EffectParam Sample

Collapse All

The EffectParam sample demonstrates two D3DX effect system features: parameter blocks and parameter sharing (see **Sharing Effect Parameters**> and **Use Parameter Blocks to Manage Effect Parameters**). Parameter blocks group multiple Setxxx API calls and associate them with an effect handle. An application can then use the parameter block to make all those state changes with a single API call. Parameter sharing synchronizes parameters in multiple effects; each time an application updates a parameter in one effect, that parameter is updated in all the other shared effects.

Path

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

How the Sample Works

This sample renders a scene that consists of seven mesh objects. The meshes are arranged in a circular ring. The user can rotate either the front mesh, or the ring to bring another mesh to the front. Each mesh has one or more materials, and each mesh can specify the effect file to render each material with.

A mesh is encapsulated by the CEffectMesh class, which wraps the **ID3DXMesh** interface. The class is mainly a helper class for setting up mesh materials. When loading a mesh, D3DX returns an array of materials used by the mesh. Each material has several attributes that define the appearance of the material, such as diffuse color, specular color, texture name, and the effect file used to render the material. D3DX does not set up materials automatically when loading meshes. It is up to the application to do so. CEffectMesh takes care of this task when an instance is created.

Each material is represented as a CMeshMaterial. This class contains three data members representing the needed data when rendering this material: the texture object, the effect object, and a handle for setting the effect's parameters.

To create a mesh, CEffectMesh::Create is called with the name of the mesh file. The sample calls **D3DXLoadMeshFromX** to create an ID3DXMesh interface and material information. Then, for each material, it checks for the associated effect instance. The effect instance (**D3DXEFFECTINSTANCE**) specifies the effect file name for rendering the material, optional effect parameters and their default values. If an effect instance exists for this material, then an **ID3DXEffect** interface is created from the effect file. Next, the sample creates a parameter block that contains the effect parameters. This is done by first calling **ID3DXEffect::BeginParameterBlock**. This call indicates to the effect object that it should record all subsequent set calls, until **ID3DXEffect::EndParameterBlock** is called. When **ID3DXEffect::EndParameterBlock** is called, the effect returns a **D3DXHANDLE** for the parameter block. From this point on, when the parameter block is applied (with **ID3DXEffect::ApplyParameterBlock**), the effect looks at the parameters inside the block, and sets the values for each of the parameters. The result is as if the application called this series of set methods. For instance, if an effect has two float4 parameters called Diffuse and Specular, a parameter block can be created to include these two parameters like this:

```
D3DXVECTOR4 vDiffuse;
D3DXVECTOR4 vSpecular;
D3DXHANDLE  hBlock;

// Initialize vDiffuse and vSpecular

pEffect->BeginParameterBlock();
pEffect->SetVector( "Diffuse", &vDiffuse );
pEffect->SetVector( "Specular", &vSpecular );
hBlock = pEffect->EndParameterBlock();
```

After this code is run, hBlock contains two parameters, Diffuse and Specular, and the recorded value of vDiffuse and vSpecular. Now, whenever the application does this:

```
// Apply the parameter block

pEffect->ApplyParameterBlock(hBlock);
```

The result is equivalent to the following, assuming vDiffuse and vSpecular still have the correct values:

```
pEffect->SetVector( "Diffuse", &vDiffuse );
pEffect->SetVector( "Specular", &vSpecular );
```

After the parameter block is created, the sample creates a new CMeshMaterial object to encapsulate the material. An effect may have more than one parameter, but CMeshMaterial only uses one D3DXHANDLE because parameter blocks allow the sample to use a single handle to represent multiple parameters.

For materials without an effect instance, a default effect is used for rendering those materials, and a parameter block is created for the standard Direct3D material attributes; that is, ambient, diffuse, specular, emissive, and specular power.

Parameter Sharing

When loading an effect, an application can optionally supply an **ID3DXEffectPool** pointer. An application passes NULL for the pool to indicate that there is no parameter sharing between effects. When a valid ID3DXEffectPool is passed, parameters declared with shared usage are shared between multiple effects. When the value of the parameter changes in one effect, every effect that shares this parameter will also see the change. To the application, it is as if there is only one instance of the parameter, used by all effects. For instance, if two effects are created with the same ID3DXEffectPool, and they both have a shared parameter named WorldViewProjection, then updating WorldViewProjection on one effect changes its value as seen by the other effect.

