MY BRIGHT FUTURE

DSU Dongseo University 동서대학교

Beginning Direct3D Game Programming:

10. Shader Detail

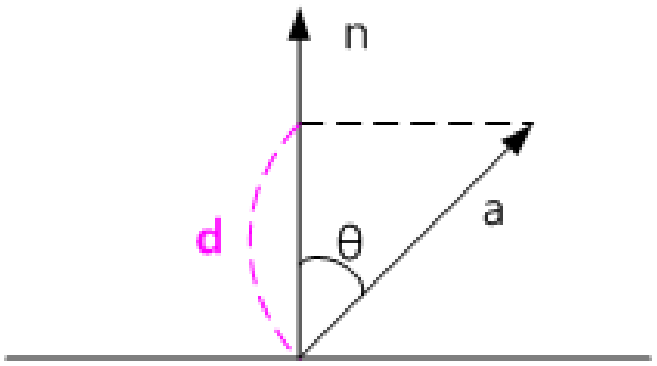
jintaeks@gmail.com

Division of Digital Contents, DongSeo University.

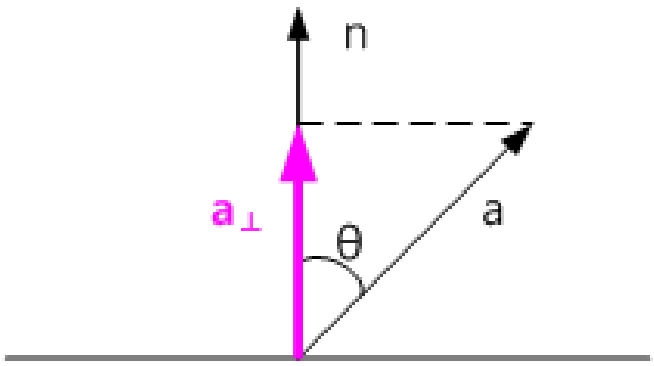
May 2016

DSU Dongseo University 동서대학교

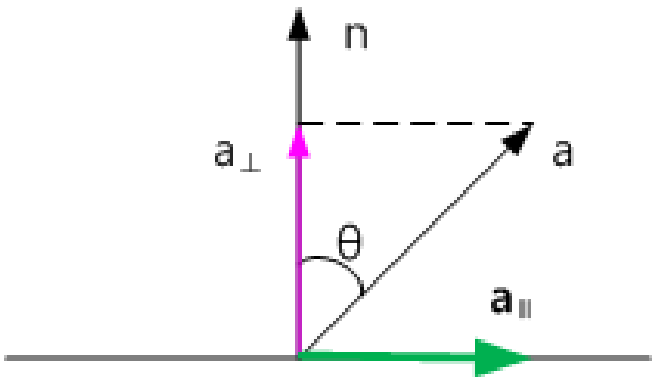
Decompose a vector



$\cos(\theta)=d/|a|$
$a\cdot n=|a||n|\cos(\theta)=|a|\cos(\theta)$
$\mathbf{d=a\cdot n}$



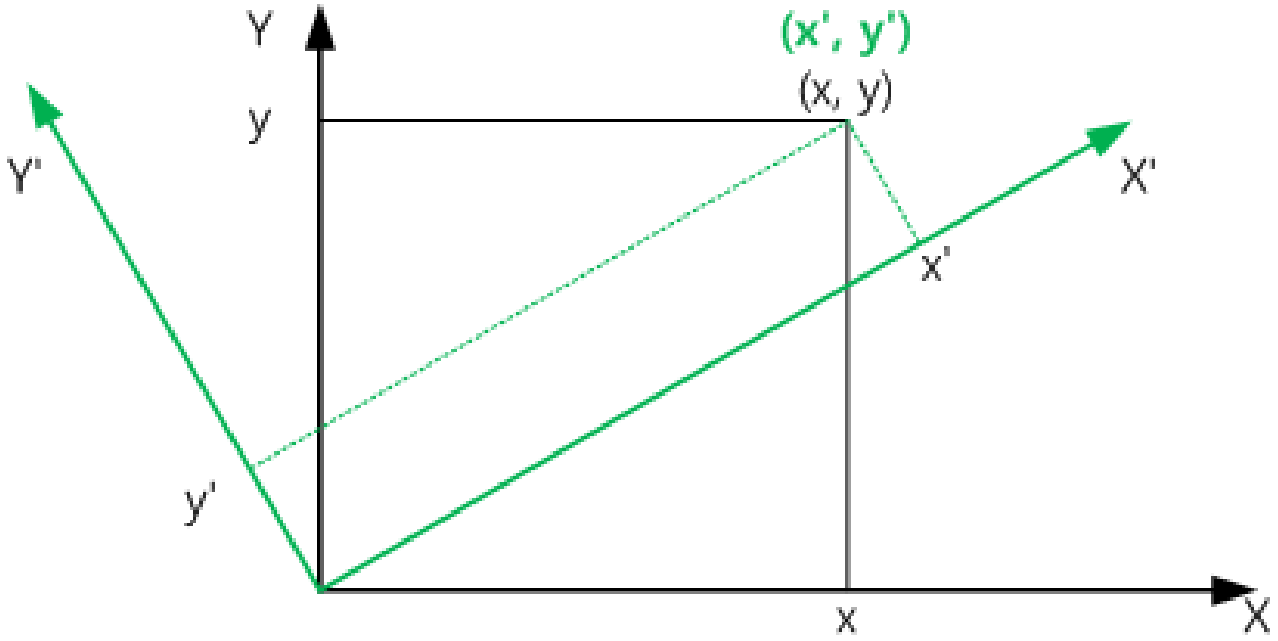
$\mathbf{a_{\perp}=(a\cdot n)n}$



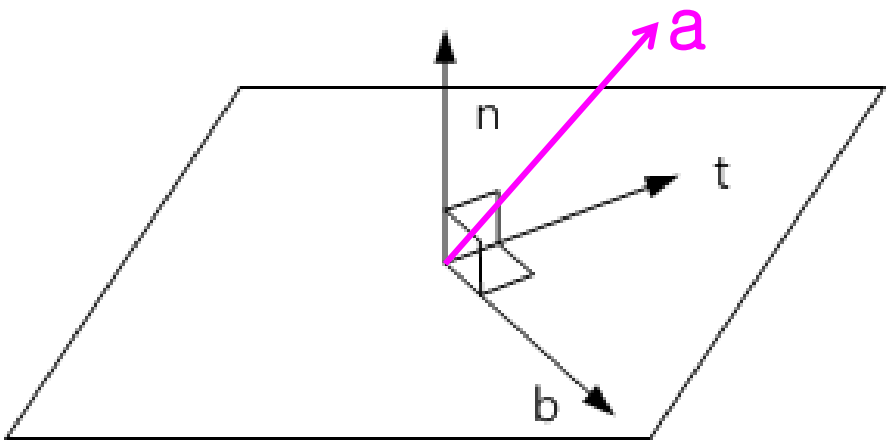
$\mathbf{a_{\parallel}=a-a_{\perp}=a-(a\cdot n)n}$

- ✓ $\mathbf{d=(a\cdot n)}$
- ✓ $\mathbf{a_{\perp}=(a\cdot n)n}$
- ✓ $\mathbf{a_{\parallel}=a-a_{\perp}=a-(a\cdot n)n}$

Coordinate Transform

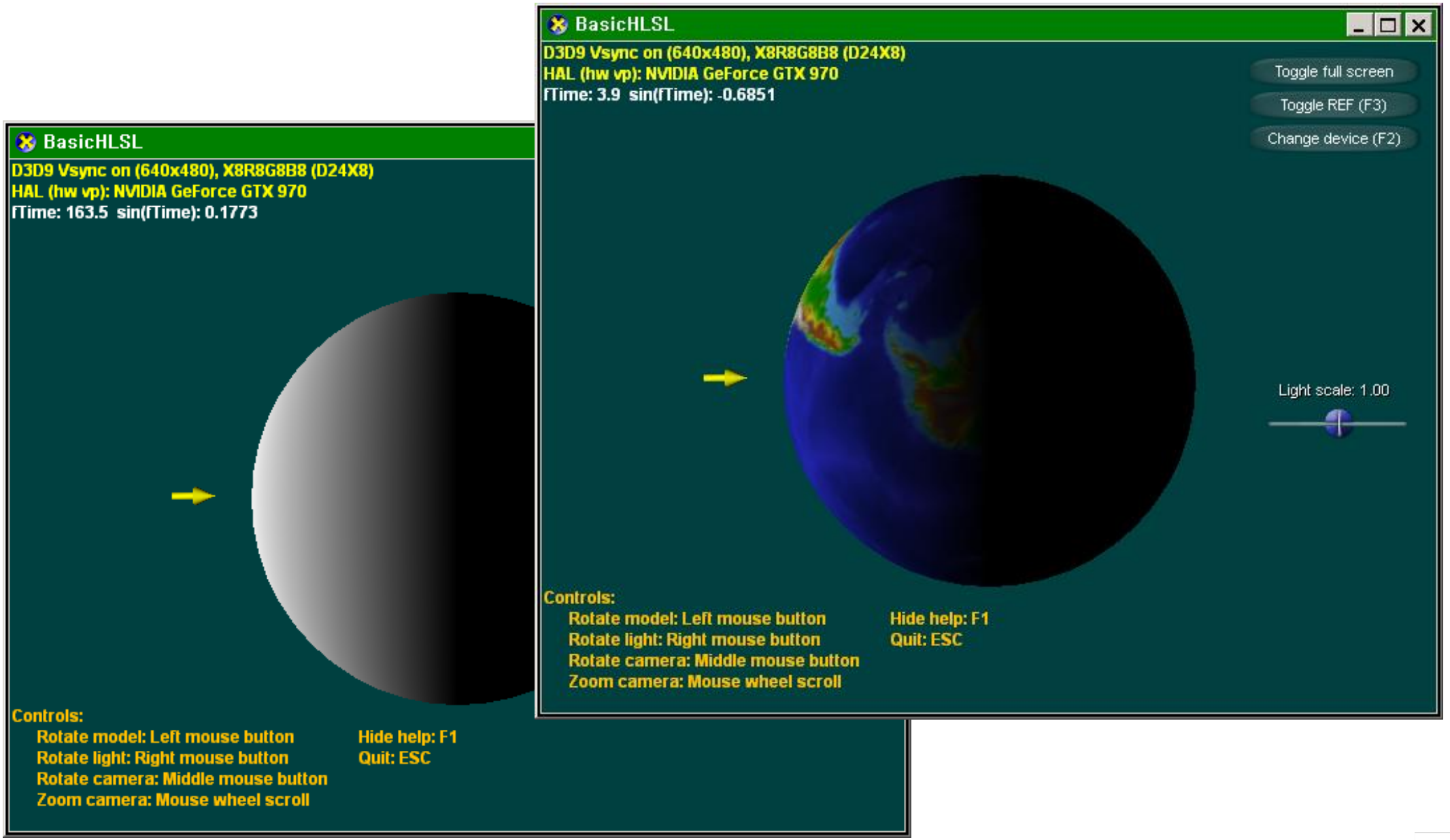


(a·n, a·t, a·b)



