DirectX Programming #3

Kang, Seong-tae

Computer Graphics, 2008 Spring

Contents

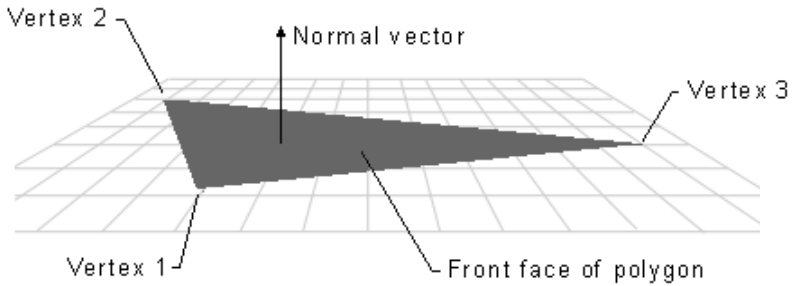
- Material and Lighting
- Sample Code Walkthrough

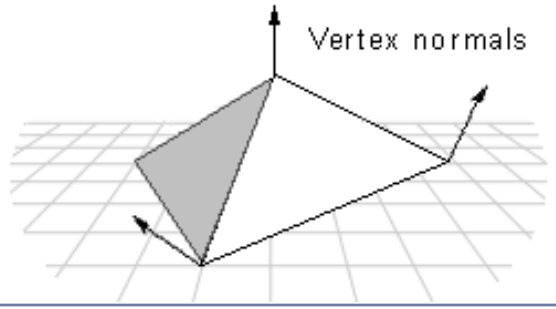
Material and Lighting

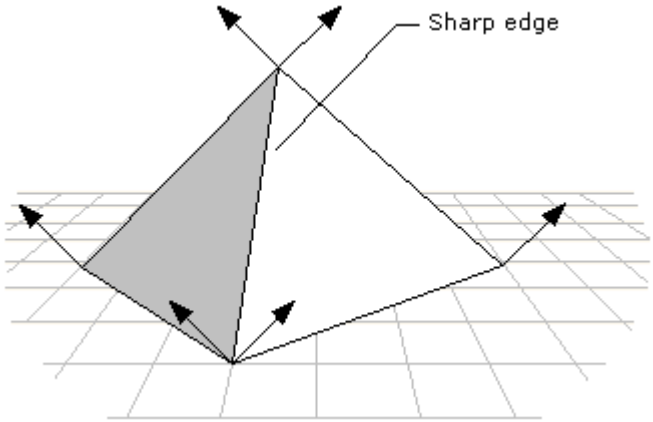
Computer Graphics, 2008 Spring

Normal

- For shading, you have to specify the vertex normal








```
struct CUSTOMVERTEX
{
    D3DXVECTOR3 position; // The 3D position for the vertex.
    D3DXVECTOR3 normal;   // The Quad normal for the vertex.
};

#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ|D3DFVF_NORMAL)
```

Material

- Defines characteristics of the Quad of a geometric object
  - Color
    - Diffuse
    - Ambient
    - Specular
  - Glossiness
    - Power (of exponent term)

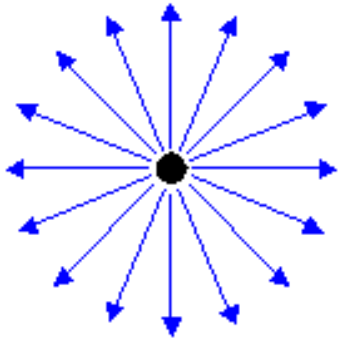
Per-vertex colors

- We have used per-vertex color already
  - D3DFVF_DIFFUSE : diffuse color
- Either per-vertex information or material based information can be used as shading factor
  - D3DMCS_MATERIAL : material color
  - D3DMCS_COLOR1 : per-vertex diffuse color (DVDFVF_DIFFUSE)
  - D3DMCS_COLOR2 : per-vertex specular color (DVDFVF_SPECULAR)

