Appendix E

Using Graphics and Share Groups

Graphics is a shared attribute among processes that share virtual address space by using the *sproc* system call. Prior to IRIX release 3.3, only a single thread of a share group could perform a `winopen()` and be allowed access to the graphics pipe. The flexibility of having any thread performing a graphics call may result in increased performance for those applications that take advantage of this feature, but there are a number of caveats that the programmer should be aware of. This appendix describes the potential problems and how to avoid them.

The graphics pipe on all architectures is a memory mapped device—that is, loads and stores to memory locations send commands and data to the hardware. Thus the graphics hardware is a region of virtual memory that is made accessible to a graphics process once it has done a `winopen()` call. The semantics of the `PR_SADDR` option of *sproc* are extended so that children created with this option share this virtual address space as well as sharing regular memory. Because access to other graphics resources (input queues, and so on) is done through open file descriptors, the `PR_SFDS` option should be enabled as well.

The resulting effect is that when one thread performs a graphics call, it is as if all threads perform the call. For example, if one process performs a `winopen()`, all threads in the share group have access to the new window. If any thread performs a `winset()`, any succeeding graphics call by any other thread is applied to the new window. A `color()` call sets the display color for all threads, and so on.

Unlike the library routines in *libmpc.a*, no code has been added to the IRIS GL to prevent simultaneous access by separate processes to either GL data structures or to the graphics pipe. Because the programming model presented by the GL is fundamentally modal, the responsibility for the definition and

protection of critical regions must be owned by the application program. For example, suppose that one process within the share group wants to perform the following sequence, each for a different polygon:

```
bgnpolygon(), v3f(), v3f(), ... endpolygon()
```

As soon as the process has made the `bgnpolygon()` call, any other process in the share group may not perform a GL call until the first process has performed the corresponding `endpolygon()` call. Thus, the application code must make the above sequence a critical region in each process, in order to ensure that the two processes do not interleave their sequences of calls. The routines described in *ussetlock* that use test-and-set style spinlocks are one effective way of enforcing the synchronization.

The effects of failing to synchronize access to the graphics pipe and associated data structures are unpredictable. At the least, some display anomalies will occur. The most catastrophic result is unexpected shutdown of the window manager and the graphics subsystem.

There are two other rules to follow when using graphics share groups. These are issues with the current implementation and may not apply to a future release. First, the process that performed the initial `winopen()` must remain alive while any thread performs GL calls. Second, unless a `foreground()` hint call is made prior to the `winopen()`, the `winopen()` call should happen prior to any call to *sproc*. This is because `winopen()` by default places a process in the background by calling *fork* and having the original parent process die. The *fork* will cause the new process to exit the share group. Delaying the *sproc* until after the `winopen()` creates the share group after graphics has been initialized.

This list summarizes the rules for sharing graphics among processes:

1. Perform a *sproc* call with sharing options `PR_SADDR` and `PR_SFDS`.
2. Treat all atomic sequences of GL calls as critical regions.
3. The process that did the first `winopen()` must not exit until all threads are finished performing GL calls.
4. Perform the *sproc* call after the `winopen()` call unless the `foreground()` call is used.