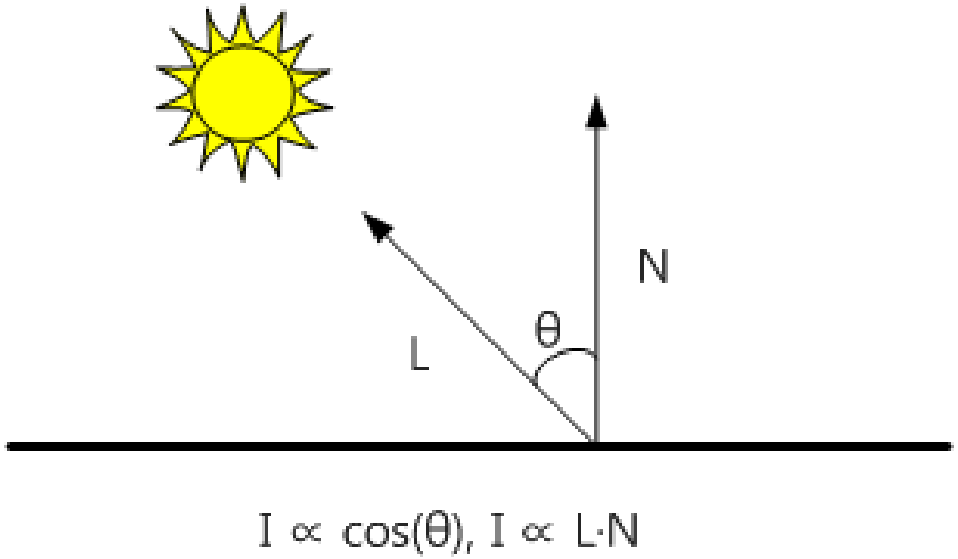
n×t = b

Step1 : Diffuse Light



Diffuse Lighting

- ✓ The diffuse lighting model following **Lambert's law** is described with the help of two vectors—the light vector L and the normal vector N.
- ✓ The diffuse reflection has its peak ($\cos(\theta) \equiv 1$) when L and N are aligned.

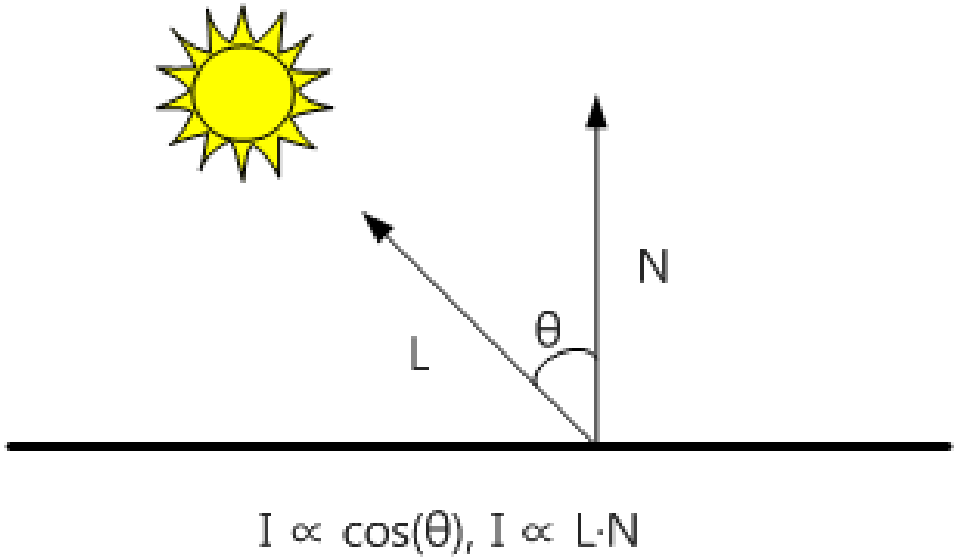


$I \propto \cos(\theta), I \propto L \cdot N$

Diffuse Lighting

- ✓ This diffuse lighting component is usually added to the ambient lighting component like this:

$$I = A_{intensity} * A_{color} + D_{intensity} * D_{color} * N \cdot L + Specular$$



Effect File: Declare Global Variables

```
//--------------------------------------------------------------------------------
// Global variables
//--------------------------------------------------------------------------------
float4 g_MaterialAmbientColor;      // Material's ambient color
float4 g_MaterialDiffuseColor;      // Material's diffuse color

float3 g_LightDir;                  // Light's direction in world space
float4 g_LightDiffuse;              // Light's diffuse color
float4 g_LightAmbient = float4(0.5,0.5,0.5,0.5);            // Light's ambient color

texture g_MeshTexture;              // Color texture for mesh

float    g_fTime;                   // App's time in seconds
float4x4 g_mWorld;                  // World matrix for object
float4x4 g_mWorldViewProjection;    // World * View * Projection matrix
```

Vertex Shader Output Structure

```
//--------------------------------------------------------------------------------
// Vertex shader output structure
//--------------------------------------------------------------------------------
struct VS_OUTPUT
{
    float4 Position   : POSITION;   // vertex position
    float4 Diffuse    : COLOR0;     // vertex diffuse color (note that COLOR0 is clamped from 0..1)
    float2 TextureUV  : TEXCOORD0;  // vertex texture coords
};
```

Vertex Shader

```
VS_OUTPUT RenderSceneVS( float4 vPos : POSITION, float3 vNormal : NORMAL, float2
vTexCoord0 : TEXCOORD0, uniform int nUnused, uniform bool bTexture, uniform bool
bAnimate )
{
    VS_OUTPUT Output;
    float3 vNormalWorldSpace;
    float4 vAnimatedPos = vPos;

    // Transform the position from object space to homogeneous projection space
    Output.Position = mul(vAnimatedPos, g_mWorldViewProjection);

    // Transform the normal from object space to world space
    vNormalWorldSpace = normalize(mul(vNormal, (float3x3)g_mWorld));
```

Vertex Shader

```
VS_OUTPUT RenderSceneVS( float4 vPos : POSITION, float3 vNormal : NORMAL, float2
vTexCoord0 : TEXCOORD0, uniform int nUnused, uniform bool bTexture, uniform bool
bAnimate )
{
    VS_OUTPUT Output;
    …
    // Compute simple directional lighting equation
    float3 vTotalLightDiffuse = float3(0,0,0);
    vTotalLightDiffuse += g_LightDiffuse * max( 0, dot( vNormalWorldSpace,
g_LightDir ) );
    Output.Diffuse.rgb = g_MaterialDiffuseColor * vTotalLightDiffuse +
                         g_MaterialAmbientColor * g_LightAmbient;
    Output.Diffuse.a = 1.0f;

    // Just copy the texture coordinate through
    Output.TextureUV = vTexCoord0;

    return Output;
}
```

Pixel Shader

```
struct PS_OUTPUT
{
    float4 RGBColor : COLOR0;  // Pixel color
};

PS_OUTPUT RenderScenePS( VS_OUTPUT In, uniform bool bTexture )
{
    PS_OUTPUT Output;

    // Lookup mesh texture and modulate it with diffuse
    if( bTexture )
        Output.RGBColor = tex2D(MeshTextureSampler, In.TextureUV) * In.Diffuse;
    else
        Output.RGBColor = In.Diffuse;

    return Output;
}
```

Technique

```
technique RenderSceneWithTexture1Light
{
    pass P0
    {
        VertexShader = compile vs_2_0 RenderSceneVS( 1, false, false );
        PixelShader = compile ps_2_0 RenderScenePS( false );
    }
}
```

Client : Declare Global Variables

```
//--------------------------------------------------------------------------------
// Global variables
//--------------------------------------------------------------------------------
ID3DXFont*                  g_pFont = NULL;         // Font for drawing text
ID3DXSprite*                g_pSprite = NULL;       // Sprite for batching draw text calls
bool                        g_bShowHelp = true;     // If true, it renders the UI control text
CModelViewerCamera          g_Camera;               // A model viewing camera
ID3DXEffect*                g_pEffect = NULL;       // D3DX effect interface
ID3DXMesh*                  g_pMesh = NULL;         // Mesh object
IDirect3DTexture9*          g_pMeshTexture = NULL;  // Mesh texture
CDXUTDialogResourceManager  g_DialogResourceManager; // manager for shared
resources of dialogs
```

OnCreateDevice()

```
// Read the D3DX effect file
    WCHAR str[MAX_PATH];
    V_RETURN( DXUTFindDXSDKMediaFileCch( str, MAX_PATH, L"BasicHLSL.fx" ) );

    // If this fails, there should be debug output as to
    // why the .fx file failed to compile
    V_RETURN( D3DXCreateEffectFromFile( pd3dDevice, str, NULL, NULL, dwShaderFlags, NULL,
&g_pEffect, NULL ) );

    // Create the mesh texture from a file
    V_RETURN( DXUTFindDXSDKMediaFileCch( str, MAX_PATH, L"earth₩₩earth.bmp" ) );

    V_RETURN( D3DXCreateTextureFromFileEx( pd3dDevice, str, D3DX_DEFAULT, D3DX_DEFAULT,
                                           D3DX_DEFAULT, 0, D3DFMT_UNKNOWN, D3DPOOL_MANAGED,
                                           D3DX_DEFAULT, D3DX_DEFAULT, 0,
                                           NULL, NULL, &g_pMeshTexture ) );
```