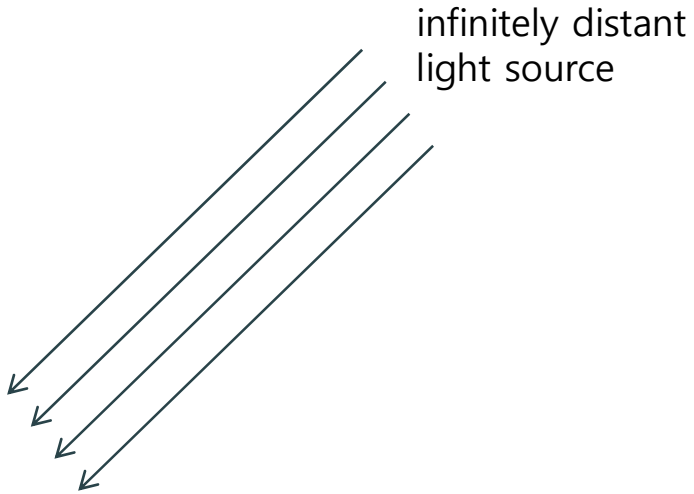
```
g_pd3dDevice->SetRenderState( D3DRS_DIFFUSEMATERIALSOURCE, D3DMCS_COLOR1 );
g_pd3dDevice->SetRenderState( D3DRS_SPECULARMATERIALSOURCE, D3DMCS_MATERIAL);
```

Light Type

- Point light
  - Position and color
  - Light bulb without lampshade

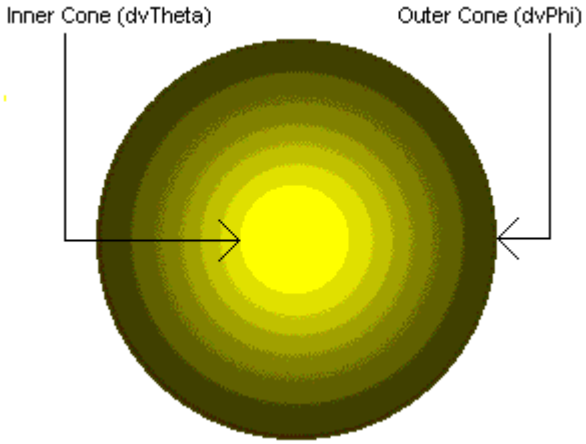


- Directional light
  - Color and direction
  - Sunray

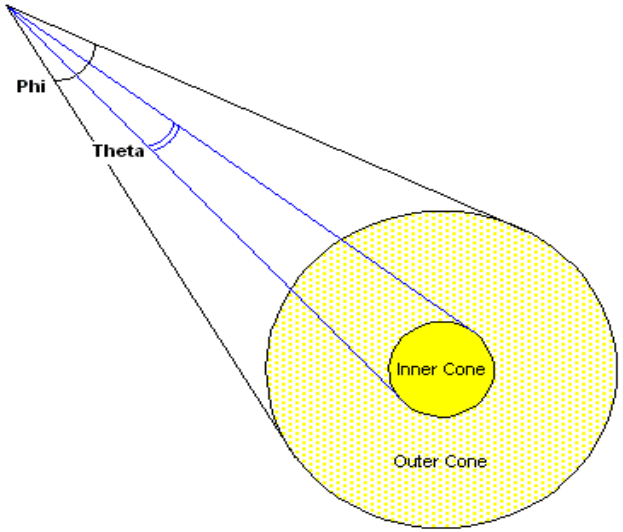


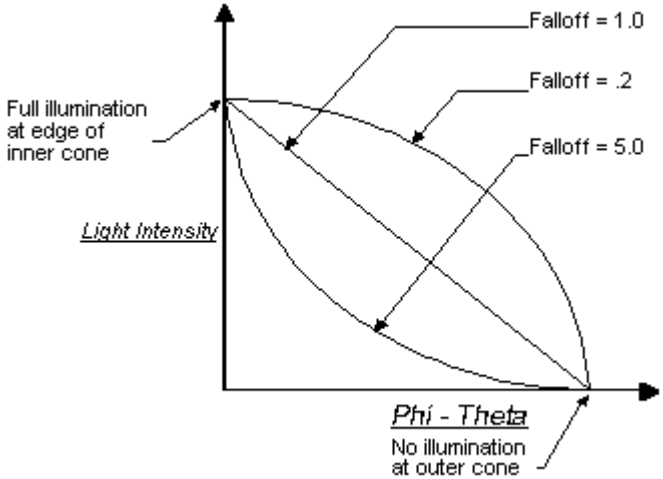
Light Type

- Spotlight
  - Color, position, direction
  - Cone shape illumination area
  - theta, phi, falloff, attenuation0/1/2



