MY BRIGHT FUTURE

DSU Dongseo University 동서대학교

Beginning Direct3D Game Programming:

7. Lights and Materials

jintaeks@gmail.com

Division of Digital Contents, DongSeo University.

December 2019

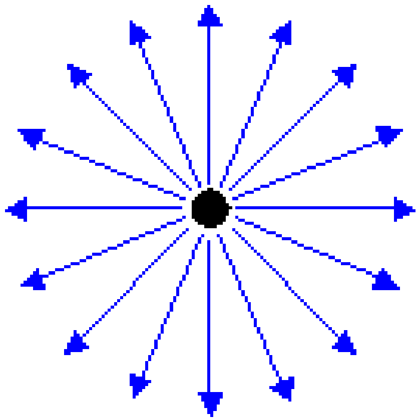
DSU Dongseo University 동서대학교

Light Types

- ✓ The light type property defines which type of light source you're using.
- ✓ The light type is set by using a value from the D3DLIGHTTYPE C++ enumeration in the Type member of the light's D3DLIGHT9 structure.
- ✓ There are three types of lights in Direct3D:
  - point lights
  - spotlights
  - directional lights

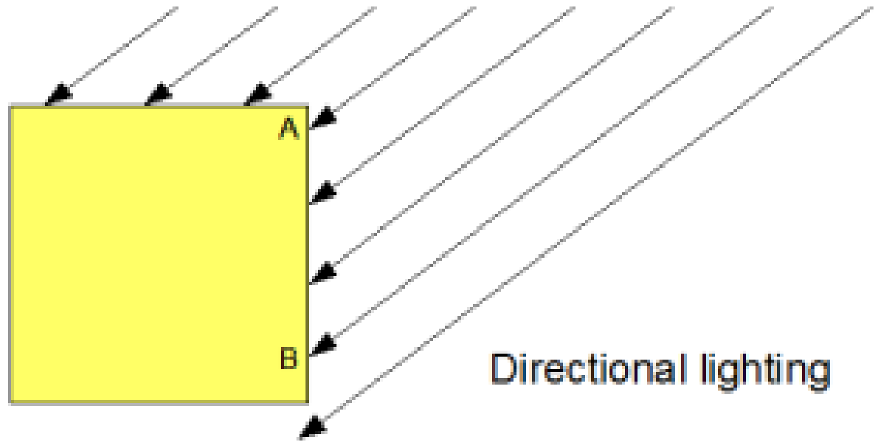
Point Light

- ✓ Point lights have color and position within a scene, but no single direction.
- ✓ They give off light equally in all directions.
- ✓ A light bulb is a good example of a point light.



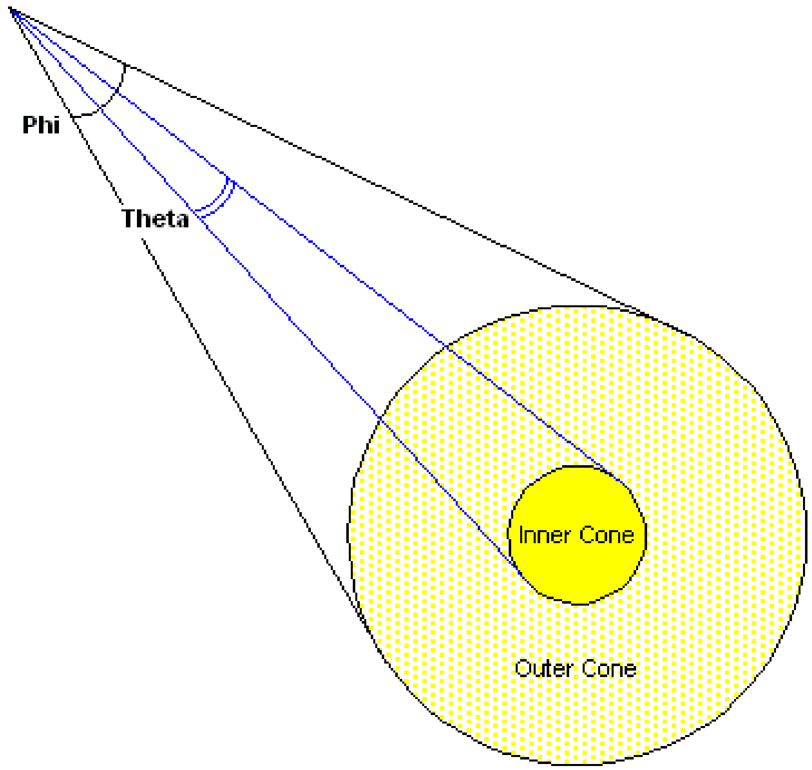
Directional Light

- ✓ Directional lights have only color and direction, not position.
- ✓ They emit parallel light. This means that all light generated by directional lights travels through a scene in the same direction.
- ✓ Imagine a directional light as a light source at near infinite distance, such as the sun.

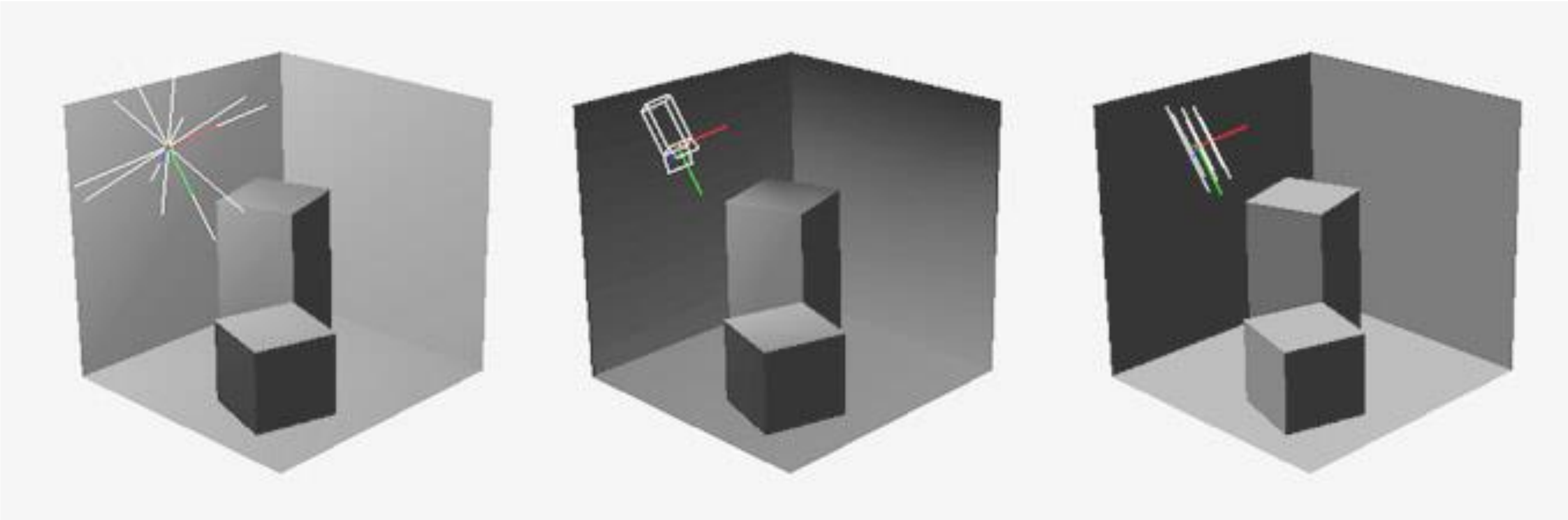


Spot Light

- ✓ Spotlights have **color**, **position**, and **direction** in which they emit light.
- ✓ Light emitted from a spotlight is made up of a **bright inner cone** and a **larger outer cone**, with the light intensity diminishing between the two.