OnCreateDevice()

```
// Set effect variables as needed
D3DXCOLOR colorMtrlDiffuse( 1.0f, 1.0f, 1.0f, 1.0f );
D3DXCOLOR colorMtrlAmbient( 0.35f, 0.35f, 0.35f, 0 );

V_RETURN( g_pEffect->SetValue( "g_MaterialAmbientColor", &colorMtrlAmbient, sizeof( D3DXCOLOR ) ) );
V_RETURN( g_pEffect->SetValue( "g_MaterialDiffuseColor", &colorMtrlDiffuse, sizeof( D3DXCOLOR ) ) );
V_RETURN( g_pEffect->SetTexture( "g_MeshTexture", g_pMeshTexture ) );
```

```
//--------------------------------------------------------------------------------
// Global variables
//--------------------------------------------------------------------------------
float4 g_MaterialAmbientColor;      // Material's ambient color
float4 g_MaterialDiffuseColor;      // Material's diffuse color

float3 g_LightDir;                  // Light's direction in world s
float4 g_LightDiffuse;              // Light's diffuse color
float4 g_LightAmbient = float4(0.5,0.5,0.5,0.5);              //
```

OnResetDevice()

```
HRESULT hr;

V_RETURN( g_DialogResourceManager.OnD3D9ResetDevice() );
V_RETURN( g_SettingsDlg.OnD3D9ResetDevice() );

if( g_pFont )
    V_RETURN( g_pFont->OnResetDevice() );
if( g_pEffect )
    V_RETURN( g_pEffect->OnResetDevice() );

// Create a sprite to help batch calls when drawing many lines of text
V_RETURN( D3DXCreateSprite( pd3dDevice, &g_pSprite ) );
```

OnFrameRender()

```
// Render the scene
    if( SUCCEEDED( pd3dDevice->BeginScene() ) )
    {
        // Get the projection & view matrix from the camera class
        mWorld = g_mCenterWorld * *g_Camera.GetWorldMatrix();
        mProj = *g_Camera.GetProjMatrix();
        mView = *g_Camera.GetViewMatrix();

        mWorldViewProjection = mWorld * mView * mProj;

        D3DXCOLOR arrowColor = D3DXCOLOR( 1, 1, 0, 1 );
        V( g_LightControl.OnRender9( arrowColor, &mView, &mProj, g_Camera.GetEyePt() )
);
        vLightDir = g_LightControl.GetLightDirection();
        vLightDiffuse = g_fLightScale * D3DXCOLOR( 1, 1, 1, 1 );
```

```
        V( g_pEffect->SetValue( "g_LightDir", &vLightDir, sizeof( D3DXVECTOR3 ) ) );
        V( g_pEffect->SetValue( "g_LightDiffuse", &vLightDiffuse, sizeof( D3DXVECTOR4 ) )
);

        // Update the effect's variables.  Instead of using strings, it would
        // be more efficient to cache a handle to the parameter by calling
        // ID3DXEffect::GetParameterByName
        V( g_pEffect->SetMatrix( "g_mWorldViewProjection", &mWorldViewProjection ) );
        V( g_pEffect->SetMatrix( "g_mWorld", &mWorld ) );
        V( g_pEffect->SetFloat( "g_fTime", ( float )fTime ) );

        D3DXCOLOR vWhite = D3DXCOLOR( 1, 1, 1, 1 );
        V( g_pEffect->SetValue( "g_MaterialDiffuseColor", &vWhite, sizeof( D3DXCOLOR )
) );
        V( g_pEffect->SetFloat( "g_fTime", ( float )fTime ) );
```

```
V( g_pEffect->SetTechnique( "RenderSceneWithTexture1Light" ) );

// Apply the technique contained in the effect
V( g_pEffect->Begin( &cPasses, 0 ) );

for( iPass = 0; iPass < cPasses; iPass++ )
{
    V( g_pEffect->BeginPass( iPass ) );

    V( g_pEffect->CommitChanges() );

    // Render the mesh with the applied technique
    V( g_pMesh->DrawSubset( 0 ) );

    V( g_pEffect->EndPass() );
}
V( g_pEffect->End() );
```

OnLostDevice()

```
void CALLBACK OnLostDevice( void* pUserContext )
{
    g_DialogResourceManager.OnD3D9LostDevice();
    g_SettingsDlg.OnD3D9LostDevice();
    CDXUTDirectionWidget::StaticOnD3D9LostDevice();
    if( g_pFont )
        g_pFont->OnLostDevice();
    if( g_pEffect )
        g_pEffect->OnLostDevice();
    SAFE_RELEASE( g_pSprite );

}
```

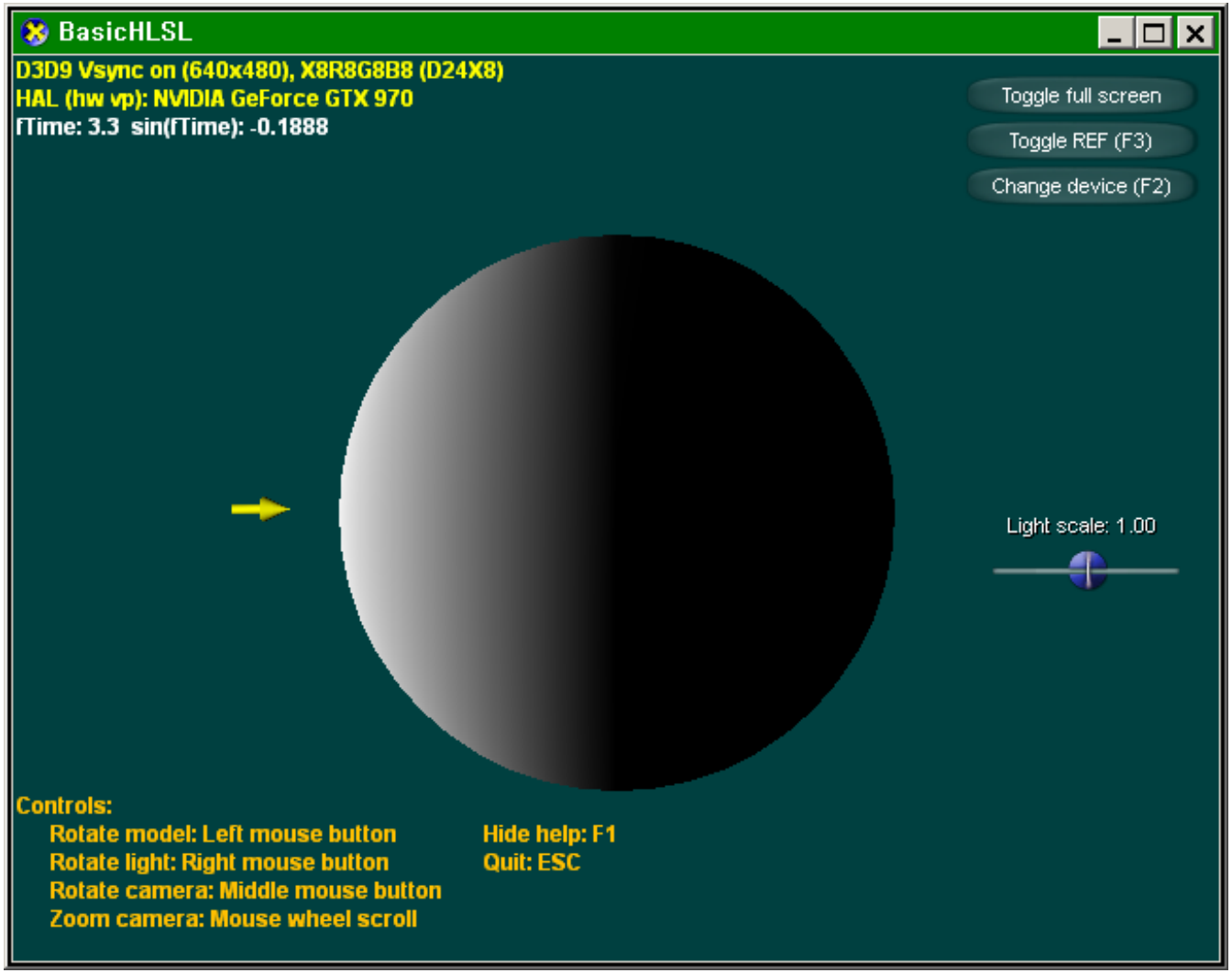
OnDestroyDevice()

```
void CALLBACK OnDestroyDevice( void* pUserContext )
{
    g_DialogResourceManager.OnD3D9DestroyDevice();
    g_SettingsDlg.OnD3D9DestroyDevice();
    CDXUTDirectionWidget::StaticOnD3D9DestroyDevice();
    SAFE_RELEASE( g_pEffect );
    SAFE_RELEASE( g_pFont );
    SAFE_RELEASE( g_pMesh );
    SAFE_RELEASE( g_pMeshTexture );
}
```

Practice: Build and Run BasicHLSL Step1

Step2 : Diffuse Light in Local Space

- ✓ Use Object space uniform variables in Shader to speed up.



Client: OnFrameRender() Differences

```
    //D3DXVECTOR3 vLightDir[ MAX_LIGHTS ];
    D3DXVECTOR3 vLightDir;
    //{{ step2
    D3DXVECTOR3 vLightDirObject;
    //}} step2
    //D3DXCOLOR vLightDiffuse[ MAX_LIGHTS ];
    D3DXCOLOR vLightDiffuse;
    UINT iPass, cPasses;
    D3DXMATRIXA16 mWorld;
    //{{ step2
    D3DXMATRIXA16 mWorldInv;
    //}] step2
    D3DXMATRIXA16 mView;
```