$$Atten = 1/(att0_i + att1_i * d + att2_i * d^2)$$





Using Lights

Light configuration

```
D3DXVECTOR3 vecDir;

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

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

vecDir=D3DXVECTOR3(1.0f, 2.0f, 0.0f);
D3DXVec3Normalize( (D3DXVECTOR3*)&light.Direction, &vecDir );
```

Attach light to the device

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

- There's no need to normalize the light direction vector.

Using Lights

- Enable lighting

```
g_pd3dDevice->SetRenderState( D3DRS_LIGHTING, TRUE );    // global configuration
g_pd3dDevice->LightEnable( 0, TRUE);                     // per-light configuration
```

- Ambient color setting
  - All object will be lighted up by this color, regardless of its material variables

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

Shading Model

- Three shading model
  - Flat shading : D3DSHADE_FLAT
  - Gouraud shading : D3DSHADE_GOURAUD
  - Phong shading : ~~D3DSHADE_PHONG~~
- DX does not support implicit Phong shading
  - Setting the shading model to D3D_SHADE_PHONG will lead unpredictable behavior
  - Can be implemented using the programmable shader

```
pDev->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_FLAT);
```

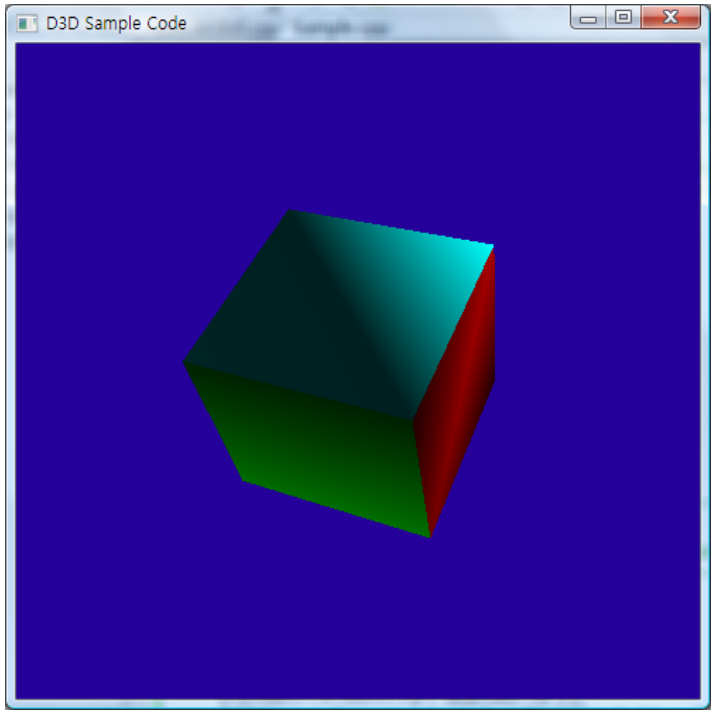
The default shading model is D3DSHADE_GOURAUD.

Sample Code Walkthrough

Kang, Seong-tae
Computer Graphics, 2008 Spring

Sample Code

- Creating a Window
- Creating Direct3D resources
  - Device
  - Vertex buffer
- Setting up parameters
  - Transformations
  - Material and Lighting
- Drawing primitives
- Transforming UI
  - ArcBall interface



