StateManager Sample

Collapse All

This sample demonstrates how to implement a set of callbacks using the **ID3DXEffectStateManager** to measure the number and types of state changes in a render loop managed by the effect system. The sample also demonstrates how to use the callbacks to override the default state management, which gives an application the option of implementing a custom approach to state filtering. An application that is CPU-bound is very likely to improve performance dramatically by controlling the amount of redundant state changes that occur in a complex render loop.

Path

Source	*SDK root* \Samples\C++\Direct3D\StateManager
Executable	*SDK root* \Samples\C++\Direct3D\Bin\ *x86 or x64* \StateManager.exe

Prerequisites:

How The Sample Works

This sample renders a scene composed of several mesh objects including a gazebo, trees, rocks, terrain, and a skybox. Each mesh may be composed of multiple materials (snow, wood, etc). Each material is implemented by a different effect. Because of the scene complexity, the render sequence has many state changes to render the variety of materials and objects. The render sequence is created after calling these functions in OnCreateDevice:

- BuildSceneFromX - This method loads the scene .x file, which specifies each of the meshes and their world transform. Each mesh then loads it's corresponding materials, which consists of an effect instance and the associated textures (if any).
- QueueAndSortRenderables - This method does two things. First, it adds to the render queue all the state changes necessary to render the scene. Second, it reorders the queue according to the type of device, and the state manager.
- SetStateManager - This method propagates the state manager to each of the effects in preparation for rendering.

The sample demonstrates the benefits of filtering redundant state changes with a per device. When a device is created (**IDirect3D9::CreateDevice**), the type of device is specified with **D3DDEVTYPE**. A pure device (created with D3DCREATE_PUREDEVICE) filters a smaller subset of possible state change commands. It is very fast because is essentially streams the pipeline commands striaght to the hardware. A pure device does not do any validation of parameters, and does little or no redundant state filtering. In constrast, a non-pure device (created without D3DCREATE_PUREDEVICE) will check each state change for redundancy, and discard them. This reduces the amount of work that the device will need to perform.

The ID3DXEffectStateManager makes it possible to write custom user handlers for state changes. You can measure the number of state changes, that are happening, or even write handlers that perform custom processing (such as filtering out redundant state changes. This is precisely what the StateManager sample does. When you run the sample, it defaults to: running a pure device, and filtering redundant states with ID3DXEffectStateManager. This is done by creating a CPureDeviceStateManager class which derives from ID3DXEffectStateManager.

CPureDeviceStateManager counts the state changes, filters redundant **IDirect3DDevice9::SetRenderState**, **IDirect3DDevice9::SetSamplerState** and **IDirect3DDevice9::SetTextureStageState** commands, and invokes the corresponding Direct3D command. The number of redundant state changes that were filtered is returned to the application.

For instance:

State Changes	Filtered State Changes	Number of Rocks	% Reduction in State Changes

120	38	1	-31
7971	3856	200	-48

Another Performance Tip: Reordering The Render Sequence

In addition to filtering redundant state changes, another option to improve performance is to re-order the render sequence. This can have an impact especially on a non-pure device, or in a scenario that is more limited by the amount of matrix transforms that are set (as opposed to the number of materials used). For example, here is a render sequence that is ordered by each mesh object:

```
// Render each mesh object
For each instance of mesh x
    Setup transforms (ID3DXEffect SetMatrix)
    For each material of mesh x
        Set up material (ID3DXEffect Begin/BeginPass)
        Draw
```

This is the same sequence re-ordered by material:

```
// Render each effect to minimize state changes
For each material y
    Set up material (ID3DXEffect Begin/Begin Pass)
    For each instance of material y
      Set up transforms (ID3DXEffect SetMatrix)
      Draw
```

Each sequence is better in some situations and worse in others. An application that has lots of materials may reduce the number of redundant state changes by ordering by material.

For further information about measuring the performance of a CPU bound scenario using similar techniques to a profiler, see **Accurately Profiling Direct3D API Calls (Direct3D 9)**.

© 2010 Microsoft Corporation. All rights reserved.
Send feedback to DxSdkDoc@microsoft.com.
Version: 1962.00