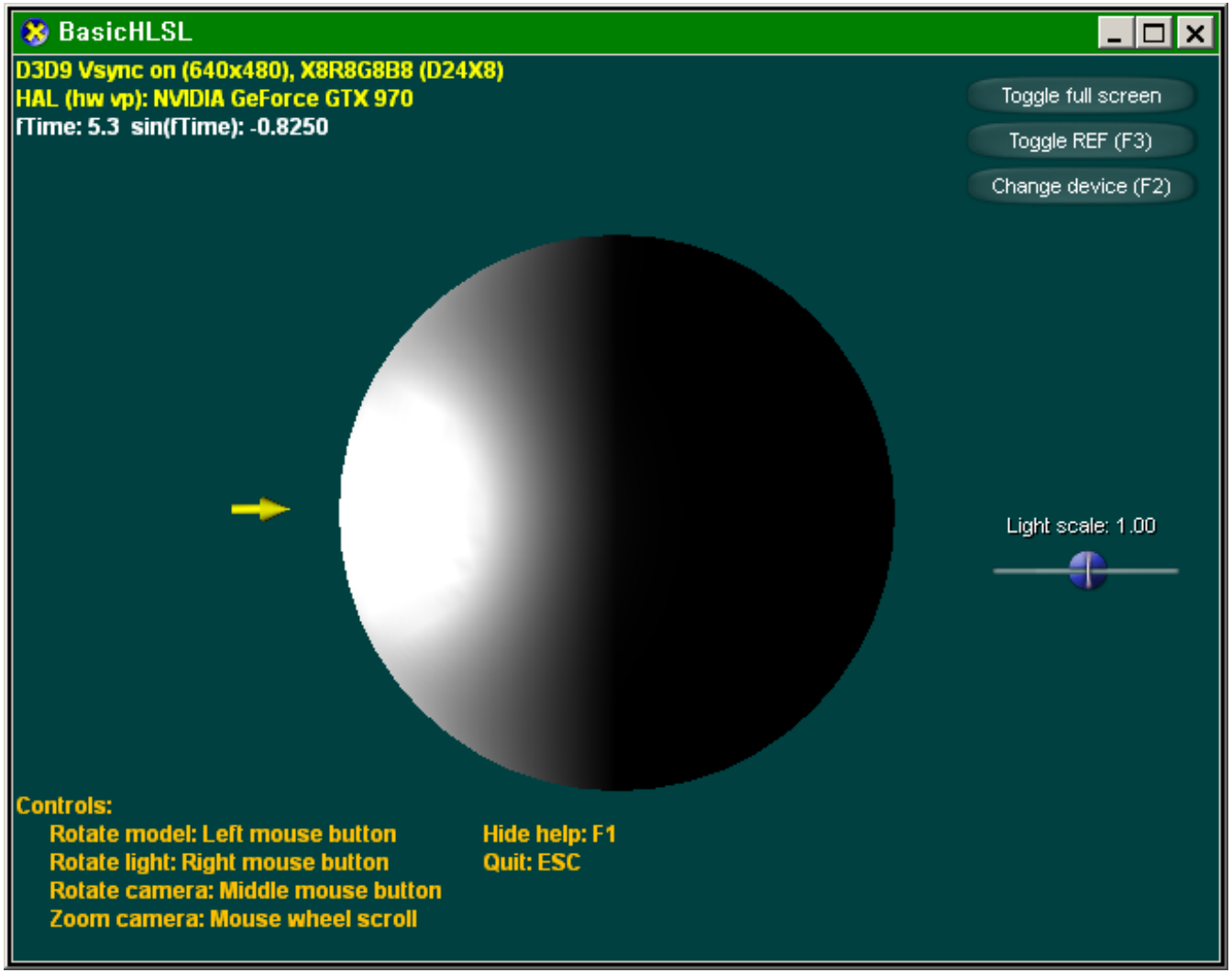
```
// Get the projection & view matrix from the camera class
mWorld = g_mCenterWorld * *g_Camera.GetWorldMatrix();
//{{ step2
D3DXMatrixInverse( OUT &mWorldInv, NULL, &mWorld );
//}} step2
mProj = *g_Camera.GetProjMatrix();
mView = *g_Camera.GetViewMatrix();

mWorldViewProjection = mWorld * mView * mProj;

D3DXCOLOR arrowColor = D3DXCOLOR( 1, 1, 0, 1 );
V( g_LightControl.OnRender9( arrowColor, &mView, &mProj, g_Camera.GetEyePt(
vLightDir = g_LightControl.GetLightDirection();
//{{ step2
D3DXVec3TransformCoord( OUT &vLightDirObject, &vLightDir, &mWorldInv );
//}} step2
vLightDiffuse = g_fLightScale * D3DXCOLOR( 1, 1, 1, 1 );

V( g_pEffect->SetValue( "g_LightDir", &vLightDirObject, sizeof( D3DXVECTOR3
V( g_pEffect->SetValue( "g_LightDiffuse", &vLightDiffuse, sizeof( D3DXVECTO
```

Vertex Shader Differences

```
// Transform the normal from object space to world space
//{{ step2 commented out
//vNormalWorldSpace = normalize(mul(vNormal, (float3x3)g_mWorld)); // norma
//}} step2

// Compute simple directional lighting equation
float3 vTotalLightDiffuse = float3(0,0,0);
//{{ step2
//vTotalLightDiffuse += g_LightDiffuse * max( 0, dot( vNormalWorldSpace, g_
vTotalLightDiffuse += g_LightDiffuse * max( 0, dot( vNormal, g_LightDir ) )
//}} step2
```