Walkthrough #1

```
#include <Windows.h>
#include <d3dx9.h>
#pragma warning( disable : 4996 ) // disable deprecated warning
#include <strsafe.h>
#pragma warning( default : 4996 )
#include "D3DArcBall.h"

// Global variables
LPDIRECT3D9                     g_pD3D = NULL;
LPDIRECT3DDEVICE9               g_pd3dDevice = NULL;
LPDIRECT3DVERTEXBUFFER9         g_pVB = NULL;
CD3DArcBall*                    g_pArcBall = NULL;

struct SAMPLEVERTEX             // A structure for our custom vertex type
{
    D3DXVECTOR3 position;       // The 3D position for the vertex
    D3DXVECTOR3 normal;         // The Quad normal for the vertex
    D3DCOLOR diffuse;           // The diffuse color for the vertex
    D3DCOLOR specular;          // The specular color for the vertex
};

#define D3DFVF_SAMPLEVERTEX (D3DFVF_XYZ|D3DFVF_NORMAL|D3DFVF_DIFFUSE|D3DFVF_SPECULAR)
```

Walkthrough #2

```
//-----------------------------------------------------------------------
// Name: InitD3D()
// Desc: Initializes Direct3D
//-----------------------------------------------------------------------
HRESULT InitD3D( HWND hWnd )
{
    // Create the D3D object
    if( NULL == ( g_pD3D = Direct3DCreate9( D3D_SDK_VERSION ) ) )
        return E_FAIL;

    D3DPRESENT_PARAMETERS d3dpp;                    // Set up the structure used to create the
D3DDevice
    ZeroMemory( &d3dpp, sizeof(d3dpp) );
    d3dpp.Windowed = TRUE;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;
    d3dpp.EnableAutoDepthStencil = TRUE;
    d3dpp.AutoDepthStencilFormat = D3DFMT_D16;

    // Create the D3DDevice
    if( FAILED( g_pD3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
                                      D3DCREATE_SOFTWARE_VERTEXPROCESSING, &d3dpp, &g_pd3dDevice ) ) )
        return E_FAIL;

    g_pd3dDevice->SetRenderState( D3DRS_CULLMODE, D3DCULL_NONE );
    g_pd3dDevice->SetRenderState( D3DRS_ZENABLE, TRUE );          // Turn on the Z-buffer
    g_pArcBall=new CD3DArcBall();                                 // Initialize the ArcBall interface
    g_pArcBall->SetWindow(500, 500);

    return S_OK;
}
```

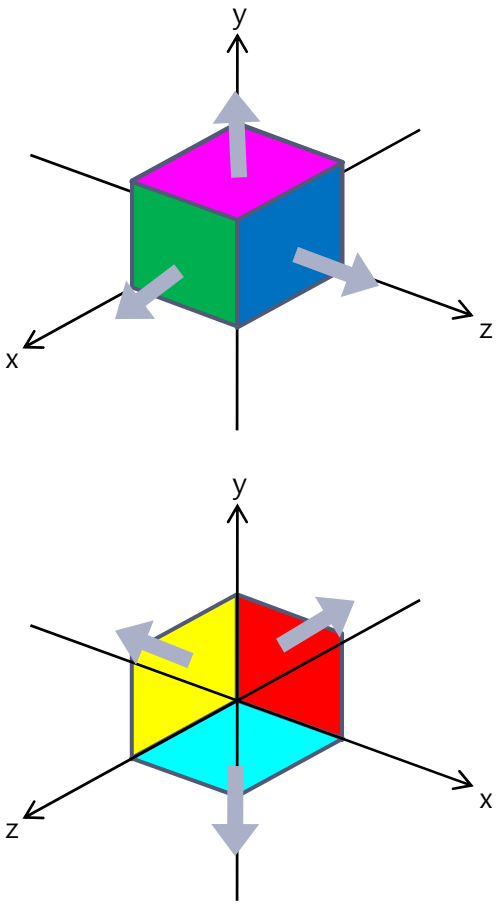
Walkthrough #3

```
//-----------------------------------------------------------------------------
// Name: InitGeometry()
// Desc: Creates the scene geometry.
// Add your custom vertex initialization code here.
//-----------------------------------------------------------------------------
HRESULT InitGeometry()
{
    // Create the vertex buffer.
    if( FAILED( g_pd3dDevice->CreateVertexBuffer( 24*sizeof(SAMPLEVERTEX),
                                  0, D3DFVF_SAMPLEVERTEX, D3DPOOL_DEFAULT, &g_pVB, NULL ) ) )
        return E_FAIL;

    SAMPLEVERTEX myVertices[24]=                    // store vertex data in CPU memory
    {
            { D3DXVECTOR3( 0.0f,10.0f, 0.0f), D3DXVECTOR3( 0.0f, 0.0f,-1.0f), 0xFFFF0000, 0xFFFFFFFF },
                                                    :
                                                    :
            { D3DXVECTOR3( 0.0f, 0.0f, 0.0f), D3DXVECTOR3( 0.0f,-1.0f, 0.0f), 0xFF00FFFF, 0xFFFFFFFF }
    };

    // Fill the vertex buffer
    SAMPLEVERTEX* pVertices;
    if( FAILED( g_pVB->Lock( 0, 0, (void**)&pVertices, 0 ) ) )  return E_FAIL;

    memcpy(pVertices, myVertices, sizeof(SAMPLEVERTEX)*24);              // Copy from CPU to GPU memory

    g_pVB->Unlock();

    return S_OK;
}
```