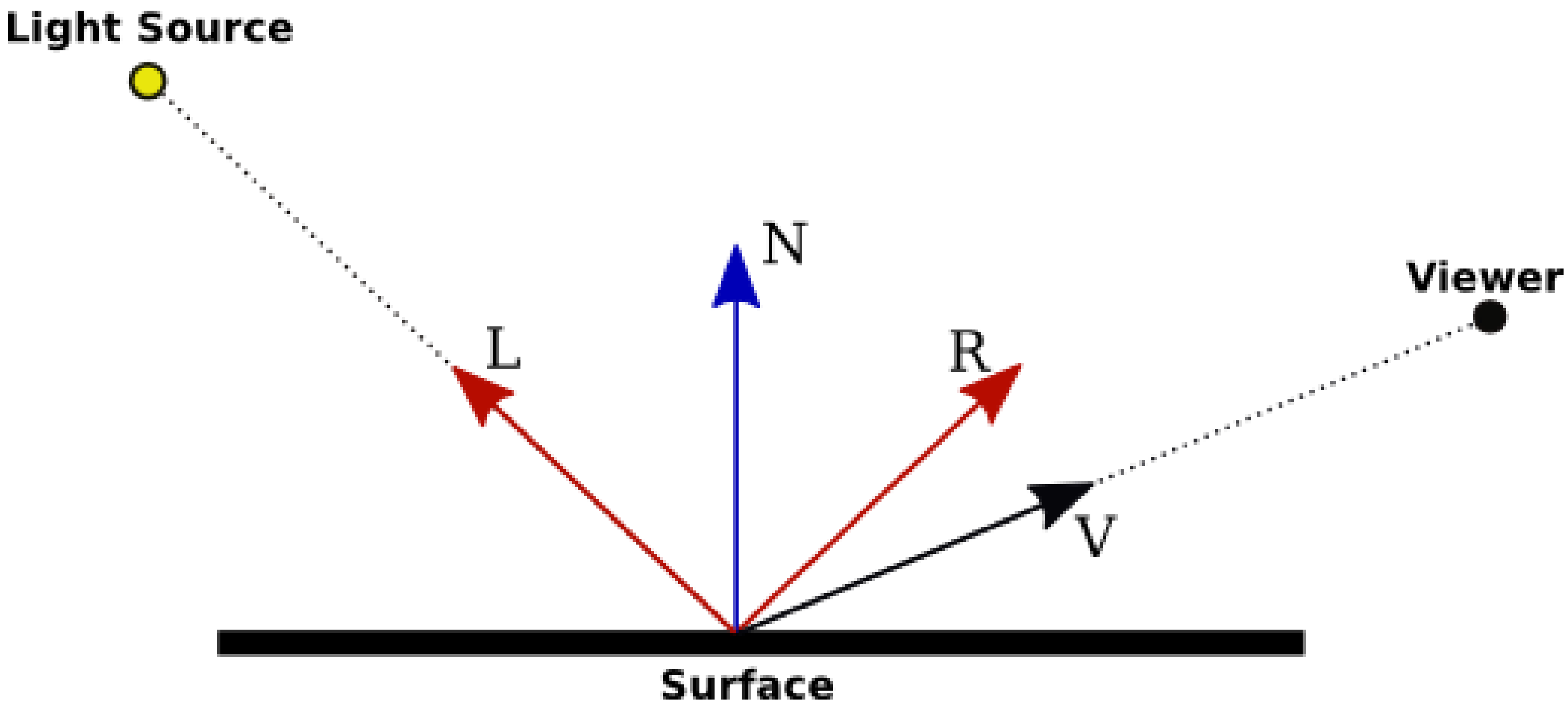
Comparison between Light Types



Materials

- ✓ Materials describe how polygons reflect light or appear to emit light in a 3D scene.
- ✓ Material properties detail a material's diffuse reflection, ambient reflection, light emission, and specular highlight characteristics.
- ✓ Direct3D uses the D3DMATERIAL9 structure to carry all material property information.

✓ Ambient lighting provides constant lighting for a scene.



Global Illumination = **Ambient** Light + **Diffuse** Light + **Specular** Light + **Emissive** Light

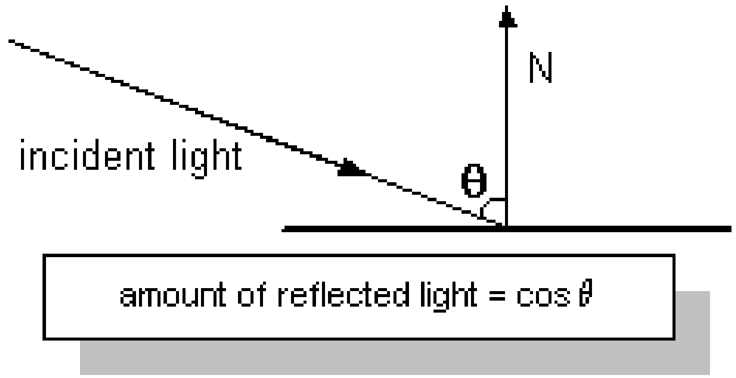
```
class KLight {
public:
    enum LightType {
        LT_DIRECTIONAL,
        LT_POINT,
        LT_SPOT,
    };

    KRgb            m_ambient = KRgb(0, 0, 0);
    KRgb            m_diffuse = KRgb(1, 1, 1);
    KRgb            m_specular = KRgb(0, 0, 0);
    KRgb            m_emissive = KRgb(0, 0, 0);
    LightType       m_lightType = LT_DIRECTIONAL;
    KVector3        m_dir = KVector3(0, 0, -1);

public:
    CONSTRUCTOR     KLight();
    DESTRUCTOR      ~KLight();
};
```

```
class KMaterial
{
public:
    KRgb            m_ambient = KRgb(0,0,0);
    KRgb            m_diffuse = KRgb(1,1,1);
    KRgb            m_specular = KRgb(0,0,0);
    KRgb            m_emissive = KRgb(0,0,0);
public:
    CONSTRUCTOR      KMaterial();
    DESTRUCTOR      ~KMaterial();
};
```

```
KVector3 normal;
normal = Cross( m_vertexBuffer[i1] - m_vertexBuffer[i0]
              , m_vertexBuffer[i2] - m_vertexBuffer[i0] );
normal.Normalize();

float shade = Dot(light.m_dir, normal);

KRgb color = light.m_ambient
    + light.m_diffuse * mtrl.m_diffuse * shade
    + light.m_specular * mtrl.m_specular;
color[0] = __min(1.0f, color[0]);
color[1] = __min(1.0f, color[1]);
color[2] = __min(1.0f, color[2]);
```

Diffuse and Ambient Reflection

- ✓ The Diffuse and Ambient members of the **D3DMATERIAL9** structure describe how a material reflects the ambient and diffuse light in a scene.
- ✓ Because most scenes contain much more diffuse light than ambient light, diffuse reflection plays the largest part in determining color.