In the EffectParam sample, the parameters that are declared with the shared usage are the world, view, and projection matrices (or combinations of them). Because all materials of a mesh use the same transformation, it makes sense to share them among the effects so that the sample only needs to set each matrix once per mesh rendered, instead of once per material. The sample allows the user to enable or disable parameter sharing to see its impact on performance. During rendering of a mesh, the sample always sets the transformation matrices once in the default effect. When the share option is enabled, all effects of the mesh see the correct matrices and are able to transform vertices appropriately. This means that all parts of the mesh are rendered regardless of their materials (Figure 1). When the option is set to disable, the updated matrices are only seen by the default effect. All other effects never see up-to-date values in the matrices, unless they are set by other effects. Because the sample does not do this, only materials rendered by the default effect are visible (Figure 2).

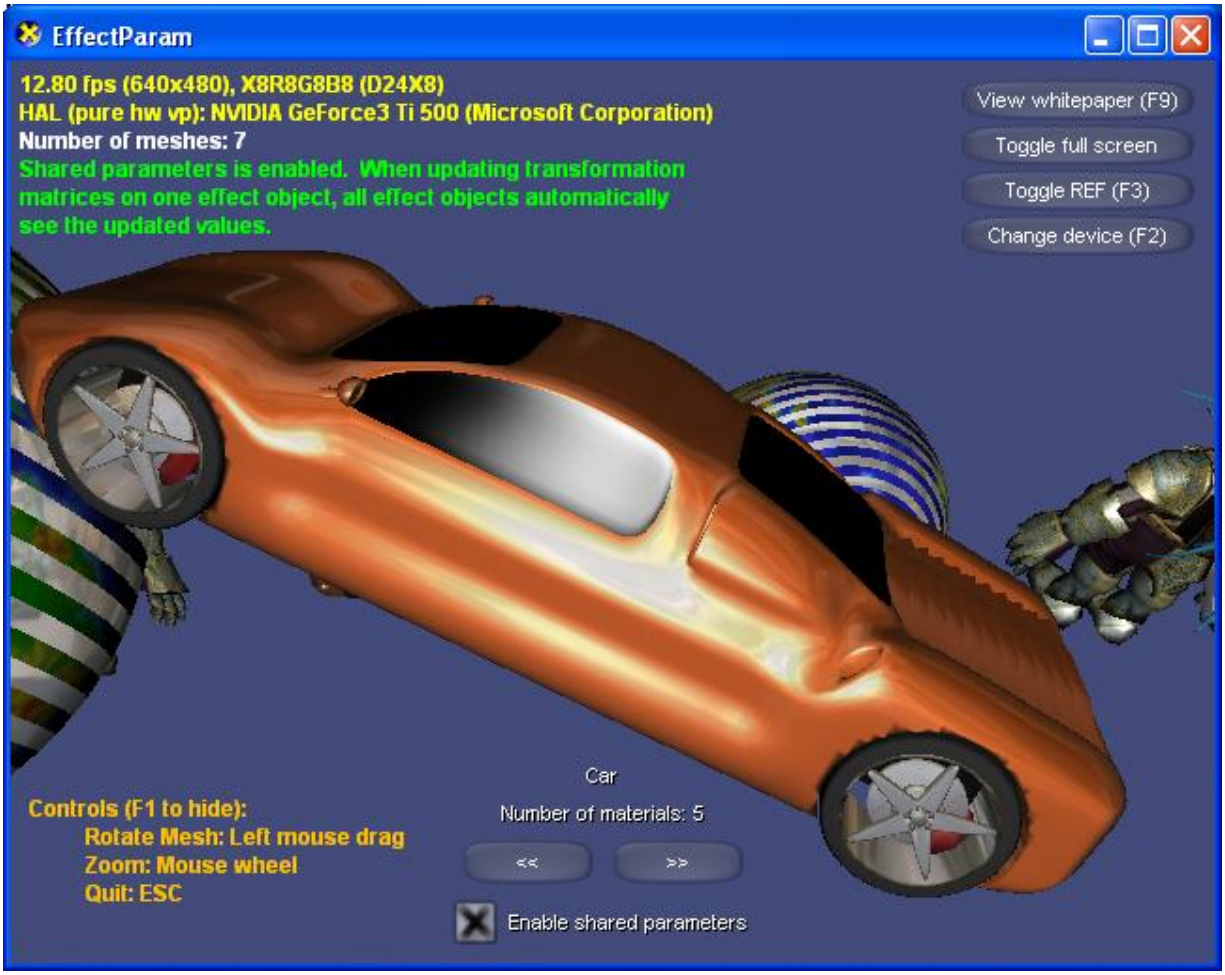

Figure 1. Parameter sharing enabled. All materials are rendered correctly.

