

# Tutorial 12: Single Pass Multitexturing

by Victor Saar

## Content

With DirectX 6.1 Microsoft introduced multitexturing, also known as texture blending, to his API. Direct3D supports the blending of up to eight textures on a primitive in a single pass through the use of texture stages. You can apply one texture to all of these stages. Each texture stage takes two arguments and performs a blending operation between them, both defined by the application. There are several texture arguments and texture operation flags usable. For a complete list please take a look in the SDK documentation. With the help of these arguments you can access results of a previous texture stage, which is used to perform multitexturing. All texture operations are possible for color and alpha values.

In this tutorial we will do something called dark mapping, sometimes also called light mapping. We will take a base texture and modulate it with a dark map, which includes the lighting information of the appropriate primitive.

Before we can use multitexturing we have to check how many simultaneous textures are supported by the device. Then we set the texture arguments and texture operation flags needed to perform multitexturing. At the end we load the two textures and simply render them.

## d3ddefs.h

When we want two textures on a single primitive we also need two sets of texture coordinates, of course. `D3DFVF_TEX2` defines that we have two sets. If you want more textures just replace the number at the end of the flag. The vertex structure now consists of two more `float` values for the second texture.

```
//constants
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ | D3DFVF_TEX2)

//structures
struct D3DVERTEX
{
float fX,
    fY,
    fZ;
float fU1, //first tex coords
    fV1;
float fU2, //second tex coords
    fV2;
};
```

The texture coordinates are defined in `main.cpp`. In this tutorial we use the same texture coordinates for both textures, because we want to see everything of both. I will not explain that again, because that was already handled in another tutorial.

## main.cpp

At the beginning we need two pointers to the texture objects.

```
//texture pointers
LPDIRECT3DTEXTURE9 pBaseMap = NULL, pDarkMap = NULL;
```

The next step is to load the base texture and the dark map from their files. This is simply done with the Direct3DX library functions.

```
//load base texture
D3DXCreateTextureFromFileA(g_App.GetDevice(), "texture.png", &pBaseMap);
//load dark map
D3DXCreateTextureFromFileA(g_App.GetDevice(), "darkmap.png", &pDarkMap);
```

The render part is nearly unchanged. The only new thing is that we apply two textures to the first and second texture stage. With help of the texture stage states Direct3D will perform all texture operations between these two textures.

```
g_App.GetDevice()->SetTexture(0,pBaseMap); //set base texture (NEW)
g_App.GetDevice()->SetTexture(1,pDarkMap); //set dark map (NEW)
g_App.GetDevice()->SetStreamSource(0,pQuadVB,sizeof(D3DVERTEX));
g_App.GetDevice()->DrawPrimitive(D3DPT_TRIANGLESTRIP,0,2);
```

## capplication.cpp

This file consists of the most important part. In `InitScene()` we first define the render states. We deactivate lighting, because all lighting information are now stored in the dark map.

```
m_pDirect3DDevice->SetRenderState(D3DRS_AMBIENT,RGB(255,255,255));
m_pDirect3DDevice->SetRenderState(D3DRS_LIGHTING,false);
m_pDirect3DDevice->SetRenderState(D3DRS_CULLMODE,D3DCULL_NONE);
m_pDirect3DDevice->SetRenderState(D3DRS_ZENABLE,D3DZB_TRUE);
```

The following texture stage states define that the first argument is the texture and that the texture operation is only selecting argument one. This means that the result of this texture stage is only the texture itself.

```
//arg 1 is texture (NEW)
m_pDirect3DDevice->SetTextureStageState(0,D3DTSS_COLORARG1,D3DTA_TEXTURE);
//select arg 1 (NEW)
m_pDirect3DDevice->SetTextureStageState(0,D3DTSS_COLOROP,D3DTOP_SELECTARG1);
```

Now comes the multitexturing part. The first argument of the second texture stage is again the texture itself. The second is `D3DTA_CURRENT`, which defines that argument two is the result of the last texture stage. Direct3D now takes the first and the second texture and performs the texture operation we define now. We set the operation to `D3DTOP_MODULATE`, so that the arguments are multiplied.

```
//arg 1 is texture (NEW)
m_pDirect3DDevice->SetTextureStageState(1,D3DTSS_COLORARG1,D3DTA_TEXTURE);
//arg 2 is last stage (NEW)
m_pDirect3DDevice->SetTextureStageState(1,D3DTSS_COLORARG2,D3DTA_CURRENT);
//multiply arguments (NEW)
m_pDirect3DDevice->SetTextureStageState(1,D3DTSS_COLOROP,D3DTOP_MODULATE);
```

At the end we check the device capabilities in the application function `CheckDeviceCaps()`. If the number of simultaneous textures is not more than one we have to quit.

```
if(!(m_DeviceCaps.MaxSimultaneousTextures > 1)) //number of textures
{
    MessageBox(m_hWindow,
        "single pass multitexturing not supported!",
```

```
        "CheckDeviceCaps()",  
        MB_OK);  
m_bRunningD3D = false;  
}
```

For devices that don't support single pass multitexturing there is a method called multi pass multitexturing. This will be explained in the next tutorial.