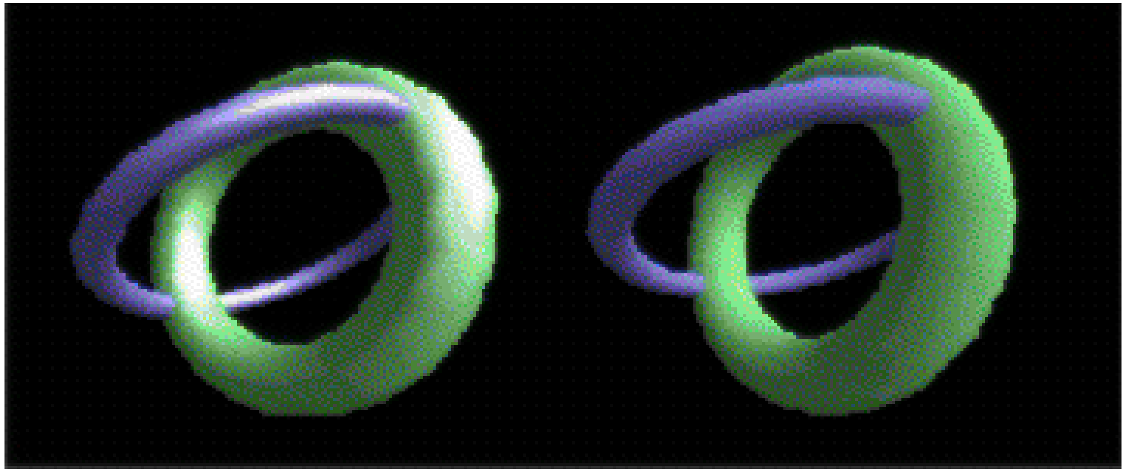


Emission

- ✓ Materials can be used to make a rendered object appear to be self-luminous.
- ✓ The Emissive member of the **D3DMATERIAL9** structure is used to describe the color and transparency of the emitted light.

Specular Reflection

- ✓ Specular reflection creates highlights on objects, making them appear shiny.
- ✓ The **D3DMATERIAL9** structure contains two members that describe the specular highlight color as well as the material's overall shininess.



Setting Material Properties

```
D3DMATERIAL9 mat;

// Set the RGBA for diffuse reflection.
mat.Diffuse.r = 0.5f;
mat.Diffuse.g = 0.0f;
mat.Diffuse.b = 0.5f;
mat.Diffuse.a = 1.0f;

// Set the RGBA for ambient reflection.
mat.Ambient.r = 0.5f;
mat.Ambient.g = 0.0f;
mat.Ambient.b = 0.5f;
mat.Ambient.a = 1.0f;
```

```
// Set the color and sharpness of specular highlights.
mat.Specular.r = 1.0f;
mat.Specular.g = 1.0f;
mat.Specular.b = 1.0f;
mat.Specular.a = 1.0f;
mat.Power = 50.0f;

// Set the RGBA for emissive color.
mat.Emissive.r = 0.0f;
mat.Emissive.g = 0.0f;
mat.Emissive.b = 0.0f;
mat.Emissive.a = 0.0f;
```

- ✓ After preparing the D3DMATERIAL9 structure, you apply the properties by calling the IDirect3DDevice9::SetMaterial method of the rendering device.

```
// This code example uses the material properties defined for
// the mat variable earlier in this topic. The pd3dDev is assumed
// to be a valid pointer to an IDirect3DDevice9 interface.
HRESULT hr;
hr = pd3dDev->SetMaterial(&mat);
if(FAILED(hr))
{
    // Code to handle the error goes here.
}
```

Mathematics of Lighting

✓ The Direct3D Light Model covers ambient, diffuse, specular, and emissive lighting.

Global Illumination = **Ambient** Light + **Diffuse** Light + **Specular** Light + **Emissive** Light

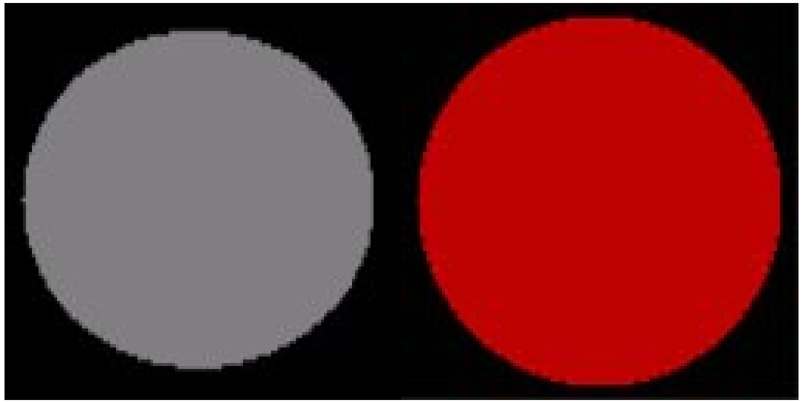
Ambient Lighting

- ✓ Ambient lighting provides constant lighting for a scene.
- ✓ It lights all object vertices the same because it is not dependent on any other lighting factors such as vertex normals, light direction, light position, range, or attenuation.

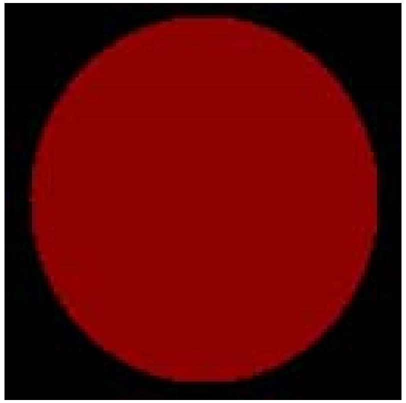
```
#define GRAY_COLOR     0x00bfbfbf

// create material
D3DMATERIAL9 mtrl;
ZeroMemory(&mtrl, sizeof(mtrl));
mtrl.Ambient.r = 0.75f;
mtrl.Ambient.g = 0.0f;
mtrl.Ambient.b = 0.0f;
mtrl.Ambient.a = 0.0f;
m_pd3dDevice->SetMaterial(&mtrl);
m_pd3dDevice->SetRenderState(D3DRS_AMBIENT, GRAY_COLOR);
```

Example

- ✓ These two images show the material color, which is gray, and the light color, which is bright red.