Vertex Declaration

- A cube : 6 quads those are a triangle strip respectively

```
{ D3DXVECTOR3( 0.0f,10.0f, 0.0f), D3DXVECTOR3( 0.0f, 0.0f,-1.0f), 0xFFFF0000, 0xFFFFFFFF },
{ D3DXVECTOR3(10.0f,10.0f, 0.0f), D3DXVECTOR3( 0.0f, 0.0f,-1.0f), 0xFFFF0000, 0xFFFFFFFF },  // Quad 1
{ D3DXVECTOR3( 0.0f, 0.0f, 0.0f), D3DXVECTOR3( 0.0f, 0.0f,-1.0f), 0xFFFF0000, 0xFFFFFFFF },
{ D3DXVECTOR3(10.0f, 0.0f, 0.0f), D3DXVECTOR3( 0.0f, 0.0f,1.0f), 0xFFFF0000, 0xFFFFFFFF },

{ D3DXVECTOR3(10.0f,10.0f, 0.0f), D3DXVECTOR3( 1.0f, 0.0f, 0.0f), 0xFF00FF00, 0xFFFFFFFF },
{ D3DXVECTOR3(10.0f,10.0f,10.0f), D3DXVECTOR3( 1.0f, 0.0f, 0.0f), 0xFF00FF00, 0xFFFFFFFF },  // Quad 2
{ D3DXVECTOR3(10.0f, 0.0f, 0.0f), D3DXVECTOR3( 1.0f, 0.0f, 0.0f), 0xFF00FF00, 0xFFFFFFFF },
{ D3DXVECTOR3(10.0f, 0.0f,10.0f), D3DXVECTOR3( 1.0f, 0.0f, 0.0f), 0xFF00FF00, 0xFFFFFFFF },

{ D3DXVECTOR3(10.0f,10.0f,10.0f), D3DXVECTOR3( 0.0f, 0.0f, 1.0f), 0xFF0000FF, 0xFFFFFFFF },
{ D3DXVECTOR3( 0.0f,10.0f,10.0f), D3DXVECTOR3( 0.0f, 0.0f, 1.0f), 0xFF0000FF, 0xFFFFFFFF },  // Quad 3
{ D3DXVECTOR3(10.0f, 0.0f,10.0f), D3DXVECTOR3( 0.0f, 0.0f, 1.0f), 0xFF0000FF, 0xFFFFFFFF },
{ D3DXVECTOR3( 0.0f, 0.0f,10.0f), D3DXVECTOR3( 0.0f, 0.0f, 1.0f), 0xFF0000FF, 0xFFFFFFFF },

{ D3DXVECTOR3( 0.0f,10.0f,10.0f), D3DXVECTOR3(-1.0f, 0.0f, 0.0f), 0xFFFFFF00, 0xFFFFFFFF },
{ D3DXVECTOR3( 0.0f,10.0f, 0.0f), D3DXVECTOR3(-1.0f, 0.0f, 0.0f), 0xFFFFFF00, 0xFFFFFFFF },  // Quad 4
{ D3DXVECTOR3( 0.0f, 0.0f,10.0f), D3DXVECTOR3(-1.0f, 0.0f, 0.0f), 0xFFFFFF00, 0xFFFFFFFF },
{ D3DXVECTOR3( 0.0f, 0.0f, 0.0f), D3DXVECTOR3(-1.0f, 0.0f, 0.0f), 0xFFFFFF00, 0xFFFFFFFF },

{ D3DXVECTOR3( 0.0f,10.0f,10.0f), D3DXVECTOR3( 0.0f, 1.0f, 0.0f), 0xFFFF00FF, 0xFFFFFFFF },
{ D3DXVECTOR3(10.0f,10.0f,10.0f), D3DXVECTOR3( 0.0f, 1.0f, 0.0f), 0xFFFF00FF, 0xFFFFFFFF },  // Quad 5
{ D3DXVECTOR3( 0.0f,10.0f, 0.0f), D3DXVECTOR3( 0.0f, 1.0f, 0.0f), 0xFFFF00FF, 0xFFFFFFFF },
{ D3DXVECTOR3(10.0f,10.0f, 0.0f), D3DXVECTOR3( 0.0f, 1.0f, 0.0f), 0xFFFF00FF, 0xFFFFFFFF },

{ D3DXVECTOR3(10.0f, 0.0f,10.0f), D3DXVECTOR3( 0.0f,-1.0f, 0.0f), 0xFF00FFFF, 0xFFFFFFFF },
{ D3DXVECTOR3( 0.0f, 0.0f,10.0f), D3DXVECTOR3( 0.0f,-1.0f, 0.0f), 0xFF00FFFF, 0xFFFFFFFF },  // Quad 6
{ D3DXVECTOR3(10.0f, 0.0f, 0.0f), D3DXVECTOR3( 0.0f,-1.0f, 0.0f), 0xFF00FFFF, 0xFFFFFFFF },
{ D3DXVECTOR3( 0.0f, 0.0f, 0.0f), D3DXVECTOR3( 0.0f,-1.0f, 0.0f), 0xFF00FFFF, 0xFFFFFFFF }
```



