

[0134] If it is determined that the commanded effect cannot be loaded in step 434, then in step 436 the command is given a failure status, and the commanded effect is not loaded to the device. Of course, only the “play” command itself has failed; the effect data still resides in the host cache and memory model. In some embodiments, the application program can remain ignorant of the failure; this allows the application program to believe that the force effect is playing properly and to issue another play command for that effect at a later time without disruption or additional processing (and the later command may succeed); in addition, this prevents the application from overreacting to the failure. In other embodiments, it may be desirable to inform the application program of any failure to play an effect so that the application program can compensate for the failure in other ways. The application program can be provided with varying degrees of information; for example, that the effect has been cached but did not play, or that the effect simply did not play. The process continues to step 438, described below.

[0135] In an alternate embodiment, the process can mark a failed cached commanded effect as “waiting.” Effects which have a status of “waiting” can be given a high priority to be loaded if any of the effect slots on the device should open up in future iterations. The host can maintain the effect’s duration while it has a waiting status so that if an effect slot opens up, the host will know whether the waiting effect should still be output and if so, at which point in its duration. Thus, only effects which have a relatively long duration need be given a waiting status. For example, if a periodic effect having a duration of 4 seconds is waiting to be loaded on the device, the host keeps track of the duration; if 2 seconds have elapsed before an effect slot is available, the host commands the periodic effect starting at the third second. If four seconds have elapsed before an effect slot becomes available, then the host should cancel the effect since its duration has expired. In such a waiting embodiment, the process can check whether any waiting effects can be loaded to the device after an effect is untagged in step 422 or destroyed in step 414; if so, the create command of step 416 or step 423 can be sent for the waiting effect (if the waiting effect has a high enough priority), and a play command can be sent, if appropriate, to play the formerly-waiting effect. Also, in steps 430 and 434, a waiting effect can be assigned a priority or its existing priority can be increased due to the waiting status, and the waiting effect may be loaded before a currently-commanded effect if its priority is higher. It should be noted that in many implementations, such a waiting status is unnecessary, since many force effects are too short in duration to justify the extra processing required. In addition, devices having several effect slots can usually maintain realistic forces even if some force effects are discarded.

[0136] In step 438, the host can check whether any playing effect has expired, similarly to step 334 of FIG. 5. If no effects have expired, the process returns to step 406. If at least one effect has expired, then the process continues to step 440 to untag the expired effect in the host memory model. In other embodiments, steps 438 and 440 can be omitted. The process then returns to step 406.

[0137] Force effect caching on the host can also be useful in other memory management paradigms in addition to the implementation described above where the host maintains a device memory model. For example, if only the device

knows whether a commanded force effect can be stored in device memory, the device is queried by the host. If the device says that it cannot store any more effects, a driver on the host can create and cache the effect and inform the application program that its effect has been created, rather than indicating that the create command has failed.

[0138] It is important to note that the process described above preferably is implemented at a level on the host computer lower than the application program controlling the forces. The application program thus is unaware of all the effect processing that may be going on. This relieves the application program from having to determine which effects should be destroyed and which should be created at different times, and allows the developer of the application to focus on other important aspects of application and force design.

[0139] FIGS. 9a and 9b are diagrammatic illustrations of the memory of the host and device when caching a force effect as explained in FIG. 7. In the example of FIG. 9a, a device has five effect slots 480, and all five slots have been filled with a force effect as shown. Two of the effects are currently playing (tagged) as shown in column 482. The host, meanwhile, is storing a memory model 484 that includes seven force effects 485. This is because the application program has created seven force effects and believes that all seven effects have been created on the device. Therefore, two of the created force effects have been cached by the host since the device can only store five effects.

[0140] As shown in column 486, the host driver keeps track of which force effects have actually been created (loaded) on the device. The host driver also keeps track in column 488 of which force effects are currently playing, i.e. output to the user. Thus, in the example shown, the host knows that the effects in slots 1, 3, 4, 5, and 6 of the host are loaded in the available slots of the device. The slots of the host and the device need not correspond since the host loads and unloads different effects from the device during application execution; however, the host driver does need to know which slots of the device the effects are stored so that the proper index into the effect block may be sent to the device. The host also knows that the effects in slots 3 and 4 of the host are currently playing on the device. If a cached effect is commanded to be played by the application, such as the Spring effect in slot 7 of the host, then the host can examine the loaded effect slots 480 to determine which slot the Spring effect can be loaded to. For example, the Periodic1, TriggerForce, and Periodic2 effects on the device are not currently playing; since Trigger effects have a high priority, the Periodic1 or Periodic2 effect could likely be unloaded and the Spring2 effect loaded in the available slot, depending on the conditions of availability and priorities used. In addition, in some embodiments the host can also maintain a “priority” field for each effect in the model 485 to allow the comparison of priorities for loading purposes.

[0141] FIG. 9b illustrates an embodiment 490 providing the waiting feature described above as an alternative to step 436 in FIG. 7. The host keeps track of which force effects are “waiting” as shown in column 492. Thus, in the example shown, the effects in slots 1, 3, 4, 5, and 6 of the device have been loaded to the device and are all tagged, meaning they are all being currently output. The ConstantForce1 effect in slot 2 of the host has been commanded by the application program to be played, but there is no available effect slot to