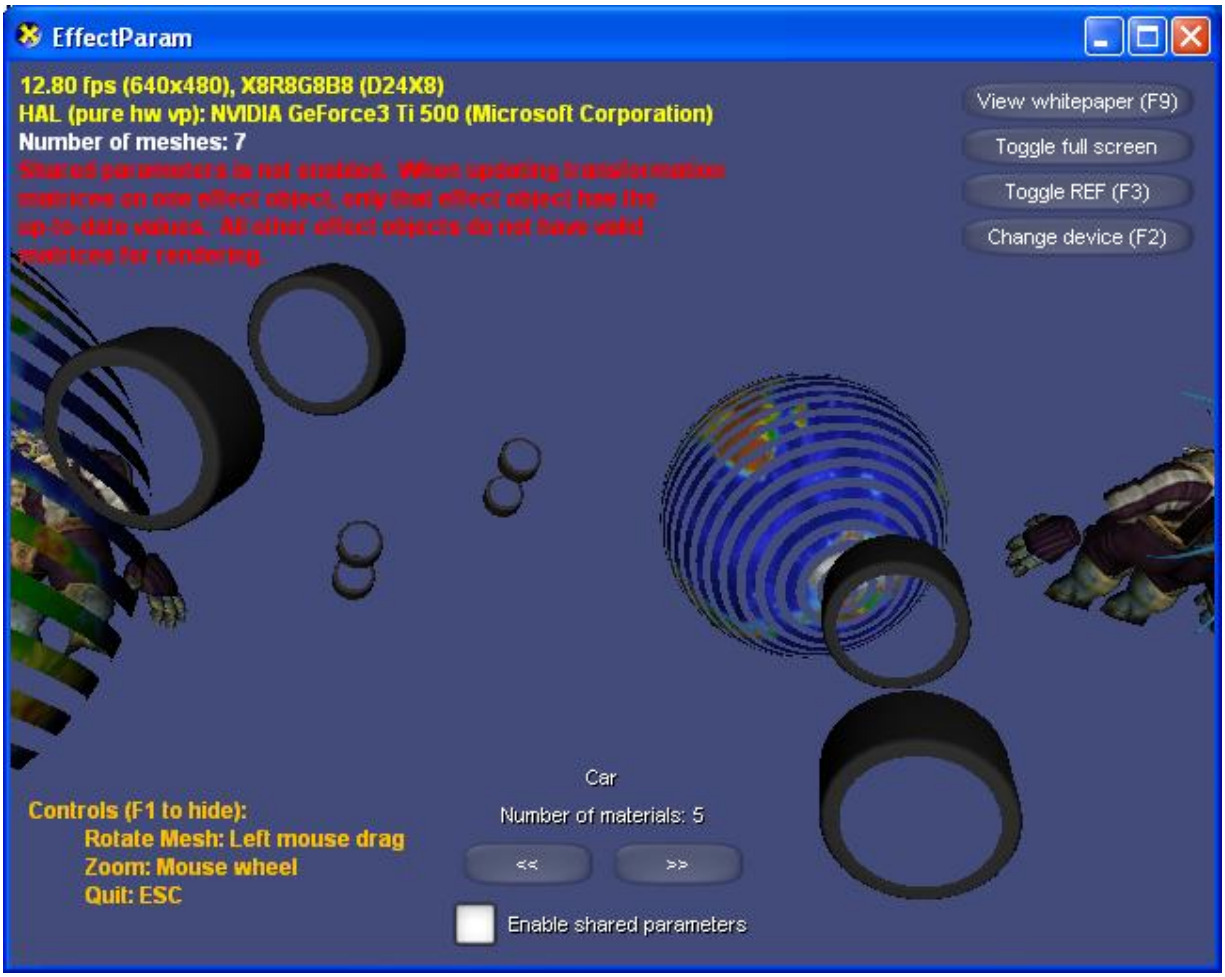

Figure 2. Parameter sharing disabled. Only materials rendered by the default effect are correctly transformed.