Walkthrough #4

```
//-----------------------------------------------------------------------------
// Name: Cleanup()
// Desc: Releases all previously initialized objects
//-----------------------------------------------------------------------------
VOID Cleanup()
{
 if( g_pArcBall != NULL)
       delete g_pArcBall;

    if( g_pVB != NULL )
        g_pVB->Release();

    if( g_pd3dDevice != NULL )
        g_pd3dDevice->Release();

    if( g_pD3D != NULL )
        g_pD3D->Release();
}
```

Walkthrough #5

```
//-----------------------------------------------------------------------------
// Name: SetupMatrices()
// Desc: Sets up the world, view, and projection transform matrices.
//-----------------------------------------------------------------------------
VOID SetupMatrices()
{
    // Set up world matrix
    D3DXMATRIXA16 matWorld;
    D3DXMatrixIdentity( &matWorld );
    D3DXMatrixTranslation( &matWorld, -5.0f, -5.0f, -5.0f );

    // apply arcball factors. THIS MUST BE THE LAST WORLD TRANSFORMATION
    D3DXMatrixMultiply( &matWorld, &matWorld, g_pArcBall->GetRotationMatrix());
    D3DXMatrixMultiply( &matWorld, &matWorld, g_pArcBall->GetTranslationMatrix());

    g_pd3dDevice->SetTransform( D3DTS_WORLD, &matWorld );

    // Set up view matrix
    D3DXVECTOR3 vEyePt( 25.0f, 15.0f,-25.0f );
    D3DXVECTOR3 vLookatPt( 0.0f, 0.0f, 0.0f );
    D3DXVECTOR3 vUpVec( 0.0f, 1.0f, 0.0f );
    D3DXMATRIXA16 matView;
    D3DXMatrixLookAtLH( &matView, &vEyePt, &vLookatPt, &vUpVec );
    g_pd3dDevice->SetTransform( D3DTS_VIEW, &matView );

    // Set up projection matrix
    D3DXMATRIXA16 matProj;
    D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI/4, 1.0f, 1.0f, 100.0f );
    g_pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
}
```

