Color, Lighting

305890
2007년 봄학기
4/6/2007
박경신

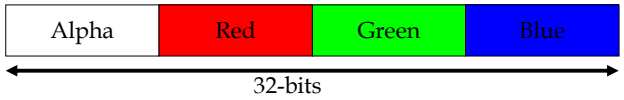
Overview

- Representing Color
- Vertex Colors
- Flat Shading, Gouraud Shading, Phong Shading
- Colored Triangle 예제
- Lighting

Representing Colors

- Color 표현 - RGB
- D3DColor: DWORD (32 bits)
  - 4개의 8bit color로 구성
  - Macro를 사용 – D3DCOLOR_ARGB, D3DCOLOR_XRGB
  - D3DCOLOR_ARGB: alpha, red, green, blue
  - D3DCOLOR_XRGB: red, green, blue (alpha는 0xff로 지정)
    - #define D3DCOLOR_XRGB(r, g, b) D3DCOLOR_ARGB(0xff, r, g, b)

```
D3DCOLOR brightRed = D3DCOLOR_ARGB(255, 255, 0, 0)
D3DCOLOR yellowGreen = D3DCOLOR_ARGB(128, 255, 0)
```



Representing Colors

- D3DCOLORVALUE: 4개의 float(0 ~ 1)

```
typedef struct _D3DCOLORVALUE {
    float r;    // red, 0~1
    float g;    // green, 0~1
    float b;    // blue, 0~1
    float a;    // alpha, 0~1
} D3DCOLORVALUE;
```

- D3DCOLOR: D3DCOLORVALUE + operators

Representing Colors

```
typedef struct D3DXCOLOR {
#ifdef __cplusplus
public:
    D3DXCOLOR() {}
    D3DXCOLOR(DWORD argb);
    D3DXCOLOR(CONST FLOAT *);
    D3DXCOLOR(CONST D3DXFLOAT16*);
    D3DXCOLOR(CONST D3DXCOLORVALUE&);
    D3DXCOLOR(FLOAT r, FLOAT g, FLOAT b, FLOAT a);

    operator DWORD() const;
    operator FLOAT* ();
    operator CONST FLOAT* () const;
    operator D3DCOLORVALUE* ();
    operator CONST D3DCOLORVALUE* () const;
    operator D3DCOLORVALUE& ();
    operator CONST D3DCOLORVALUE& () const;
```

Representing Colors

```
    D3DXCOLOR& operator += (CONST D3DXCOLOR&); //assignment
    D3DXCOLOR& operator -= (CONST D3DXCOLOR&);
    D3DXCOLOR& operator *= (FLOAT);
    D3DXCOLOR& operator /= (FLOAT);
    D3DXCOLOR operator + () const;                    // unary operator
    D3DXCOLOR operator - () const;
    D3DXCOLOR operator + (CONST D3DXCOLOR&) const; // binary operator
    D3DXCOLOR operator – (CONST D3DXCOLOR&) const;
    D3DXCOLOR operator * (FLOAT) const;
    D3DXCOLOR operator / (FLOAT) const;
    friend D3DXCOLOR operator * (FLOAT, CONST D3DXCOLOR&);
    BOOL operator == (CONST D3DXCOLOR&) const;
    BOOL operator != (CONST D3DXCOLOR&) const;
#endif // _cplusplus
    FLOAT r, g, b, a;
} D3DXCOLOR, *LPD3DXCOLOR;
```

Representing Colors

- d3dUtility.h에 전역 컬러 상수 추가

```
namespace d3d
{
    …
    const D3DXCOLOR WHITE (D3DXCOLOR_XRGB(255, 255, 255));
    const D3DXCOLOR BLACK (D3DXCOLOR_XRGB(0, 0, 0));
    const D3DXCOLOR RED (D3DXCOLOR_XRGB(255, 0, 0));
    const D3DXCOLOR GREEN (D3DXCOLOR_XRGB(0, 255, 0));
    const D3DXCOLOR BLUE (D3DXCOLOR_XRGB(0, 0, 255));
    const D3DXCOLOR YELLOW (D3DXCOLOR_XRGB(255, 255, 0));
    const D3DXCOLOR CYAN (D3DXCOLOR_XRGB(0, 255, 255));
    const D3DXCOLOR MAGENTA (D3DXCOLOR_XRGB(255, 0, 255));
}
```

Vertex Color

- Vertex Data Structure에 Color추가
  - Color를 DWORD로 지정 (Vertex Shader를 사용하지 않는 경우)

```
struct ColorVertex {
    float _x, _y, _z;          // XYZ
    D3DCOLOR _color;           // color
    static const DWORD FVF;
};
Const DWORD ColorVertex::FVF = D3DFVF_XYZ | D3DFVF_DFFUSE;
```

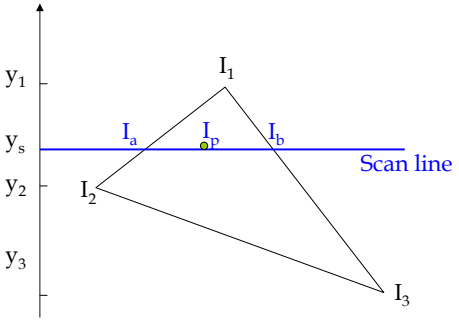
Shading

- 쉐이딩이란 물체의 표면의 밝기를 나타내는 것으로 래스터 과정 중에 이루어짐
- Face 내부의 각 pixel color값의 결정 방식은 SetRenderState로 지정
  - Flat shading: 첫번째 vertex의 color로 전체 face를 채움. 그 외의 vertex의 color값은 무시
  - Gouraud shading (smooth shading): face에서의 보간된 color값을 계산하여 채움

```
ColorVertex t[3];
t[0]._color = D3DCOLOR_XRGB(255, 0, 0);
t[1]._color = D3DCOLOR_XRGB(0, 255, 0);
t[2]._color = D3DCOLOR_XRGB(0, 0, 255);
Device->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_FLAT);
Device->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_GOURAUD);
```

Gouraud Shading

- Gouraud shading에서 보간된(interpolated) color값 계산
  - 시작점과 끝점의 밝기를 미리 계산해 두고, 그 사이의 밝기를 두 밝기의 가중평균으로 구한다.