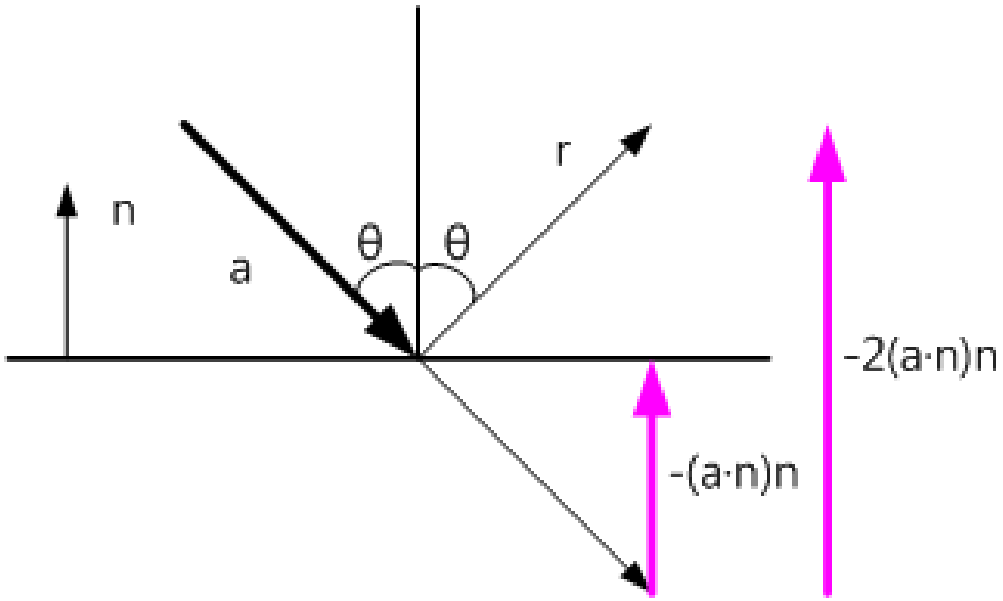
Practice: Build and Run BasicHLSL Step2

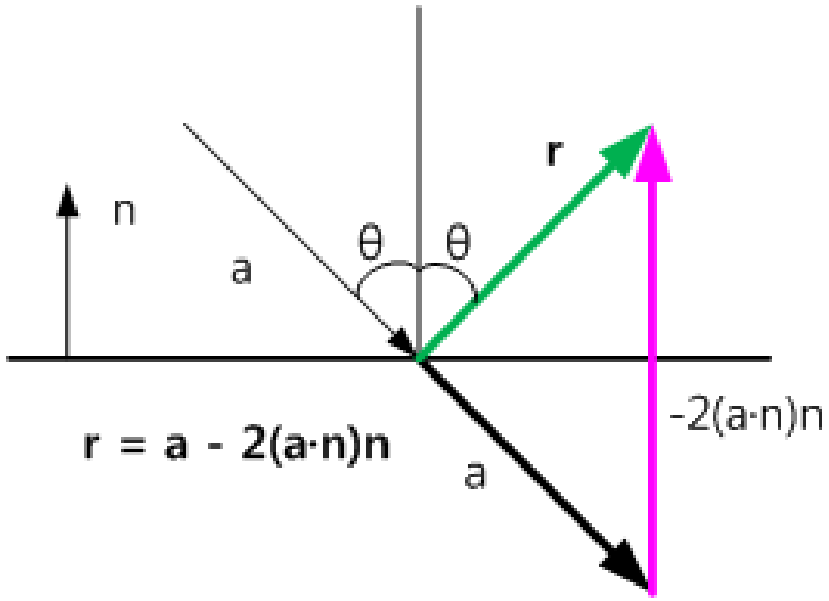
Step3 : Specular Light



Reflection vector

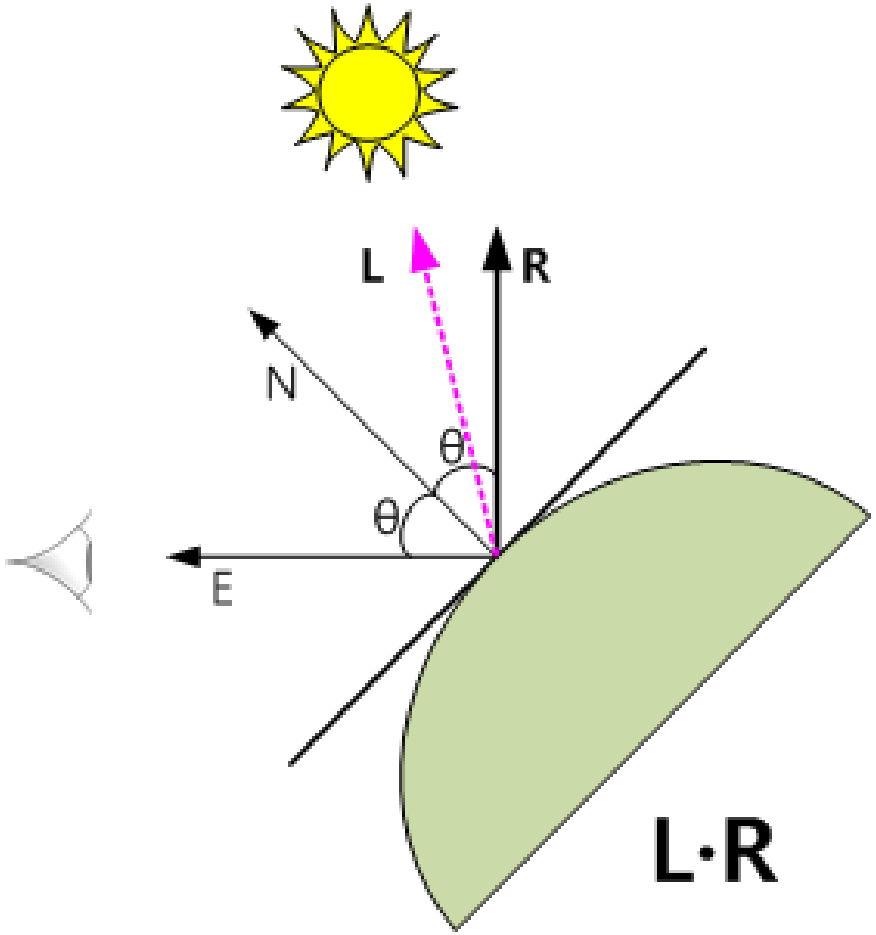
✓ **r = a - 2(a·n)n**





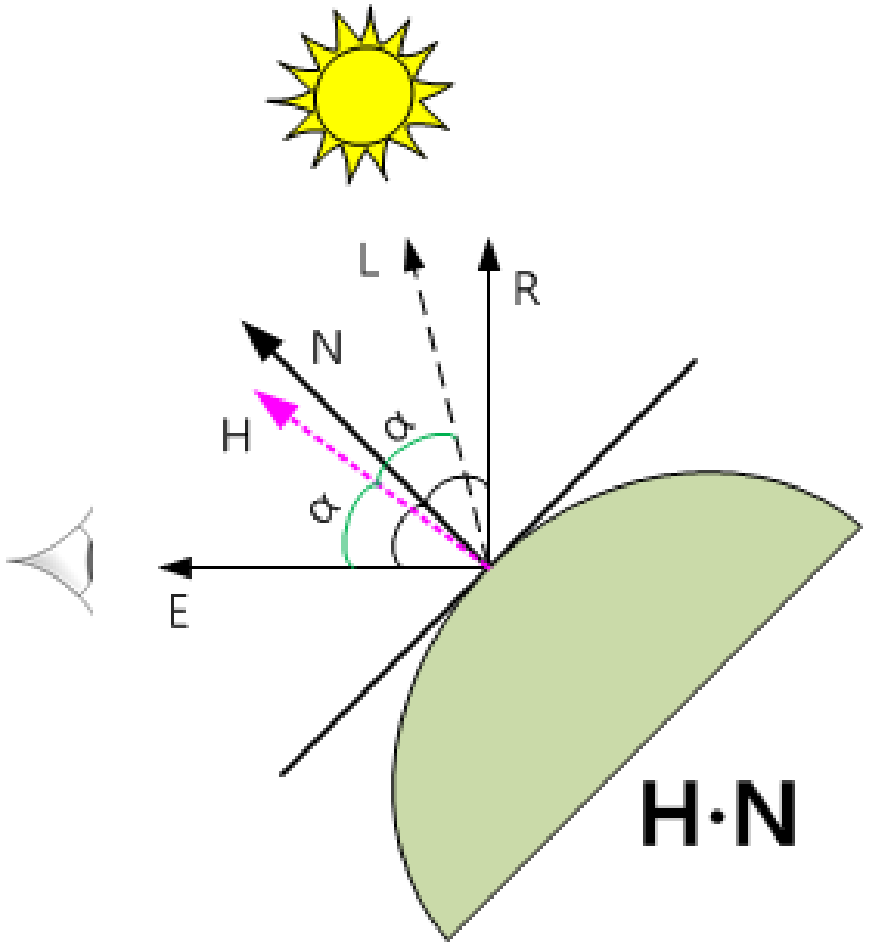
Specular Reflection

- ✓ Two vectors are used to calculate the specular component—the Light vector L and the reflection vector R.
- ✓ The more L is aligned with R, the brighter the specular light should be.



Half Vector

- ✓ If one calculates a *halfway vector* between the viewer and light-source vectors we can replace **L·R** with **H·N**.

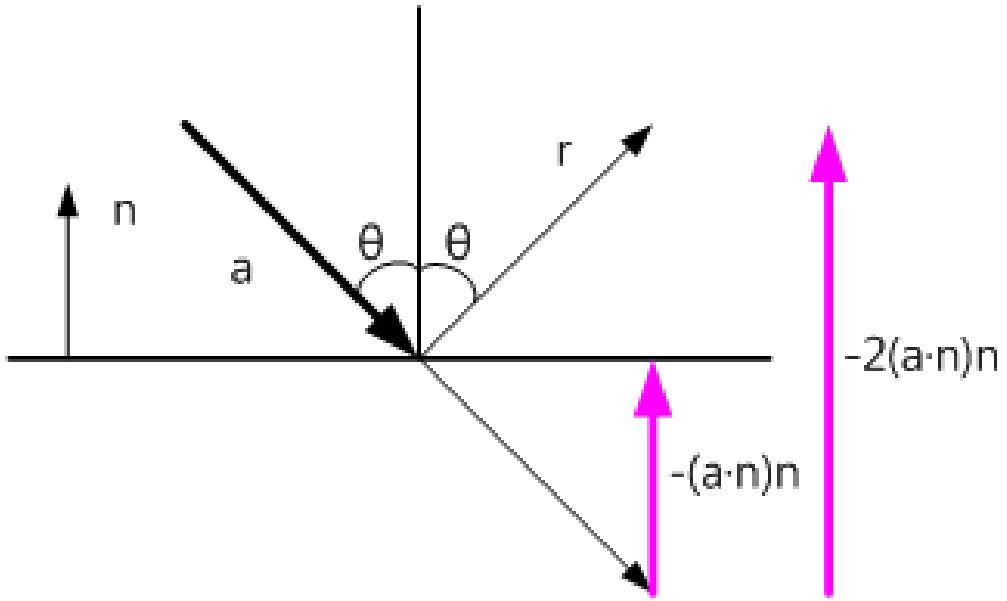


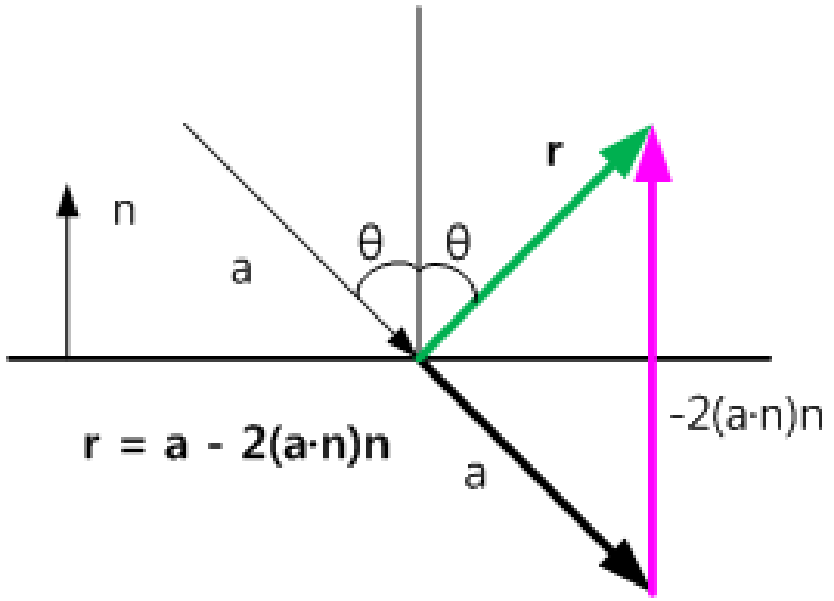
Shader Differences

```
float4 g_MaterialDiffuseColor;       // Material's diffuse color

float3 g_LightDir;                   // Light's direction in object space
float4 g_LightDiffuse;               // Light's diffuse color
float4 g_LightAmbient;               // Light's ambient color
//{{ step3
float3 g_vEyePos;                    // Eye position in Object space
//}} step3

texture g_MeshTexture;               // Color texture for mesh
```

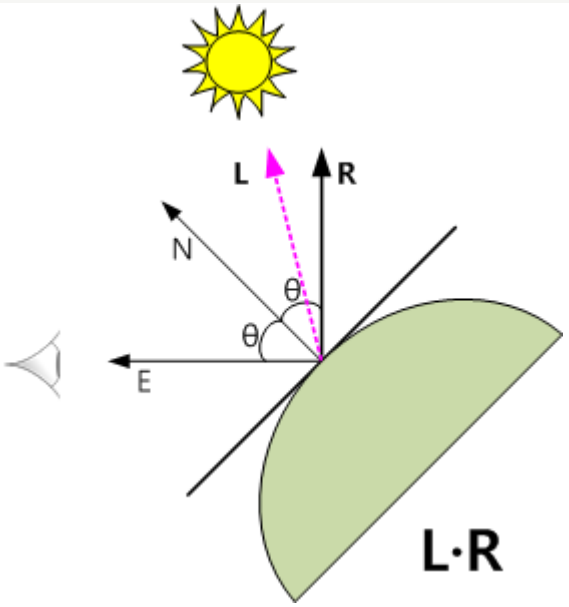
Shader Differences

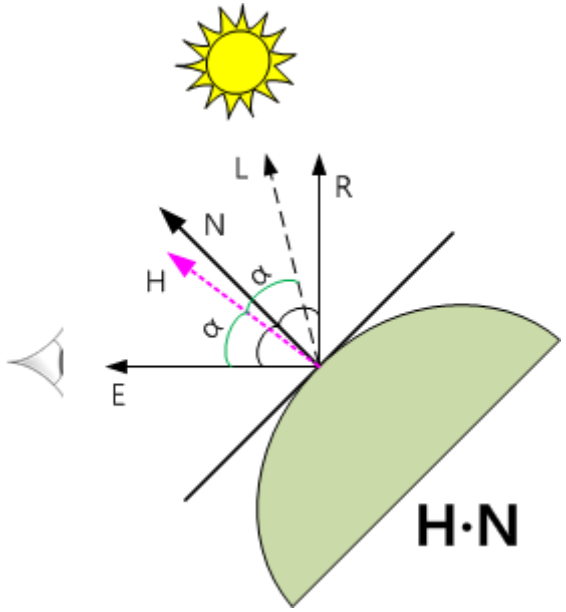
```
//}} step2

//{{ step3
float3 L = g_LightDir;
float3 eye = normalize( g_vEyePos - vPos.xyz );
float3 R = -eye + 2.0f * dot( vNormal, eye ) * vNormal; // reflection ve
float3 H = normalize( L + eye ); // half vector
//}} step3

// Compute simple directional lighting equation
float3 vTotalLightDiffuse = float3(0,0,0);
```





```
//}} step2

//{{ step3
//Output.Diffuse.rgb = g_MaterialDiffuseColor * vTotalLightDiffuse +
//    g_MaterialAmbientColor * g_LightAmbient;
// step3 case1 - use reflection vector.
//Output.Diffuse.rgb = g_MaterialDiffuseColor * vTotalLightDiffuse +
//    g_MaterialAmbientColor * g_LightAmbient + pow( max( 0, dot( L, R ) ), 10 );
// step3 case 2 - use half vector
Output.Diffuse.rgb = g_MaterialDiffuseColor * vTotalLightDiffuse +
    g_MaterialAmbientColor * g_LightAmbient + pow( max( 0, dot( vNormal, H ) ), 10 );
//}} step3
//Output.Diffuse.rgb = R;
Output.Diffuse.a = 1.0f;
```