- ✓ The resulting scene is shown below.



Diffuse Lighting

✓ The system considers two reflection types, diffuse and specular, and uses a different formula to determine how much light is reflected for each.

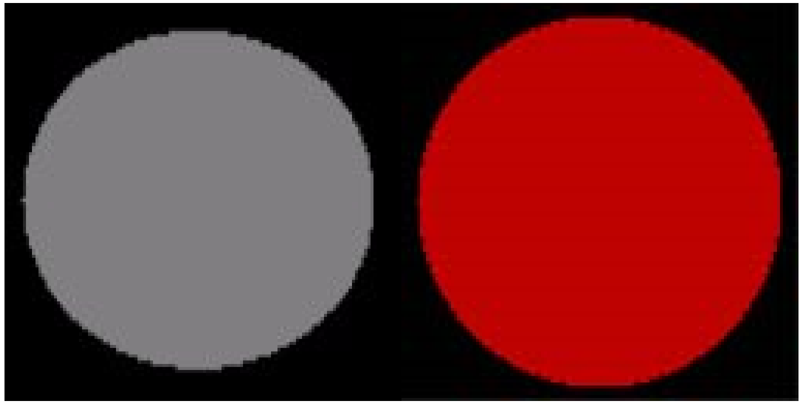
$$\text{Diffuse Lighting} = \text{sum}[C_d * L_d * (N \cdot L_{dir}) * \mathbf{Atten} * \text{Spot}]$$

Parameter	Default value	Type	Description
sum	N/A	N/A	Summation of each light's diffuse component.
C_d	(0,0,0,0)	D3DCOLORVALUE	Diffuse color.
L_d	(0,0,0,0)	D3DCOLORVALUE	Light diffuse color.
N	N/A	D3DVECTOR	Vertex normal
L_{dir}	N/A	D3DVECTOR	Direction vector from object vertex to the light.
Atten	N/A	FLOAT	Light attenuation. See Attenuation and Spotlight Factor (Direct3D 9).
Spot	N/A	FLOAT	Spotlight factor. See **Attenuation and Spotlight Factor (Direct3D 9).**

```
D3DMATERIAL9 mtrl;
ZeroMemory( &mtrl, sizeof(mtrl) );

D3DLIGHT9 light;
ZeroMemory( &light, sizeof(light) );
light.Type = D3DLIGHT_DIRECTIONAL;

D3DXVECTOR3 vecDir;
vecDir = D3DXVECTOR3(0.5f, 0.0f, -0.5f);
D3DXVec3Normalize( (D3DXVECTOR3*)&light.Direction, &vecDir );
```

```
// set directional light diffuse color
light.Diffuse.r = 1.0f;
light.Diffuse.g = 1.0f;
light.Diffuse.b = 1.0f;
light.Diffuse.a = 1.0f;
m_pd3dDevice->SetLight( 0, &light );
m_pd3dDevice->LightEnable( 0, TRUE );

// if a material is used, SetRenderState must be used
// vertex color = light diffuse color * material diffuse color
mtrl.Diffuse.r = 0.75f;
mtrl.Diffuse.g = 0.0f;
mtrl.Diffuse.b = 0.0f;
mtrl.Diffuse.a = 0.0f;
m_pd3dDevice->SetMaterial( &mtrl );
m_pd3dDevice->SetRenderState(D3DRS_DIFFUSEMATERIALSOURCE,
D3DMCS_MATERIAL);
```

- ✓ These two images show the material color, which is gray, and the light color, which is bright red.



- ✓ The resulting scene is shown below.

✓ Combining the diffuse lighting with the ambient lighting from the previous example shades the entire surface of the object.

Global Illumination = **Ambient** Light + **Diffuse** Light + **Specular** Light + **Emissive** Light

Specular Lighting

✓ Modeling specular reflection requires that the system not only know in what direction light is traveling, but also the direction to the **viewer's eye**.

Specular Lighting = C_s * sum[L_s * (N • H)^P * Atten * Spot]

Parameter	Default value	Type	Description
C_s	(0,0,0,0)	D3DCOLORVALUE	Specular color.
sum	N/A	N/A	Summation of each light's specular component.
N	N/A	D3DVECTOR	Vertex normal.
H	N/A	D3DVECTOR	Half way vector. See the section on the halfway vector.
P	0.0	FLOAT	Specular reflection power. Range is 0 to +infinity
L_s	(0,0,0,0)	D3DCOLORVALUE	Light specular color.
Atten	N/A	FLOAT	Light attenuation value. See Attenuation and Spotlight Factor (Direct3D 9).
Spot	N/A	FLOAT	Spotlight factor. See **Attenuation and Spotlight Factor (Direct3D 9)**.

