Tutorial 4: Creating and Using Lights

Direct3D lights add more realism to 3D objects. When used, each geometric object in the scene will be lit based on the location and type of lights that are used. The sample code in this tutorial introduces the topics of lights and materials.

This tutorial has the following steps to create a material and a light.

Steps

Note

The path of the Lights sample project is:

(*SDK root*)\Samples\C++\Direct3D\Tutorials\Tut04_Lights

The sample code in the Lights project is nearly identical to the sample code in the Matrices project. The Creating and Using Lights tutorial focuses only on the code unique to creating and using lights and does not cover setting up Direct3D, handling Windows messages, rendering, or shutting down. For information about these tasks, see Tutorial 1: Creating a Device.

This tutorial uses custom vertices and a vertex buffer to display geometry. For more information about selecting a custom vertex type and implementing a vertex buffer, see Tutorial 2: Rendering Vertices.

This tutorial makes use of matrices to transform geometry. For more information about matrices and transformations, see Tutorial 3: Using Matrices.

Step 1 - Initializing Scene Geometry

One of the requirements of using lights is that each surface has a normal. To do this, the Lights sample project uses a different custom vertex type. The new custom vertex format has a 3D position and a surface normal. The surface normal is used internally by Direct3D for lighting calculations.

```
struct CUSTOMVERTEX
{
    D3DXVECTOR3 position; // The 3D position for the vertex.
    D3DXVECTOR3 normal;   // The surface normal for the vertex.
};
// Custom flexible vertex format (FVF).
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ|D3DFVF_NORMAL)
```

Now that the correct vector format is defined, the Lights sample project calls InitGeometry, an application-defined function that creates a cylinder. The first step is to create a vertex buffer that stores the points of the cylinder as shown in the following sample code.

```
// Create the vertex buffer.
if( FAILED( g_pd3dDevice->CreateVertexBuffer( 50*2*sizeof(CUSTOMVERTEX),
                                              0 /*Usage*/, D3DFVF_CUSTOMVERTEX,
                                              D3DPOOL_DEFAULT, &g_pVB, NULL ) ) )
    return E_FAIL;
```

The next step is to fill the vertex buffer with the points of the cylinder. Note that in the following sample code, each point is defined by a position and a normal.

```
CUSTOMVERTEX* pVertices;
if( FAILED( g_pVB->Lock( 0, 0, (void**)&pVertices, 0 ) ) ) return E_FAIL;
```

```
for( DWORD i=0; i<50; i++ )
{
    FLOAT theta = (2*D3DX_PI*i)/(50-1);
    pVertices[2*i+0].position = D3DXVECTOR3( sinf(theta),-1.0f, cosf(theta) );
    pVertices[2*i+0].normal   = D3DXVECTOR3( sinf(theta), 0.0f, cosf(theta) );
    pVertices[2*i+1].position = D3DXVECTOR3( sinf(theta), 1.0f, cosf(theta) );
    pVertices[2*i+1].normal   = D3DXVECTOR3( sinf(theta), 0.0f, cosf(theta) );
}
```

After the preceding sample code fills the vertex buffer with the vertices for a cylinder, the vertex buffer is ready for rendering. But first, the material and light for this scene must be set up before rendering the cylinder. This is described in Step 2 - Setting Up Material and Light.

Step 2 - Setting Up Material and Light

To use lighting in Direct3D, you must create one or more lights. To determine which color a geometric object reflects, a material is created that is used to render geometric objects. Before rendering the scene, the Lights sample project calls SetupLights, an application-defined function that sets up one material and one directional light.

