

Tutorial 6: Lighting, Materials

by Victor Saar

Content

In this tutorial we will handle one of the most important topics in 3D graphics programming. Lighting helps us creating realistic scenes or different atmospheres in a game. To use lighting we have to create a light source, of course. In the fixed-function pipeline DirectX supports simultaneous light sources, but they are rarely needed. The number of supported active lights is specified in the `D3DCAPS9` structure. The light source for this tutorial is a directional light, which means that the direction of the light is equal for the whole scene. To use lighting we need to know in which direction the triangles of our scene face. This information is stored in the normal, which is perpendicular to the triangle. The light intensity for each vertex is the dot product between the light vector and the normal vector. The material for the object is either given by the diffuse color value or a material structure, which we will use here.

d3ddefs.h

First we change the vertex format. We add `D3DFVF_NORMAL` to the definition of the vertex format, so that Direct3D knows about the normal vector. Then we add a vector between the coordinates and the texture coordinates in the vertex structure.

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1)

struct D3DVERTEX
{
    float    fX,
            fY,
            fZ;
    D3DVECTOR Normal; //normal vector (NEW)
    float    fU,
            fV;
};
```

application.cpp

First we change the render states in `InitScene()`. We deactivate the ambient light by setting the color to black, because the object is now lit by the light source. Then we enable lighting.

```
//no ambient light (NEW)
m_pDirect3DDevice->SetRenderState(D3DRS_AMBIENT,RGB(0,0,0));
//lighting enabled (NEW)
m_pDirect3DDevice->SetRenderState(D3DRS_LIGHTING,true);
m_pDirect3DDevice->SetRenderState(D3DRS_CULLMODE,D3DCULL_NONE);
m_pDirect3DDevice->SetRenderState(D3DRS_ZENABLE,D3DZB_TRUE);
```

At the end of `InitScene()` we create the light source. First we fill a `D3DLIGHT9` structure with the appropriate information. I think the elements are self-explanatory.

```
ZeroMemory(&Light,sizeof(Light));
Light.Type = D3DLIGHT_DIRECTIONAL;
Light.Diffuse.r = 1.0f;
Light.Diffuse.g = 1.0f;
Light.Diffuse.b = 1.0f;
Light.Direction.x = 1.0f;
Light.Direction.y = 0.0f;
```

```
Light.Direction.z = 1.0f;
Light.Range = 1000.0f;
```

Now we have to send the information about the light to the device. The last thing is to tell the device, that it should enable the light. The first parameter of both functions is the index, in which we set the light. This is used when you want to set more than one light. Now we are ready to lit the scene.

```
m_pDirect3DDevice->SetLight(0,&Light); //set the light (NEW)
m_pDirect3DDevice->LightEnable(0,true); //enables the light (NEW)
```

main.cpp

Here we haven't much to do again. At the beginning we add a `D3DMATERIAL9` object for the material.

```
D3DMATERIAL9 PyramideMaterial; //material object (NEW)
```

Now comes the most important thing for lighting. We have to add the normal vector to our vertices. The coordinates for the normal vector are the 4th, 5th and 6th value in each vertex. They describe the x, y and z coordinate of the vector.

```
D3DVERTEX aTriangle[] = {{ 0.0f, 1.0f, 0.0f, 0.0f,1.0f,-1.0f,0.5f,0.0f},
                          {-1.0f,-1.0f,-1.0f, 0.0f,1.0f,-1.0f,0.0f,1.0f},
                          { 1.0f,-1.0f,-1.0f, 0.0f,1.0f,-1.0f,1.0f,1.0f},
                          { 0.0f, 1.0f, 0.0f,-1.0f,1.0f, 0.0f,0.5f,0.0f},
                          {-1.0f,-1.0f, 1.0f,-1.0f,1.0f, 0.0f,0.0f,1.0f},
                          {-1.0f,-1.0f,-1.0f,-1.0f,1.0f, 0.0f,1.0f,1.0f},
                          { 0.0f, 1.0f, 0.0f, 0.0f,1.0f, 1.0f,0.5f,0.0f},
                          { 1.0f,-1.0f, 1.0f, 0.0f,1.0f, 1.0f,0.0f,1.0f},
                          {-1.0f,-1.0f, 1.0f, 0.0f,1.0f, 1.0f,1.0f,1.0f},
                          { 0.0f, 1.0f, 0.0f, 1.0f,1.0f, 0.0f,0.5f,0.0f},
                          { 1.0f,-1.0f,-1.0f, 1.0f,1.0f, 0.0f,0.0f,1.0f},
                          { 1.0f,-1.0f, 1.0f, 1.0f,1.0f, 0.0f,1.0f,1.0f}};

D3DVERTEX aQuad[] = {{-1.0f,-1.0f,-1.0f,0.0f,-1.0f,0.0f,0.0f,0.0f},
                     { 1.0f,-1.0f,-1.0f,0.0f,-1.0f,0.0f,1.0f,0.0f},
                     {-1.0f,-1.0f, 1.0f,0.0f,-1.0f,0.0f,0.0f,1.0f},
                     { 1.0f,-1.0f, 1.0f,0.0f,-1.0f,0.0f,1.0f,1.0f}};
```

That is all we have to do with the vertices. At the end we must define the material for the pyramide. It describes how the polygons react on the incoming light. First we zero the memory the material uses. Then we set the diffuse color value to white, because we want to have a full reflected light.

```
//zero memory (NEW)
ZeroMemory(&PyramideMaterial,sizeof(PyramideMaterial));
//diffuse color (NEW)
PyramideMaterial.Diffuse.r = 1.0f;
PyramideMaterial.Diffuse.g = 1.0f;
PyramideMaterial.Diffuse.b = 1.0f;
```

The last thing is to set the material, so that the device knows, which one we currently want to use. We add this function after `SetTexture()` in the main loop, because this information gets also lost when you press *ALT+TAB*.

```
g_App.GetDevice()->SetMaterial(&PyramideMaterial); //set light (NEW)
```

The rest of the file needn't be changed. The lighting will automatically be used now.