum amount of shared memory is 4 GB but in most Linux kernels the maximum is set to 32 MB by default. While 4 GB of shared memory is sufficient for most CAVELib applications, the amount of 32 MB is quickly depleted as applications grow more complex.
Another reason why interprocess communication via shared memory is cumbersome is in the overhead that is caused when data needs to be copied from heap (i.e., "normal") memory to shared memory. This can occur when, for example, the main process uses external, third-party libraries to prepare data for rendering that return the output into heap memory (as most libraries do if they use the standard `malloc()` family of functions or the C++ `new` operator). Since the display processes will not be able to access this memory without causing a memory violation, the main process will have to copy this data into shared memory. The overhead caused by this, both in terms of the length of time needed to copy as well as the additional memory needed for the copy, can dramatically influence application performance, especially in dynamic applications where the data that is to be rendered changes frequently.
An easier solution to solve these problems would be if CAVELib spawned *threads* instead of processes [130]. The main difference between processes and threads is that threads share (most of) the resources with its parent process. This allows threads to access memory areas concurrently with its parent process and other threads without the side effects described above. VRCO apparently realized this themselves as since version 3.0 (which was in beta at the beginning of 2002[*]) CAVELib provides an option to spawn the different processes as threads instead of processes.

[*] Thanks to Matt Szymanski (VRCO) for allowing me to test a Linux beta version of a multi-threaded CAVELib.