- Creating a Material
- Creating a Light

Creating a Material

A material defines the color that is reflected off the surface of a geometric object when a light hits it. The following code fragment uses the **D3DMATERIAL9** structure to create a material that is yellow.

```
D3DMATERIAL9 mtrl;
ZeroMemory( &mtrl, sizeof(mtrl) );
mtrl.Diffuse.r = mtrl.Ambient.r = 1.0f;
mtrl.Diffuse.g = mtrl.Ambient.g = 1.0f;
mtrl.Diffuse.b = mtrl.Ambient.b = 0.0f;
mtrl.Diffuse.a = mtrl.Ambient.a = 1.0f;
g_pd3dDevice->SetMaterial( &mtrl );
```

The diffuse color and ambient color for the material are set to yellow. The call to the **IDirect3DDevice9::SetMaterial** method applies the material to the Direct3D device used to render the scene. The only parameter that **IDirect3DDevice9::SetMaterial** accepts is the address of the material to set. After this call is made, every primitive will be rendered with this material until another call is made to **IDirect3DDevice9::SetMaterial** that specifies a different material.

Now that material has been applied to the scene, the next step is to create a light.

Creating a Light

There are three types of lights available in Direct3D:

- point lights
- directional lights
- spotlights

The sample code creates a directional light, which is a light that goes in one direction. The code also oscillates the direction of the light.

The following code fragment uses the **D3DLIGHT9** structure to create a directional light.

```
D3DXVECTOR3 vecDir;
```

```
D3DLight9 light;
ZeroMemory( &light, sizeof(light) );
light.Type = D3DLIGHT_DIRECTIONAL;
```

The following code fragment sets the diffuse color for this light to white.

```
light.Diffuse.r = 1.0f;
light.Diffuse.g = 1.0f;
light.Diffuse.b = 1.0f;
```

The following code fragment rotates the direction of the light around in a circle.

```
vecDir = D3DXVECTOR3(cosf(timeGetTime()/360.0f),
                     0.0f,
                     sinf(timeGetTime()/360.0f) );
D3DXVec3Normalize( (D3DXVECTOR3*)&light.Direction, &vecDir );
```

The call to **D3DXVec3Normalize** normalizes the direction vector used to determine the direction of the light.

A range can be specified to tell Direct3D how far the light will have an effect. This member does not affect directional lights. The following code fragment assigns a range of 1000 units to this light.

```
light.Range = 1000.0f;
```

The following code fragment assigns the light to the Direct3D device by calling **IDirect3DDevice9::SetLight**.

```
g_pd3dDevice->SetLight( 0, &light );
```

The first parameter that **IDirect3DDevice9::SetLight** accepts is the index that this light will be assigned to. Note that if a light already exists at that location, it will be overwritten by the new light. The second parameter is a pointer to the light structure that defines the light. The Lights sample project places this light at index 0.

The following code fragment enables the light by calling **IDirect3DDevice9::LightEnable**.

```
g_pd3dDevice->LightEnable( 0, TRUE);
```

The first parameter that **IDirect3DDevice9::LightEnable** accepts is the index of the light to enable. The second parameter is a Boolean value that tells whether to turn the light on (TRUE) or off (FALSE). In the sample code above, the light at index 0 is turned on.

The following code fragment tells Direct3D to render lights by calling **IDirect3DDevice9::SetRenderState**.

```
g_pd3dDevice->SetRenderState( D3DRS_LIGHTING, TRUE );
```

The first two parameters that **IDirect3DDevice9::SetRenderState** accepts is which device state variable to modify and what value to set it to. This code sample sets the D3DRS_LIGHTING device variable to TRUE, which has the effect of enabling the rendering of lights.

The final step in this code sample is to turn on ambient lighting by again calling **IDirect3DDevice9::SetRenderState**.

```
g_pd3dDevice->SetRenderState( D3DRS_AMBIENT, 0x00202020 );
```

The preceding code fragment sets the D3DRS_AMBIENT device variable to a light gray color (0x00202020). Ambient lighting will light up all objects by the given color.

For more information about lighting and materials, see **Lights and Materials (Direct3D 9)**.

This tutorial has shown you how to use lights and materials. Tutorial 5: Using Texture Maps shows you how to add texture to surfaces.