Client: OnFrameRender() Differences

- ✓ Prepare WorldView and WorldViewInv matrices.

```
    D3DXMATRIXA16 mView;
    D3DXMATRIXA16 mProj;
    //{{ step3
    D3DXMATRIX      mWorldView;
    D3DXMATRIX      mWorldViewInv;
    //}} step3

    // Clear the render target and th
```

✓ Get Object space eye position.

```
mWorldViewProjection = mWorld * mView * mProj;
//{{ step3
mWorldView = mWorld * mView;
D3DXMatrixInverse( OUT &mWorldViewInv, NULL, &mWorldView );
D3DXVECTOR3 vEyePosLocal = D3DXVECTOR3( 0, 0, 0 );
D3DXVec3TransformCoord( OUT &vEyePosLocal, &vEyePosLocal, &mWorldViewInv );
//D3DXVec3TransformCoord( OUT &vEyePosLocal, g_Camera.GetEyePt(), &mWorldInv );
//}} step3

D3DXCOLOR arrowColor = D3DXCOLOR( 1, 1, 0, 1 );
```

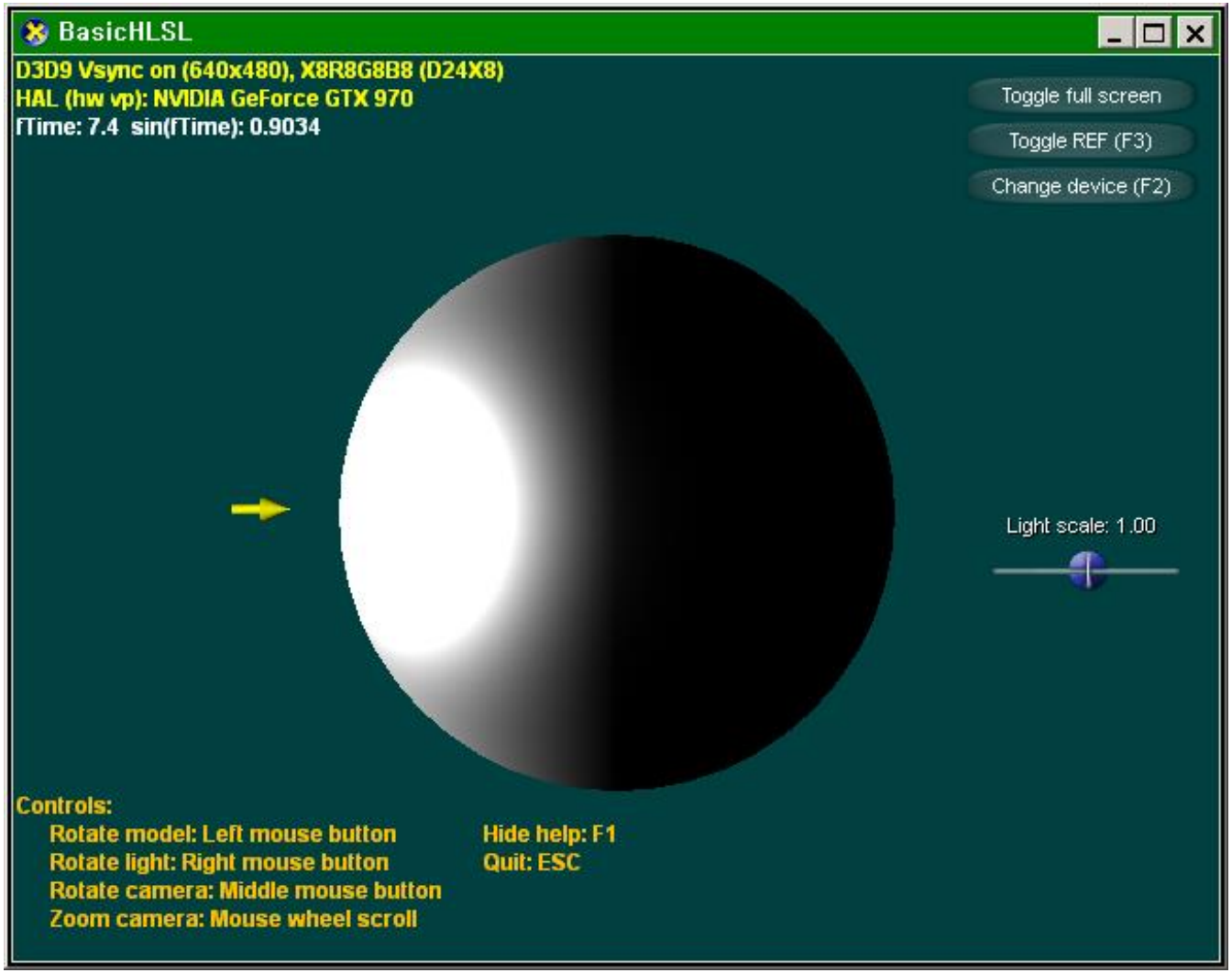
✓ Set Object space eye position.

```
V( g_pEffect->SetValue( "g_LightDir", &vLightDirObject, sizeof( D3DXVECTOR3 ) )
V( g_pEffect->SetValue( "g_LightDiffuse", &vLightDiffuse, sizeof( D3DXVECTOR4 )
//{{ step3
//V( g_pEffect->SetVector( "g_vEyePos", &vEyePosLocal ) );
V( g_pEffect->SetValue( "g_vEyePos", &vEyePosLocal, sizeof( D3DXVECTOR3 ) ) );
//}} step3

// Update the effect's variables.  Instead of using strings, it would
```

Practice: Build and Run BasicHLSL Step3

Step4 : Specular Light in Pixel Shader

- ✓ Specular light calculation in pixel shader to get more precise result.



Effect Differences

- ✓ Add Normal and Eye vector to VS_OUTPUT.

```
struct VS_OUTPUT
{
    float4 Position   : POSITION;   // vertex
    float4 Diffuse    : COLOR0;     // vertex
    float2 TextureUV  : TEXCOORD0;  // vertex
    //{{ step4
    float3 N : TEXCOORD1;
    float3 Eye : TEXCOORD2;
    //}} step4
};
```

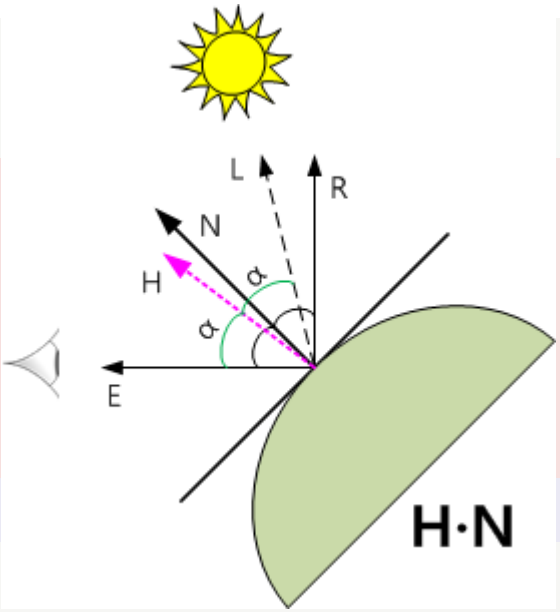
Vertex Shader Differences

- ✓ Pass Normal and Eye vector to Pixel shader.

```
    //}} step3
    //Output.Diffuse.rgb = R;
    Output.Diffuse.a = 1.0f;

    //{{ step4
    Output.N = vNormal.xyz;
    Output.Eye = g_vEyePos - vPos.xyz; // do not normalize
    //}} step4

    // Just copy the texture coordinate through
    if( bTexture )
```

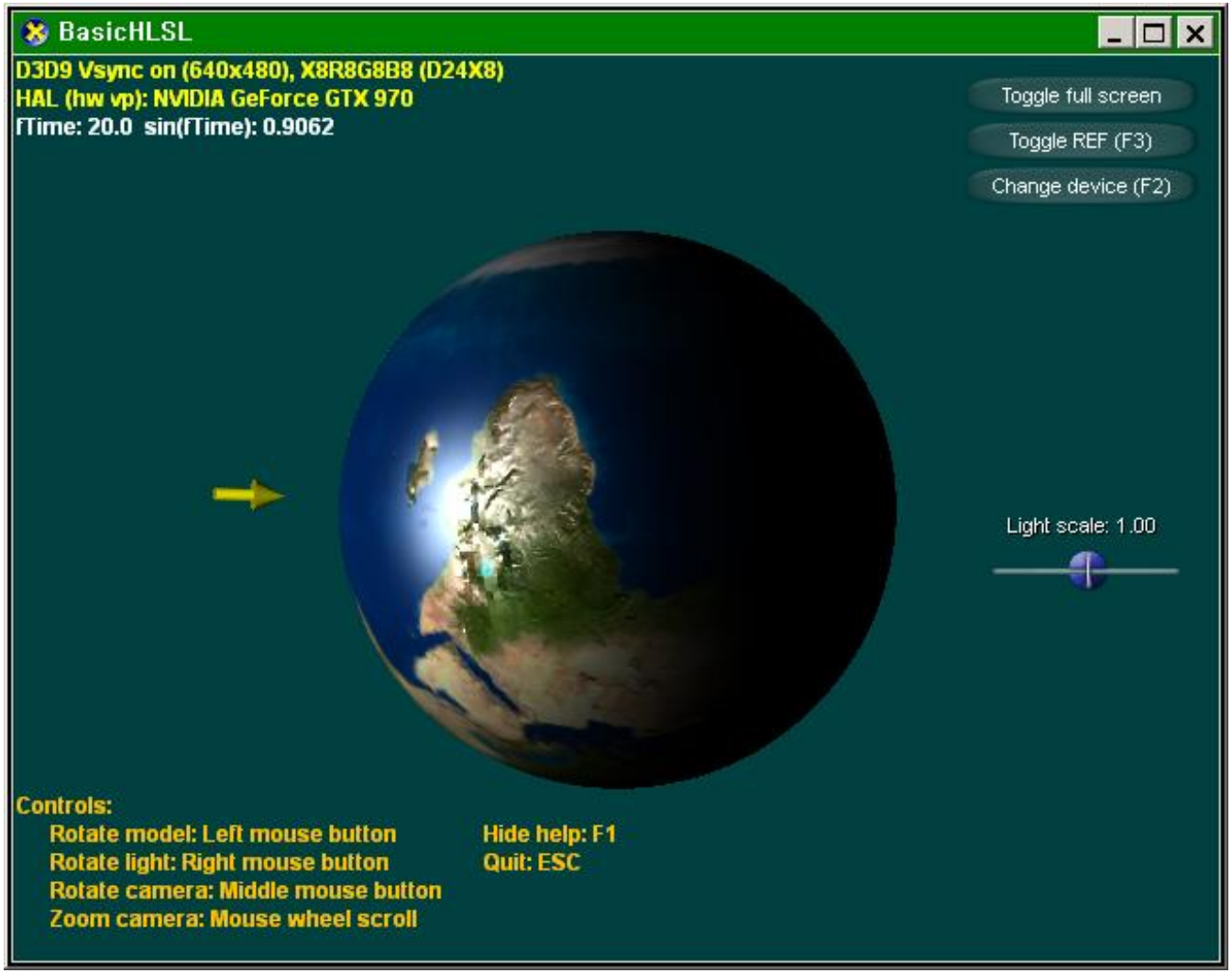
Pixel Shader Differences

```
PS_OUTPUT Output;

//{{ step4
float3 L = g_LightDir;
float3 H = normalize( L + normalize( In.Eye ) );
float3 N = normalize( In.N );
//}} step4

// Lookup mesh texture and modulate it with diffuse
if( bTexture )
    Output.RGBColor = tex2D( MeshTextureSampler, In.TextureUV ) * In.Diffuse;
else
    Output.RGBColor = In.Diffuse;
//{{ step4
Output.RGBColor += pow( max( 0, dot( N, H ) ), 10 );
//}} step4
```