Walkthrough #6

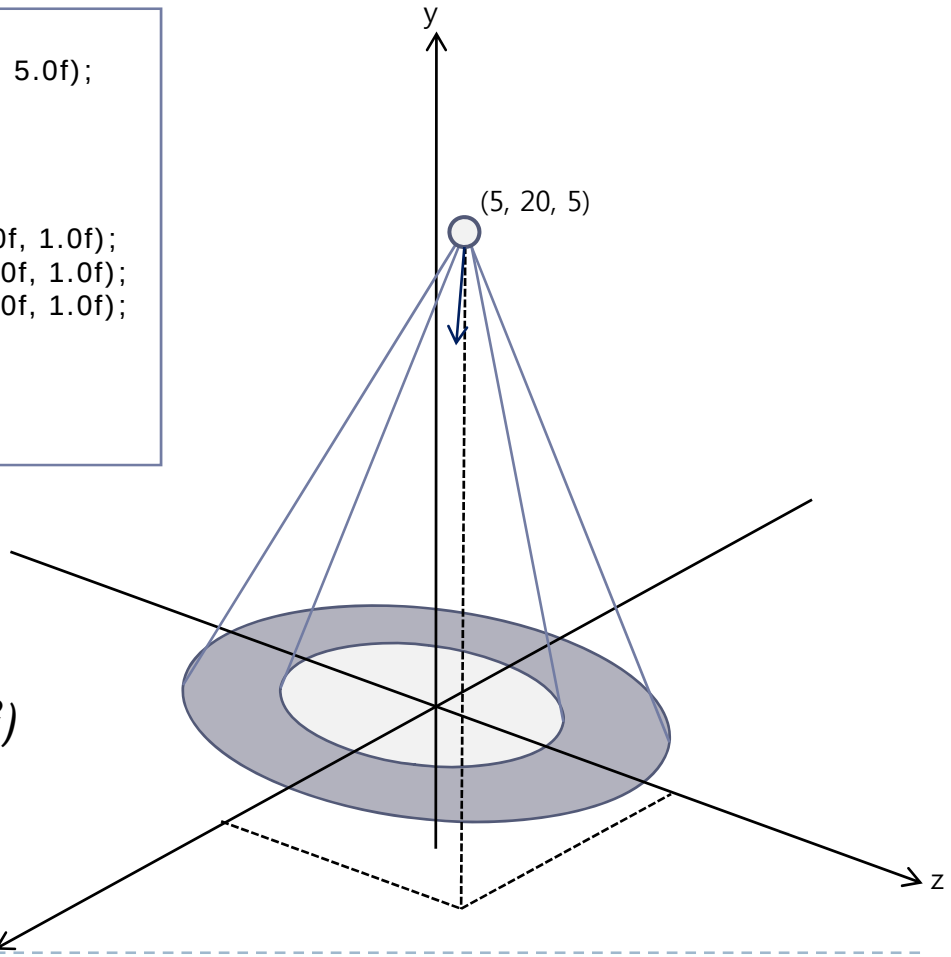
```
//-----------------------------------------------------------------------------
// Name: SetupLight()
// Desc: Sets up the light and materials for the scene.
//-----------------------------------------------------------------------------
VOID SetupLight()
{
    // Set up a material.
    D3DMATERIAL9 mtrl;
    ZeroMemory( &mtrl, sizeof(D3DMATERIAL9) );
    mtrl.Power=1.0f;
    g_pd3dDevice->SetMaterial( &mtrl );

    g_pd3dDevice->SetRenderState( D3DRS_AMBIENTMATERIALSOURCE, D3DMCS_COLOR1 );
    g_pd3dDevice->SetRenderState( D3DRS_DIFFUSEMATERIALSOURCE, D3DMCS_COLOR1 );
    g_pd3dDevice->SetRenderState( D3DRS_SPECULARMATERIALSOURCE, D3DMCS_COLOR2 );

    // Set up a spotlight
    D3DXVECTOR3 vecDir;
    D3DLIGHT9 light;
    ZeroMemory( &light, sizeof(D3DLIGHT9) );
    light.Type                          = D3DLIGHT_SPOT;
                        :
                        :
    light.Range       = 100.0f;
    g_pd3dDevice->SetLight( 0, &light );
    g_pd3dDevice->LightEnable( 0, TRUE );
    g_pd3dDevice->SetRenderState( D3DRS_LIGHTING, TRUE );

    // Finally, turn on some ambient light.
    g_pd3dDevice->SetRenderState( D3DRS_AMBIENT, 0x00202020 );
}
```