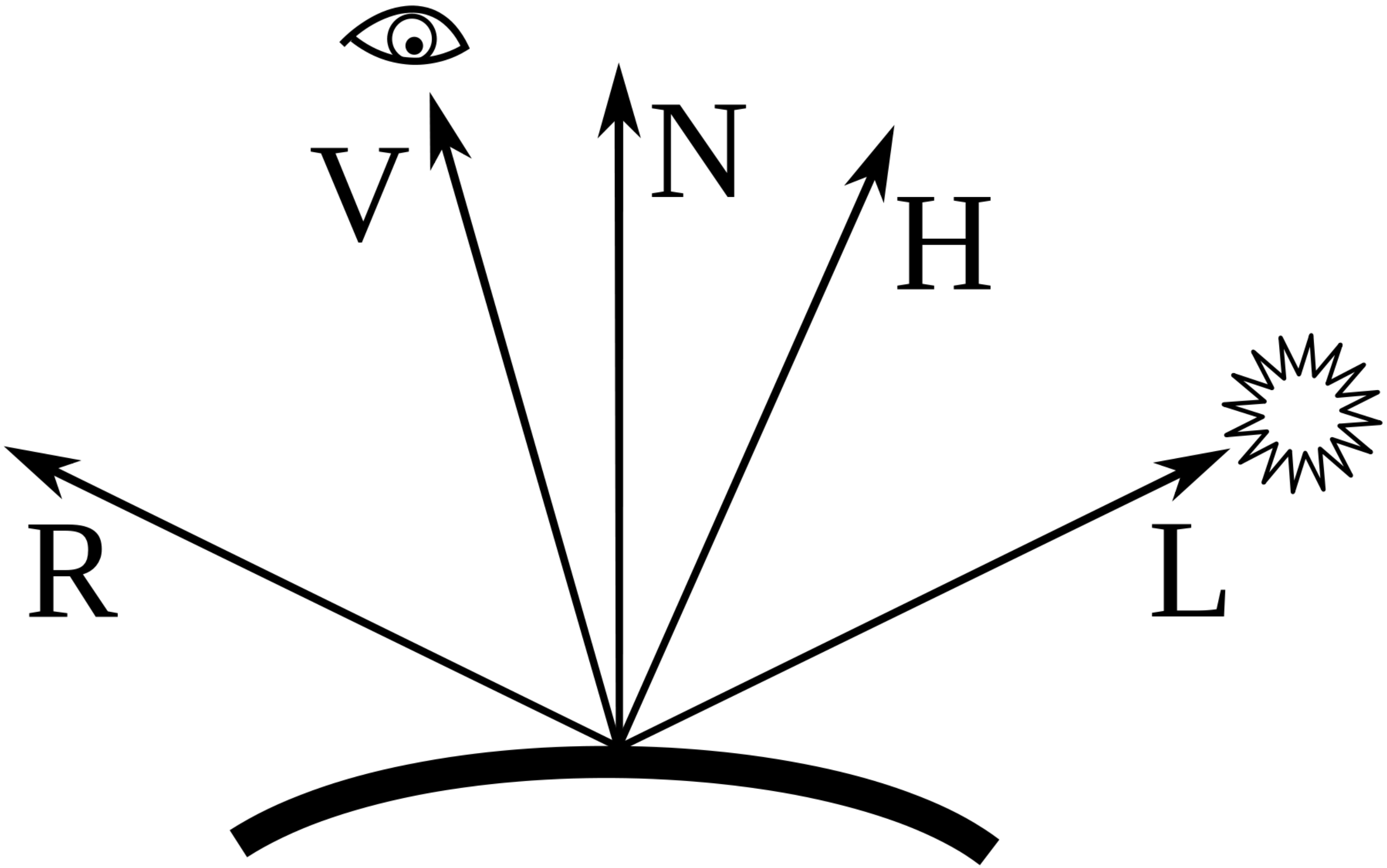
Halfway Vector

✓ The halfway vector (H) exists midway between two vectors: the vector from an object vertex to the light source, and the vector from an object vertex to the camera position.

$H = norm(norm(C_p - V_p) + L_{dir})$

Parameter	Default value	Type	Description
C_p	N/A	D3DVECTOR	Camera position.
V_p	N/A	D3DVECTOR	Vertex position.
L_{dir}	N/A	D3DVECTOR	Direction vector from vertex position to the light position.

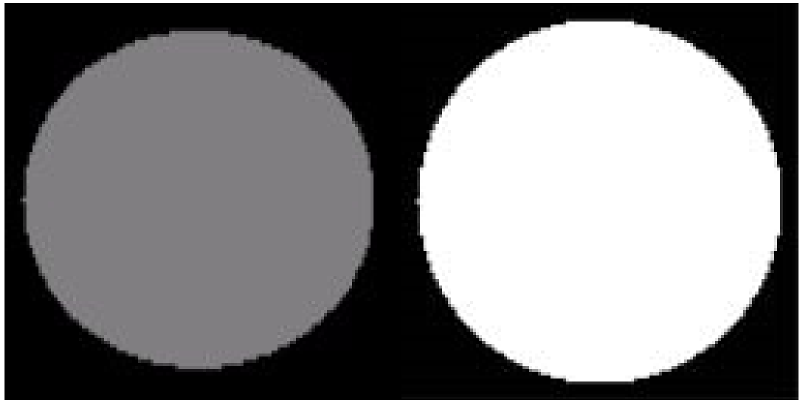
$$\text{Specular Lighting} = C_s * \text{sum}[L_s * (N \bullet H)^P * \text{Atten} * \text{Spot}]$$



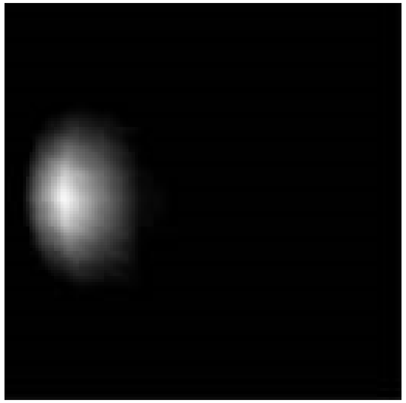
```
D3DMATERIAL9 mtrl;
ZeroMemory( &mtrl, sizeof(mtrl) );

D3DLIGHT9 light;
ZeroMemory( &light, sizeof(light) );
light.Type = D3DLIGHT_DIRECTIONAL;

D3DXVECTOR3 vecDir;
vecDir = D3DXVECTOR3(0.5f, 0.0f, -0.5f);
D3DXVec3Normalize( (D3DXVECTOR3*)&light.Direction, &vecDir );

light.Specular.r = 1.0f;
light.Specular.g = 1.0f;
light.Specular.b = 1.0f;
light.Specular.a = 1.0f;
```

```
light.Range = 1000;
light.Falloff = 0;
light.Attenuation0 = 1;
light.Attenuation1 = 0;
light.Attenuation2 = 0;
m_pd3dDevice->SetLight( 0, &light );
m_pd3dDevice->LightEnable( 0, TRUE );
m_pd3dDevice->SetRenderState( D3DRS_SPECULARENABLE, TRUE );

mtrl.Specular.r = 0.5f;
mtrl.Specular.g = 0.5f;
mtrl.Specular.b = 0.5f;
mtrl.Specular.a = 0.5f;
mtrl.Power = 20;
m_pd3dDevice->SetMaterial( &mtrl );
m_pd3dDevice->SetRenderState(D3DRS_SPECULARMATERIALSOURCE,
D3DMCS_MATERIAL);
```

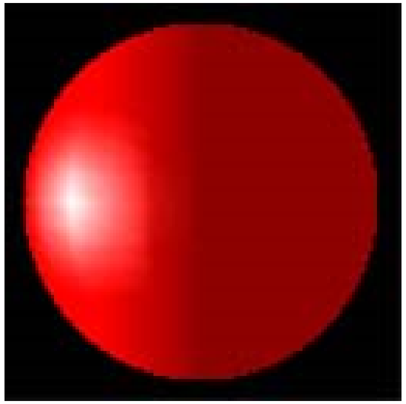
- ✓ These two images show the specular material color, which is gray, and the specular light color, which is white.



- ✓ The resulting specular highlight is shown below.



- ✓ Combining the specular highlight with the ambient and diffuse lighting produces the following image. With all three types of lighting applied, this more clearly resembles a realistic object.



Global Illumination = **Ambient** Light + **Diffuse** Light + **Specular** Light + **Emissive** Light

Emissive Lighting

- ✓ Emissive lighting is described by a single term.