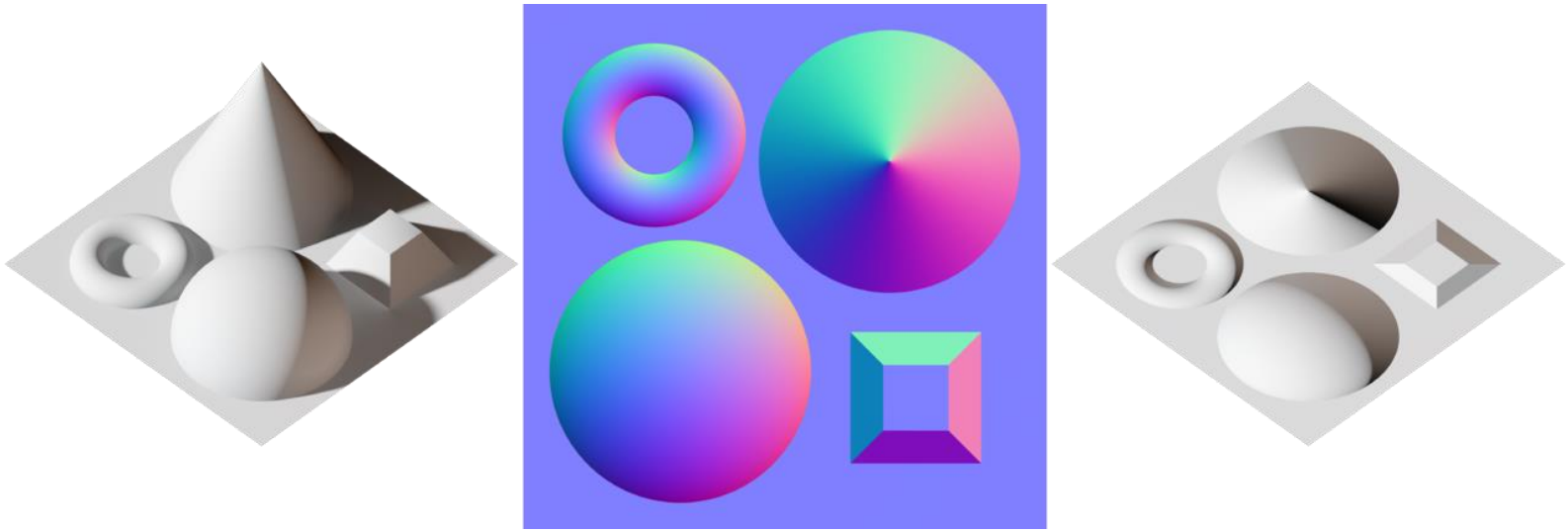
Practice: Build and Run BasicHLSL Step4

Step5 : Normal Mapping



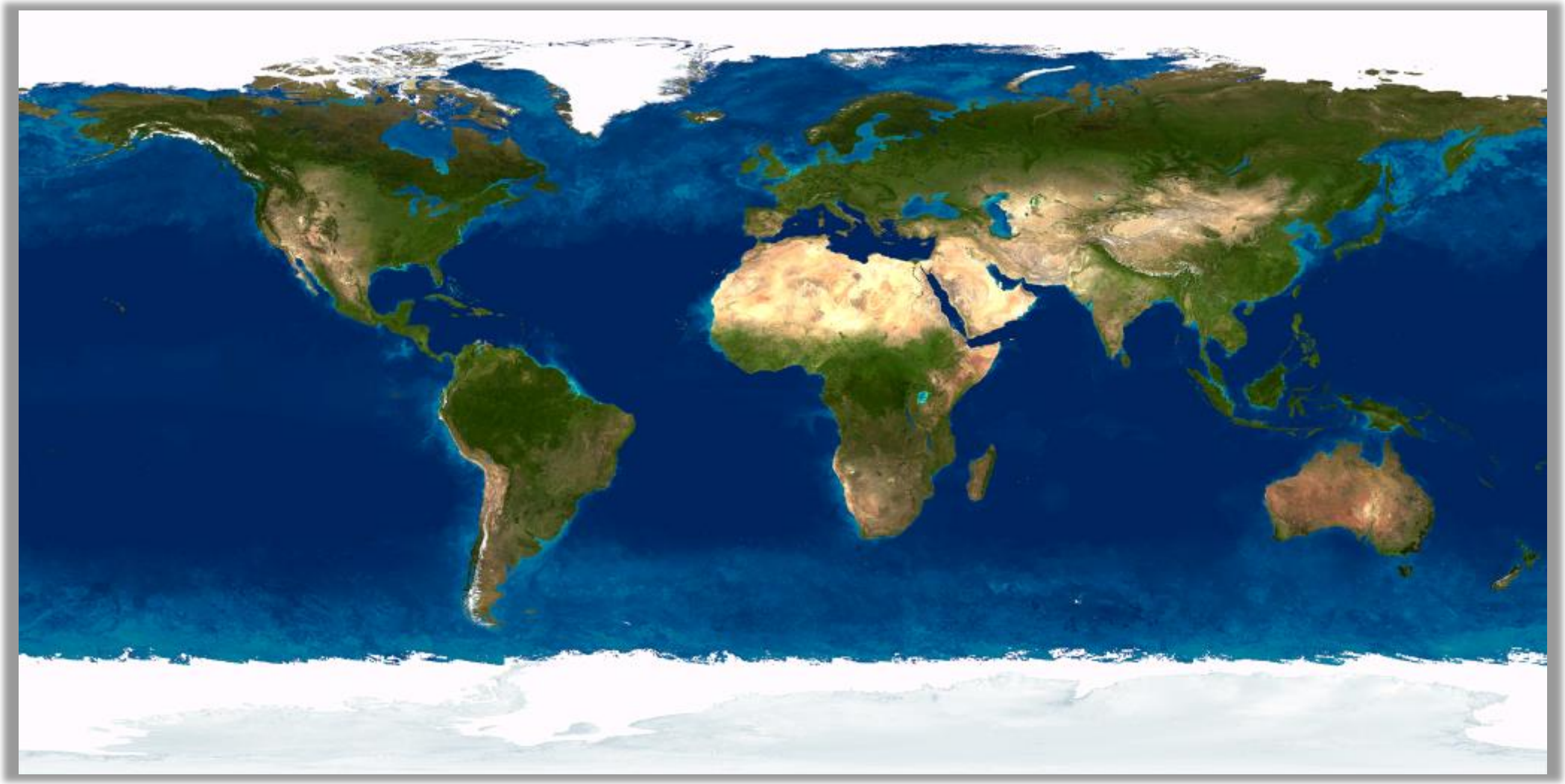
Normal Map

- ✓ In 3D computer graphics, **normal mapping** is a technique used for faking the lighting of bumps and dents – an implementation of bump mapping.
- ✓ It is used to add details without using more polygons.

