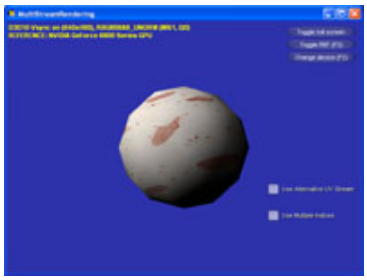
DDSWithoutD3DX Sample

See Also

Collapse All

This sample demonstrates loading a DDS file into a texture without using D3DX helper functions. This is useful for applications that do not want to take a dependency on D3DX, or as example code for creating DDS file tools.



Path

Source	*SDK root* \Samples\C++\Direct3D10\DDSWithoutD3DX
Executable	*SDK root* \Samples\C++\Direct3D10\Bin\ *platform* \DDSWithoutD3DX.exe

How the Sample Works

The interesting code for this sample is in DDSTextureLoader.cpp. There are two overloaded **CreateDDSTextureFromFile** functions, one for Direct3D 9 and one for Direct3D 10. Each of these loads the DDS data from disk and then creates a texture specific to the Direct3D version being used.

See **DDS** for more details on the file format and related structures.

Loading the DDS File

The **LoadTextureDataFromFile** function loads a DDS from disk into specific DDS structures in memory. The DDS file is split into 3 major parts. First there is the *magic number* that identifies the file as a DDS file. Second is a **DDS_HEADER** structure. Finally, there is the actual image data stored in the format specified by the **DDS_HEADER** structure.

```
// DDS files always start with the same magic number
DWORD dwMagicNumber = *(DWORD*)(*ppHeapData);
if( dwMagicNumber != DDS_MAGIC )
    return FALSE;

// setup the pointers in the process request
*ppHeader = (DDS_HEADER*)( *ppHeapData + sizeof(DWORD) );
*ppBitData  = *ppHeapData + sizeof(DWORD) + sizeof(DDS_HEADER);
*pBitSize   = FileSize.LowPart - sizeof(DWORD) - sizeof(DDS_HEADER);
```

Note that the sample makes use of the DDS.H header rather than directly referencing **DirectDraw** structures. This avoids potential problems with 64-bit native code, as well as removing the need to include DDRAW.H.

Creating a Direct3D 9 Texture

Once we have loaded the data from disk, the sample uses this data to create a device texture. In Direct3D 9 version of **CreateTextureFromDDS**, the sample gets the Direct3D 9 specific texture format from the **ddpf** member of the **DDS_HEADER** structure by decoding the **DDS_PIXELFORMAT**. The **GetD3D9Format** helper function contains conversions for some of the more common texture formats.

With the format and height, width, and mipmap information from the **DDS_HEADER**, the sample creates two Direct3D 9 texture. One is a staging texture, and one is the texture that will actually be used. Since a standard DEFAULT_POOL texture cannot be locked, a staging texture must be used in order to get the data into the DEFAULT_POOL texture. Each mipmap of staging texture is filled, and then the entire staging texture is copied to the DEFAULT_POOL texture using **UpdateTexture**.

Creating a Direct3D 10 Texture

Creating a Direct3D 10 texture is similar to create a Direct3D 9 texture. **GetDXGIFormat** is used to create a **DXGI_FORMAT** from the **ddpf** member of the **DDS_HEADER** structure by decoding the **DDS_PIXELFORMAT**.

Direct3D 10 textures can make use of the **DDS_HEADER_DXT10** extended header to define additional information about the file. The presence of this header is indicated by a **DDS_PIXELFORMAT** four-character-code of "DX10". This

extended header contains the format to use in **dxgiFormat**.

```
if ( ( pHeader->ddspf.dwFlags & DDS_FOURCC )
    && (MAKEFOURCC( 'D', 'X', '1', '0' ) == pHeader->ddspf.dwFourCC ) )
{
    DDS_HEADER_DXT10* d3d10ext = (DDS_HEADER_DXT10*)( (char*)pHeader + sizeof(DDS_HEADER) );
    desc.ArraySize = d3d10ext->arraySize;
    desc.Format = d3d10ext->dxgiFormat;
```

Some Direct3D 9 formats may not be compatible with Direct3D 10. Formats such as D3DFMT_X8R8G8B8 and D3DFMT_A8R8G8B8 will have to be converted to DXGI_FORMAT_R8G8B8A8_UNORM, which requires that some of the components be rearranged due to the more restricted color channel ordering for DXGI.

```
// swizzle if it's a format that may not be completely compatible with D3D10
if( D3DFMT_X8R8G8B8 == fmt ||
    D3DFMT_A8R8G8B8 == fmt )
{
    for( UINT i=0; i<BitSize; i+=4 )
    {
        BYTE a = pBitData[i];
        pBitData[i] = pBitData[i+2];
        pBitData[i+2] = a;
    }
}
```

When creating a Direct3D 10 texture, the sample does not need to create a staging texture. Instead a **D3D10_SUBRESOURCE_DATA** structure is created for each mipmap in the texture. The **pSysMem** member of the structure is set to the start of the data for that mipmap, and the **SysMemPitch** member is set to the stride in bytes of that mipmap's data. When creating the texture, the array of **D3D10_SUBRESOURCE_DATA** structures is passed in as the second parameter, and the texture is automatically initialized with this data. The function then creates the resource view necessary to use the texture in a shader.

See Also

DDSWithoutD3DX11 Sample