  Emissive Lighting = C_e

  // create material

  D3DMATERIAL9 mtrl;

  ZeroMemory(&mtrl, sizeof(mtrl));

  mtrl.Emissive.r = 0.0f;

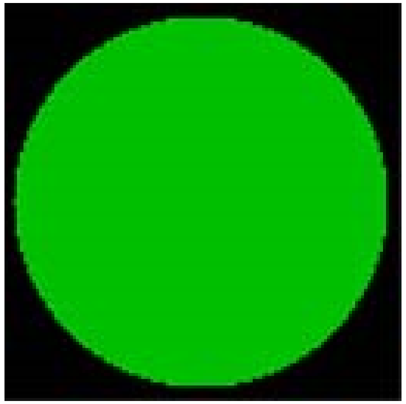
  mtrl.Emissive.g = 0.75f;

  mtrl.Emissive.b = 0.0f;

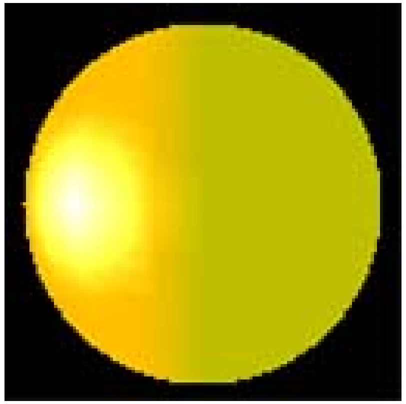
  mtrl.Emissive.a = 0.0f;

  m_pd3dDevice->SetMaterial(&mtrl);

  m_pd3dDevice->SetRenderState(**D3DRS_EMISSIVEMATERIALSOURCE**, D3DMCS_MATERIAL);



- ✓ This image shows how the emissive light blends with the other three types of lights, from the previous examples.



Global Illumination = **Ambient** Light + **Diffuse** Light + **Specular** Light + **Emissive** Light

Attenuation and Spotlight Factor

✓ The diffuse and specular lighting components of the global illumination equation contain terms that describe **light attenuation** and the spotlight cone.

$$\text{Atten} = 1/(\ att0_i + att1_i * d + att2_i * d^2)$$

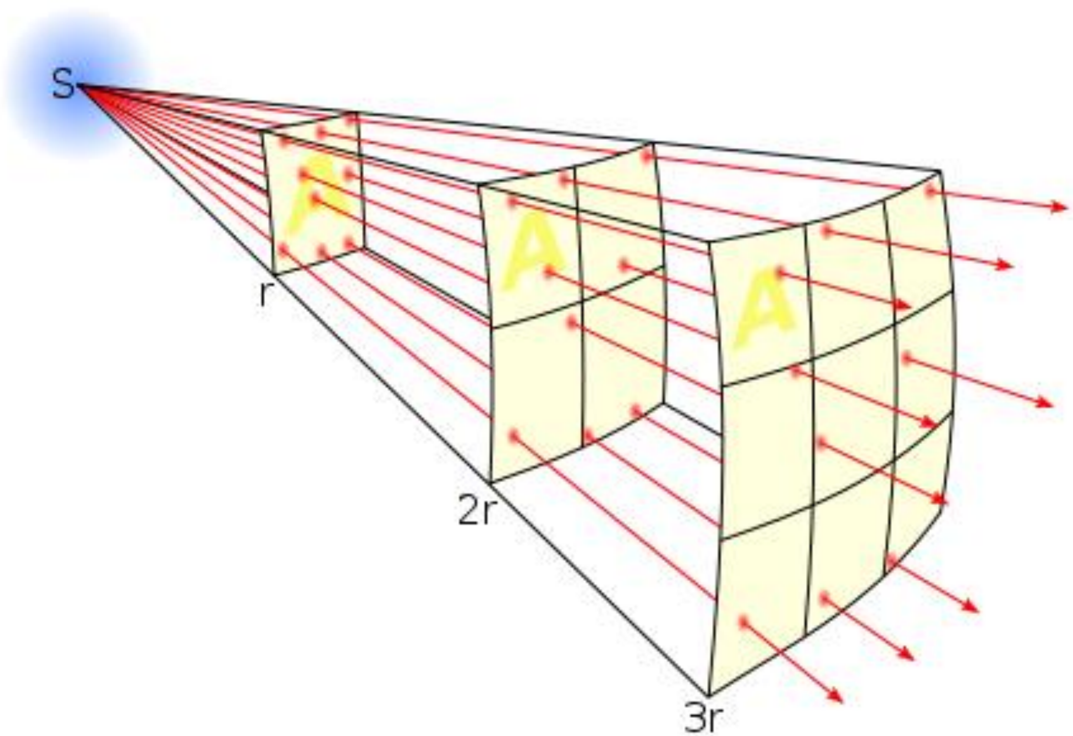
Parameter	Default value	Type	Description	Range
$att0_i$	0.0	FLOAT	Constant attenuation factor	0 to +infinity
$att1_i$	0.0	FLOAT	Linear attenuation factor	0 to +infinity
$att2_i$	0.0	FLOAT	Quadratic attenuation factor	0 to +infinity
d	N/A	FLOAT	Distance from vertex position to light position	N/A

- Atten = 1, if the light is a directional light.
- Atten = 0, if the distance between the light and the vertex exceeds the light's range.

Inverse-square law

✓ In physics, an **inverse-square law** is any physical law stating that a specified physical quantity or intensity is inversely proportional to the square of the distance from the source of that physical quantity.

$$\text{Intensity} \propto \frac{1}{\text{distance}^2}$$



Camera Space Transformations

- ✓ Vertices in the camera space are computed by transforming the object vertices with the world view matrix.
  - V = V * wvMatrix
- ✓ Vertex normals, in camera space, are computed by transforming the object normals with the inverse transpose of the world view matrix.
  - N = N * $(wvMatrix^{-1})^T$

(Proof)

- ✓ Since tangents and normals are perpendicular, the tangent vector **T** and the normal vector **N** associated with a vertex must satisfy the equation $\mathbf{N} \bullet \mathbf{T} = 0$.
- ✓ We must also require that this equation be satisfied by the transformed tangent vector $\mathbf{T}'$ and the transformed normal vector $\mathbf{N}'$.
- ✓ Given a transformation matrix **M**, we know that $\mathbf{T}' = \mathbf{MT}$.
- ✓ We would like to find the transformation matrix **G** with which the vector **N** should be transformed so that
  - $\mathbf{N}' \bullet \mathbf{T}' = (\mathbf{GN}) \bullet (\mathbf{MT}) = 0.$

- A little algebraic manipulation gives us

$$(\mathbf{GN}) \bullet (\mathbf{MT}) = (\mathbf{GN})^{T}(\mathbf{MT}) = \mathbf{N}^{T}\mathbf{G}^{T}\mathbf{MT}$$