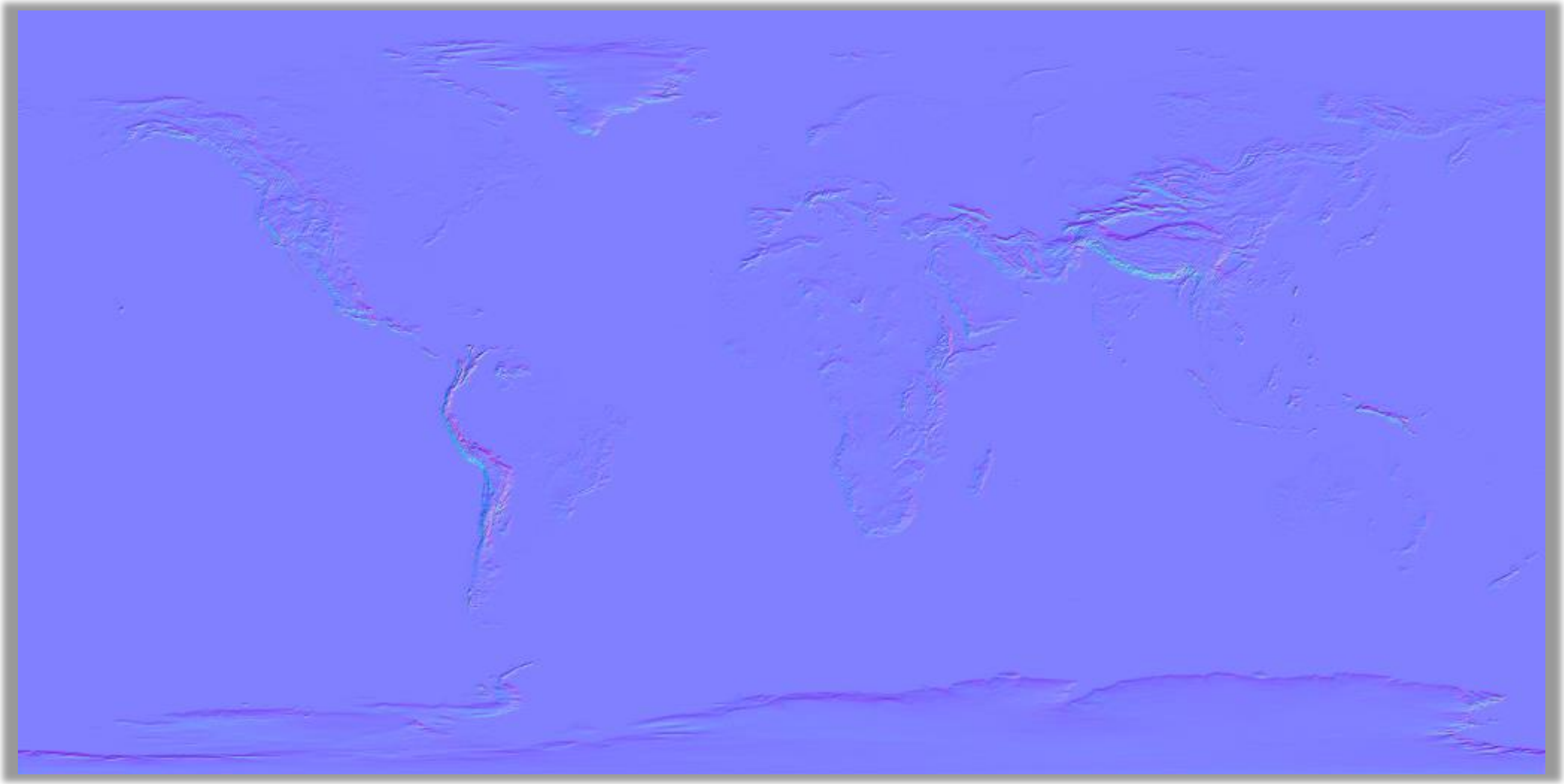


- Example of a normal map (center) with the scene it was calculated from (left) and the result when applied to a flat surface (right).

Diffuse Map



Normal Map



- ✓ RGB color of pixel in image file is a direction (x,y,z) in tangent space.

Effect Differences: Added normal map texture

```
texture g_MeshTexture;                    // Color texture for mesh
//{{ step5
texture g_MeshBumpTexture;                // Bump texture for mesh
//}} step5
```

```
//{{ step5
sampler MeshBumpTextureSampler =
sampler_state {
    Texture = <g_MeshBumpTexture>;
    MipFilter = LINEAR;
    MinFilter = LINEAR;
    MagFilter = LINEAR;
};
//}} step5
```

Added Light vector in tangent space

```
//--------------------------------------------------------
struct VS_OUTPUT
{
    float4 Position   : POSITION;   // vertex p
    float4 Diffuse    : COLOR0;     // vertex d
    float2 TextureUV  : TEXCOORD0;  // vertex t
    //{{ step4
    float3 N : TEXCOORD1;
    float3 Eye : TEXCOORD2;
    //{{ step5
    float3 L : TEXCOORD3;
    //}} step5
    //}} step4
};
```

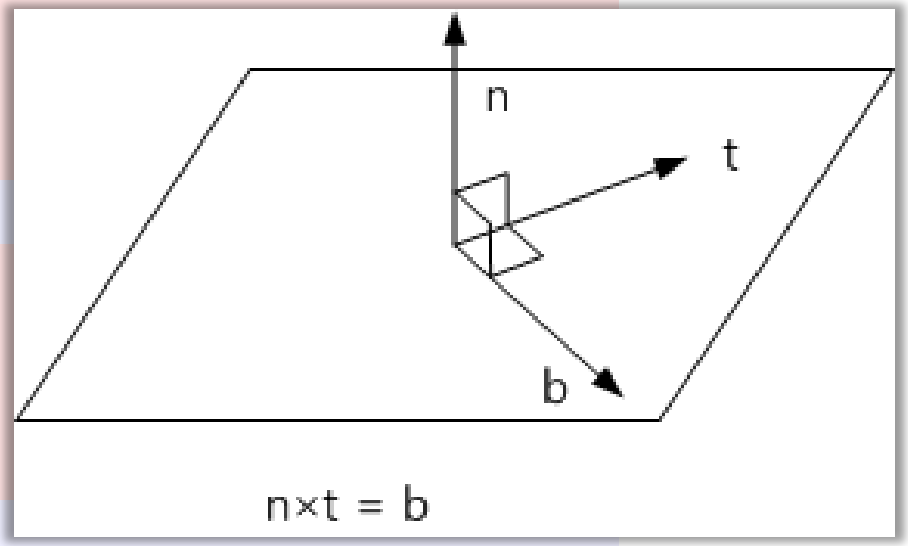
Vertex Shader Differences

```
//--------------------------------------------------------------------------
// This shader computes standard transform and lighting
//--------------------------------------------------------------------------
VS_OUTPUT RenderSceneVS( float4 vPos : POSITION,
                         float3 vNormal : NORMAL,
                         //{{ step5
                         float3 vTangent : TANGENT,
                         //}} step5
                         float2 vTexCoord0 : TEXCOORD0,
                         uniform int nUnused,
                         uniform bool bTexture,
                         uniform bool bAnimate )
{
```

43 FIL

✓ Eye and Light vectors must be transformed to tangent space.

```
    //}} step4
    //{{ step5
    float3 N = vNormal;
    float3 T = vTangent;
    float3 B = cross( N, T );

    float3 E = g_vEyePos - vPos.xyz; // eye vector
    Output.Eye.x = dot( E, T );
    Output.Eye.y = dot( E, B );
    Output.Eye.z = dot( E, N );

    Output.L.x = dot( L, T );
    Output.L.y = dot( L, B );
    Output.L.z = dot( L, N );
    //}} step5

    // Just copy the texture coordinate through
    if( bTexture )
```



Pixel Shader

```
PS_OUTPUT RenderScenePS( VS_OUTPUT In, uniform bool bTexture )
{
    PS_OUTPUT Output;

    float3 N = 2.0f * tex2D( MeshBumpTextureSampler, In.TextureUV ).xyz - 1.0f;
    float3 L = normalize( In.L );
    float3 R = reflect( -normalize( In.Eye ), N ); // reflection vector

    // Lookup mesh texture and modulate it with diffuse
    if( bTexture ) {
        Output.RGBColor = tex2D( MeshTextureSampler, In.TextureUV ) *In.Diffuse * max(
0, dot( N, L ) );
    } else {
        Output.RGBColor = In.Diffuse * max( 0, dot( N, L ) );
    }//if.. else..
    Output.RGBColor += pow( max( 0, dot( R, L ) ), 10 );

    return Output;
}
```

Client: Added normal map texture

```
ID3DXMesh*                  g_pMesh = NULL;          // Mesh object
IDirect3DTexture9*          g_pMeshTexture = NULL;   // Mesh texture
//{{ step5
IDirect3DTexture9*          g_pMeshBumpTexture = NULL;  // Mesh bump textu
//}} step3
CDXUTDialogResourceManager  g_DialogResourceManager; // manager for shared
CD3DSettingsDlg             g_SettingsDlg;           // Device settings dia
CDXUTDialog                 g_HUD;                   // manages the 3D UI
```

OnCreateDevice() Differences

```
        NULL, NULL, &g_pMeshTexture ) );

    //{{ step5
    V_RETURN( DXUTFindDXSDKMediaFileCch( str, MAX_PATH, L"misc\\Earth_NormalMap.dds" ) );
    V_RETURN( D3DXCreateTextureFromFileEx( pd3dDevice, str, D3DX_DEFAULT, D3DX_DEFAULT,
        D3DX_DEFAULT, 0, D3DFMT_UNKNOWN, D3DPOOL_MANAGED,
        D3DX_DEFAULT, D3DX_DEFAULT, 0,
        NULL, NULL, &g_pMeshBumpTexture ) );
    //}} step5

    // Set effect variables as needed
    D3DXCOLOR colorMtrlDiffuse( 1.0f, 1.0f, 1.0f, 1.0f );
    D3DXCOLOR colorMtrlAmbient( 0.35f, 0.35f, 0.35f, 0 );

    V_RETURN( g_pEffect->SetValue( "g_MaterialAmbientColor", &colorMtrlAmbient, sizeof( D3
    V_RETURN( g_pEffect->SetValue( "g_MaterialDiffuseColor", &colorMtrlDiffuse, sizeof( D3
    V_RETURN( g_pEffect->SetTexture( "g_MeshTexture", g_pMeshTexture ) );
    //{{ step5
    V_RETURN( g_pEffect->SetTexture( "g_MeshBumpTexture", g_pMeshBumpTexture ) );
    //}} step5
```

OnResetDevice() Differences()

```
D3DVERTEXELEMENT9 decl[] =
{
    { 0, 0, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_POSITION, 0 },
    { 0, 12, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_NORMAL, 0 },
    { 0, 24, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TANGENT, 0 },
    { 0, 36, D3DDECLTYPE_FLOAT2, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 0 },
    D3DDECL_END()
};//decl[]

if( g_pMesh ) {
    LPD3DXMESH pMesh;

    g_pMesh->CloneMesh(
        g_pMesh->GetOptions(), decl,
        pd3dDevice, &pMesh );
    D3DXComputeNormals( pMesh, NULL );
    D3DXComputeTangent( pMesh, 0, 0, 0, TRUE, NULL );

    SAFE_RELEASE( g_pMesh );
    g_pMesh = pMesh;
}//if
```

OnDestroyDevice() Differences

```
    SAFE_RELEASE( g_pEffect );
    SAFE_RELEASE( g_pFont );
    SAFE_RELEASE( g_pMesh );
    SAFE_RELEASE( g_pMeshTexture );
    //{{ step5
    SAFE_RELEASE( g_pMeshBumpTexture );
    //}} step5
}
```

References

MY BRIGHT FUTURE

동서대학교

DSU Dongseo University 동서대학교

DSU Dongseo University 동서대학교