Light Declaration

- A spotlight heading the origin of world coordinate

```
light.Type                  = D3DLIGHT_SPOT;
light.Position              = D3DXVECTOR3(5.0f, 20.0f, 5.0f);
light.Direction = D3DXVECTOR3(-1.0f,-4.0f,-1.0f);
light.Theta                 = D3DX_PI/6;
light.Phi                   = D3DX_PI/4;
light.Falloff               = 1.5f;
light.Diffuse               = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
light.Specular               = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
light.Ambient                = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
light.Attenuation0           = 1.00f;
light.Attenuation1           = 0.05f;
light.Attenuation2           = 0.001f;
light.Range                  = 100.0f;
```

$$Att=1/(1+0.05d+0.001d^2)$$



Walkthrough #7

```
VOID Render()
{
    // Clear the backbuffer and the zbuffer
    g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,
                         D3DCOLOR_XRGB(38,0,155), 1.0f, 0 );

    // Begin the scene
    if( SUCCEEDED( g_pd3dDevice->BeginScene() ) )
    {
        // Setup the light and materials
        SetupLight();

        // Setup the world, view, and projection matrices
        SetupMatrices();

        // Render the vertex buffer contents
        g_pd3dDevice->SetStreamSource( 0, g_pVB, 0, sizeof(SAMPLEVERTEX) );
        g_pd3dDevice->SetFVF( D3DFVF_SAMPLEVERTEX );
                            for(int i=0; i<6; i++)
        g_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLESTRIP, i*4, 2 );

        // End the scene
        g_pd3dDevice->EndScene();
    }

    // Present the backbuffer contents to the display
    g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
}
```

Walkthrough #8

```
//-----------------------------------------------------------------------------
// Name: MsgProc()
// Desc: The window's message handler
//-----------------------------------------------------------------------------
LRESULT WINAPI MsgProc( HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
    switch( msg )
    {
        case WM_DESTROY:
            Cleanup();
            PostQuitMessage( 0 );
            return 0;
    }

    // ArcBall message handler processes mouse events and apply user input.
    // ALL MOUSE EVENT HANLDING FLOW MUST REACH HERE
    // WHEN YOU TRY TO APPEND YOUR OWN MOUSE HANDLING ROUTINE, BE CAREFUL.
    if(g_pArcBall)
        g_pArcBall->HandleMessages(hWnd, msg, wParam, lParam);

    return DefWindowProc( hWnd, msg, wParam, lParam );
}
```

Walkthrough #9

```
//-----------------------------------------------------------------------------
// Name: WinMain()
// Desc: The application's entry point
//-----------------------------------------------------------------------------
INT WINAPI WinMain( HINSTANCE hInst, HINSTANCE, LPSTR, INT )
{
    WNDCLASSEX wc = { sizeof(WNDCLASSEX), CS_CLASSDC, MsgProc, 0L, 0L,              // Register the window class
                      GetModuleHandle(NULL), NULL, NULL, NULL, NULL, "D3D Sample", NULL };
    RegisterClassEx( &wc );

    HWND hWnd = CreateWindow( "D3D Sample", "D3D Sample Code",                      // Create the application's window
                        WS_OVERLAPPEDWINDOW, 100, 100, 500, 500, NULL, NULL, wc.hInstance, NULL );

    if( SUCCEEDED( InitD3D( hWnd ) ) )                      // Initialize Direct3D
    {
        if( SUCCEEDED( InitGeometry() ) )                   // Create the geometry
        {
            ShowWindow( hWnd, SW_SHOWDEFAULT );             // Show the window
            UpdateWindow( hWnd );

            MSG msg;                                        // Enter the message loop
            ZeroMemory( &msg, sizeof(msg) );
            while( msg.message!=WM_QUIT )
            {
                if( PeekMessage( &msg, NULL, 0U, 0U, PM_REMOVE ) )
                {
                    TranslateMessage( &msg );
                    DispatchMessage( &msg );
                }
                else
                    Render();
            }
        }
    }

    UnregisterClass( "D3D Sample", wc.hInstance );
    return 0;
}
```