- Since $\mathbf{N}^{T}\mathbf{T} = 0 (\mathbf{N} \bullet \mathbf{T} = 0)$, the equation $\mathbf{N}^{T}\mathbf{G}^{T}\mathbf{M}\mathbf{T} = 0$ is satisfied if $\mathbf{G}^{T}\mathbf{M} = \mathbf{I}$. We therefore conclude that

$$\mathbf{G} = (\mathbf{M}^{-1})^{T}.$$

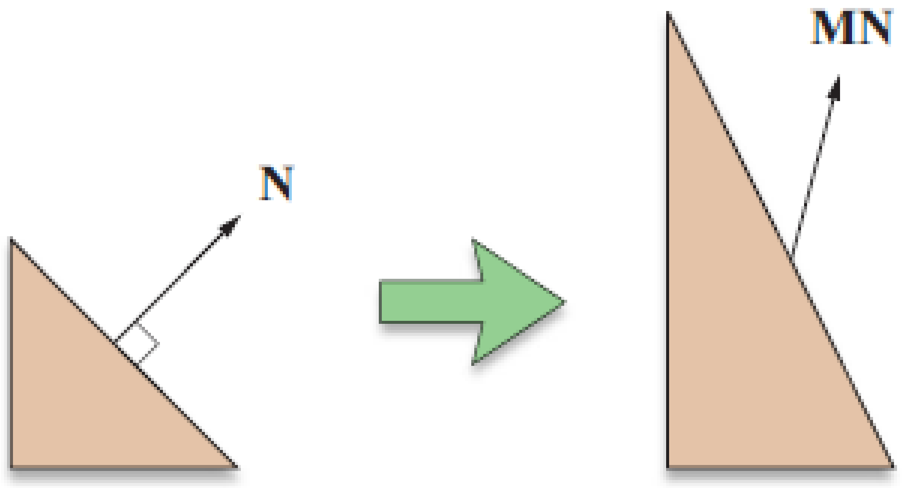
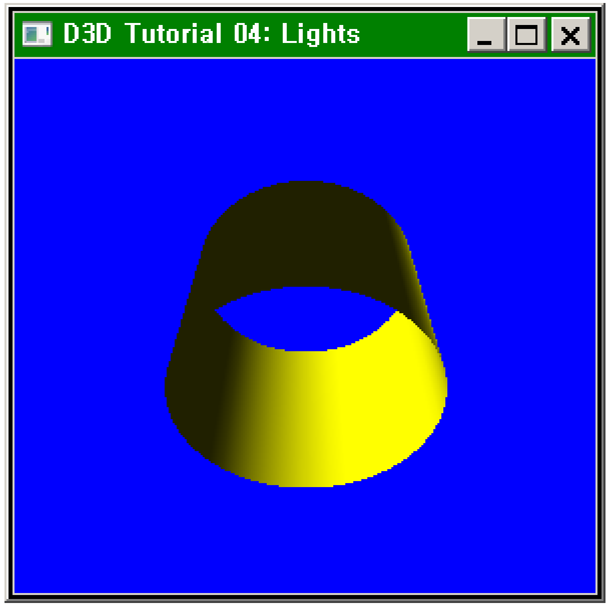


Figure 4.6. Transforming a normal vector **N** with a nonorthogonal matrix **M**.

Tutorial 04: Creating and Using Lights



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

- Step 1 - Initializing Scene Geometry
- Step 2 - Setting Up Material and Light

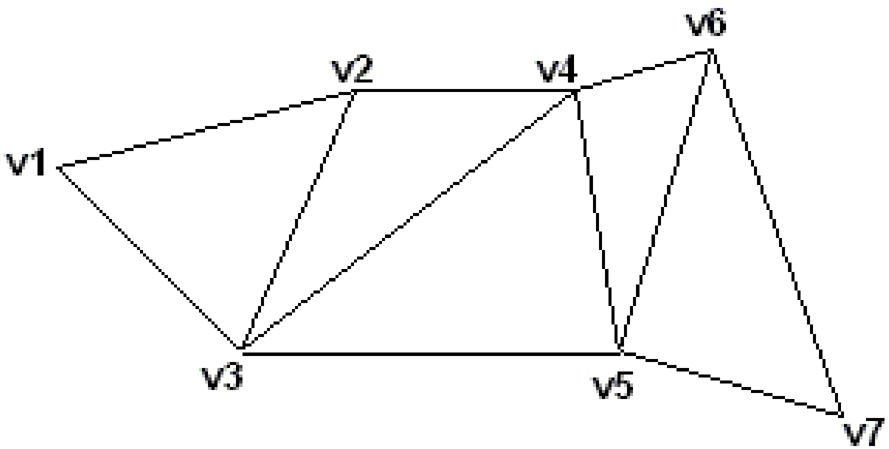
Step 1 - Initializing Scene Geometry

- ✓ One of the requirements of using lights is that each surface has a normal.

```
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)
```

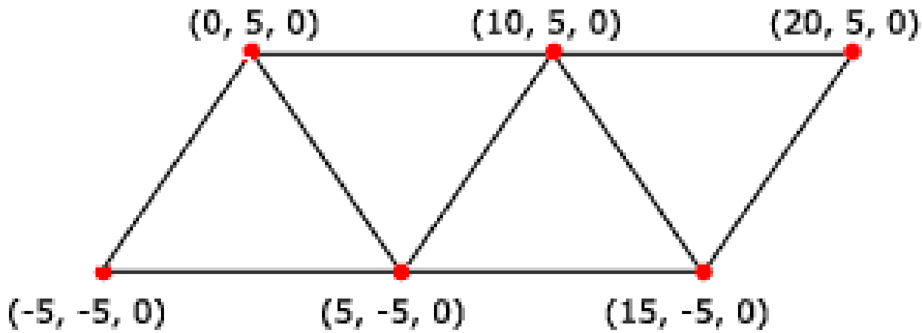
Triangle Strips

- ✓ A triangle strip is a series of connected triangles.
  - – Because the triangles are connected, the application does not need to repeatedly specify all three vertices for each triangle.



- ✓ The system uses vertices **v1, v2, and v3** to draw the first triangle; **v2, v4, and v3** to draw the second triangle.

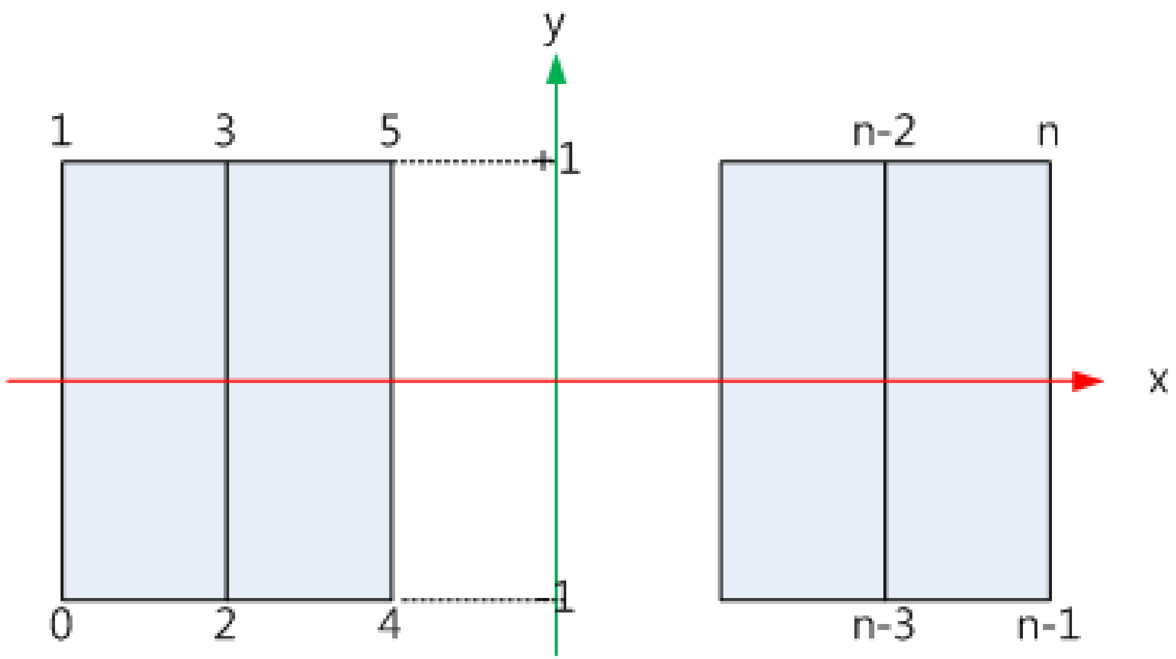
```
CUSTOMVERTEX Vertices[] =
{
    {-5.0, -5.0, 0.0},
    { 0.0,  5.0, 0.0},
    { 5.0, -5.0, 0.0},
    {10.0,  5.0, 0.0},
    {15.0, -5.0, 0.0},
    {20.0,  5.0, 0.0}
};
```

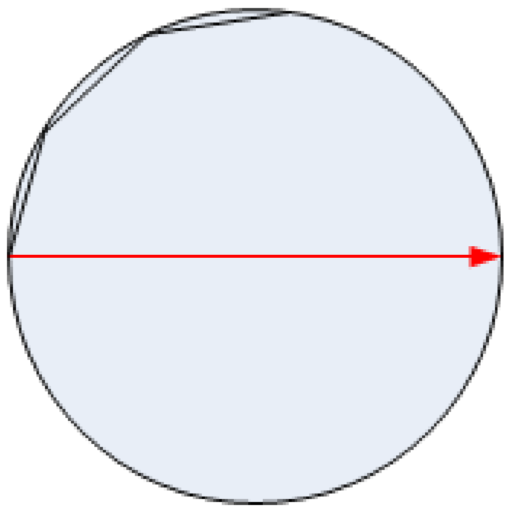


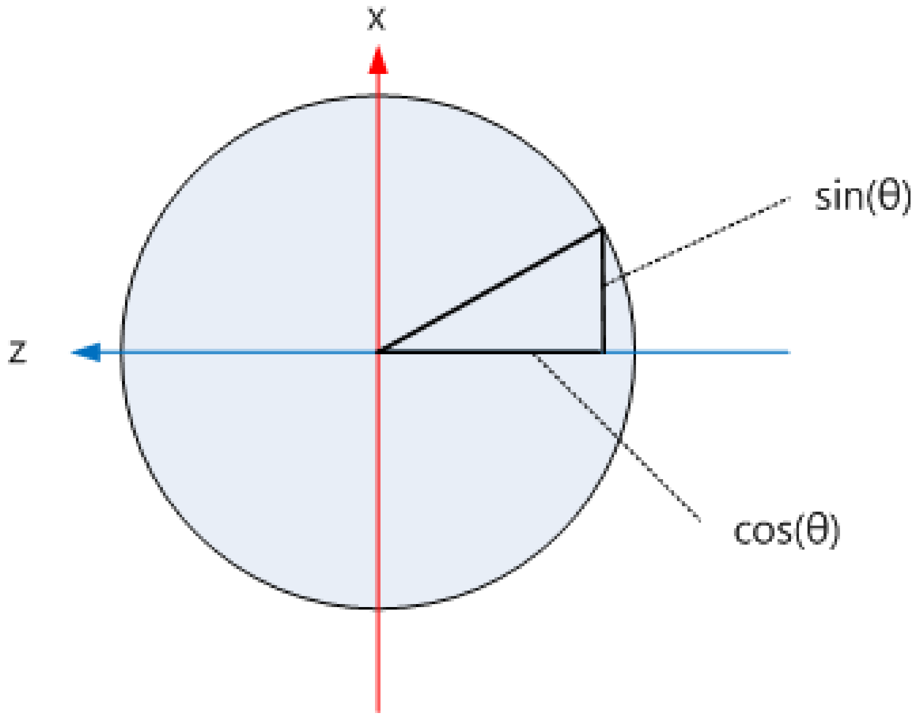
- ✓ The code example below shows how to use IDirect3DDevice9::DrawPrimitive to render this triangle strip.

```
d3dDevice->DrawPrimitive( D3DPT_TRIANGLESTRIP, 0, 4);
```

Cylinder: Continuous sectors



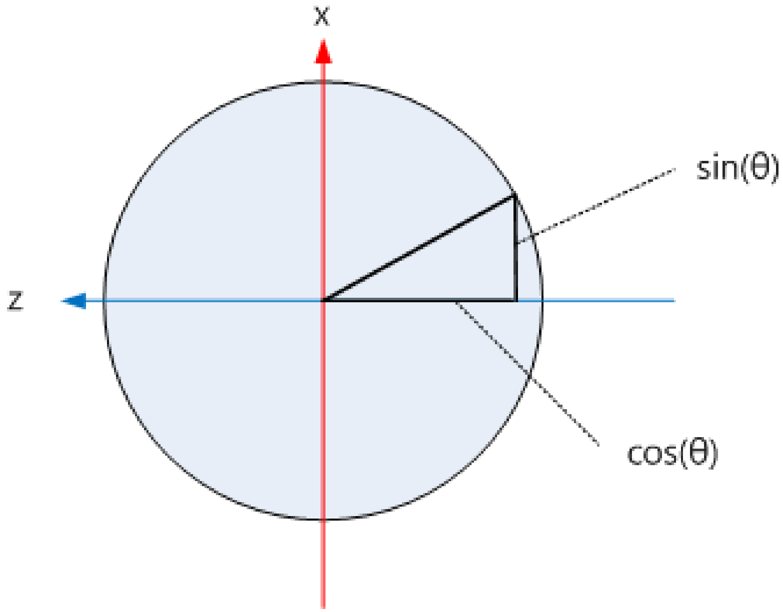




- ✓ Now that the correct vector format is defined, the Lights sample project calls InitGeometry, an application-defined function that creates a **cylinder**.

```
// 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.




```
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) );
}
```

Step 2 - Setting Up Material and Light

✓ 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 );
```

Creating a Light

- ✓ The sample code creates a directional light, which is a light that goes in one direction.

```
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 );
```

- ✓ Assigns the light to the Direct3D device, and enables a light.

```
light.Range = 1000.0f;
g_pd3dDevice->SetLight( 0, &light );
g_pd3dDevice->LightEnable( 0, TRUE );
g_pd3dDevice->SetRenderState( D3DRS_LIGHTING, TRUE );
```

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

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

References

- ✓ Mathematics.for.3D.Game.Programming.and.Computer.Graphics,.Lengyel,.3rd,.2011.pdf

MY **BRIGHT** FUTURE

동서대학교

DSU Dongseo University
동서대학교