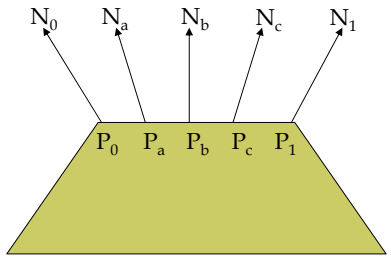
$$I_a = I_1 - (I_1 - I_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 - (I_1 - I_3)\frac{y_1 - y_s}{y_1 - y_3}$$

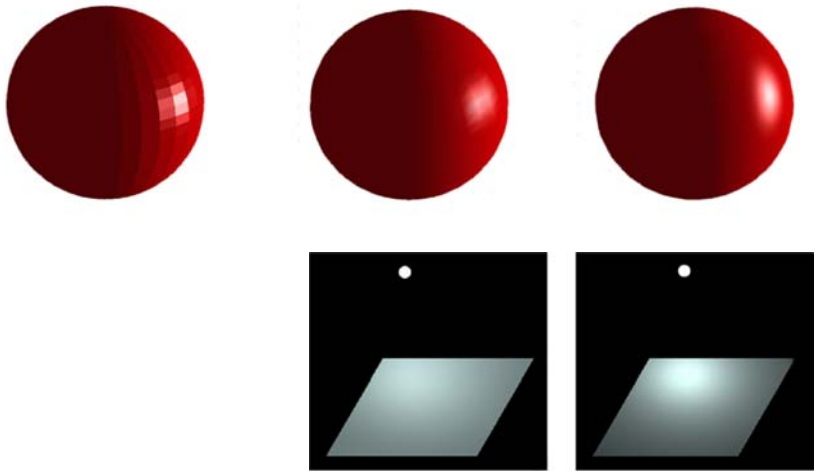
$$I_p = I_b - (I_b - I_a)\frac{x_b - x_p}{x_b - x_a}$$

Phong Shading

- Normal-vector interpolated shading이라고도 불림
- 각 점마다 normal vector를 계산
  - 물체의 표면의 밝기를 분산, 배경광원, 직접반사의 세 가지의 항목으로 계산한다.



Flat, Gouraud, and Phong Shading



Example: Colored Triangle

```
#include "d3dUtility.h"

IDirect3DDevice9* Device = 0;
const int Width = 640; const int Height = 480;
D3DXMATRIX WorldMatrix;
IDirect3DVertexBuffer9* Triangle = 0;

struct ColorVertex {
    ColorVertex() {}
    ColorVertex(float x, float y, float z, D3DCOLOR c) {
        _x = x; _y = y; _z = z; _color = c; }
    float _x, _y, _z;
    D3DCOLOR _color;
    static const DWORD FVF;
};
const DWORD ColorVertex::FVF = D3DFVF_XYZ | D3DFVF_DIFFUSE;
```

Example: Colored Triangle

```
bool Setup() {
    Device->CreateVertexBuffer(3 * sizeof(ColorVertex), D3DUSAGE_WRITEONLY,
    ColorVertex::FVF, D3DPOOL_MANAGED, &Triangle, 0);
    ColorVertex* v;
    Triangle->Lock(0, 0, (void**)&v, 0);
    v[0] = ColorVertex(-1.0f, 0.0f, 2.0f, D3DCOLOR_XRGB(255, 0, 0));
    v[1] = ColorVertex( 0.0f, 1.0f, 2.0f, D3DCOLOR_XRGB(0, 255, 0));
    v[2] = ColorVertex( 1.0f, 0.0f, 2.0f, D3DCOLOR_XRGB(0, 0, 255));
    Triangle->Unlock();
    D3DXMATRIX proj;
    D3DXMatrixPerspectiveFovLH(&proj, D3DX_PI*0.5f, (float)Width/(float)Height,
    1.0f, 1000.0f);
    Device->SetTransform(D3DTS_PROJECTION, &proj);
    Device->SetRenderState(D3DRS_LIGHTING, false); // light off
    return true;
}
void Cleanup() { d3d::Release<IDirect3DVertexBuffer9*>(Triangle); }
```

Example: Colored Triangle

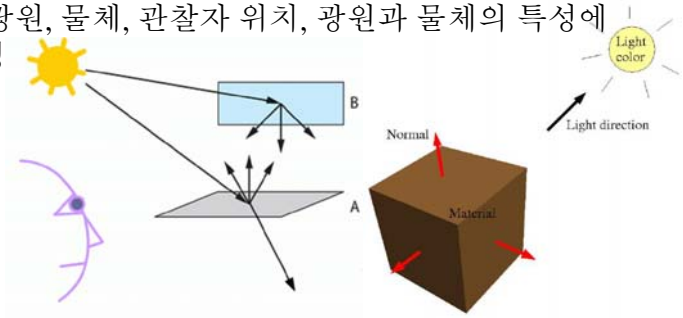
```
bool Display(float timeDelta) {
    if (Device) {
      Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0)
      Device->BeginScene();
      Device->SetFVF(ColorVertex::FVF);
      Device->SetStreamSource(0, Triangle, 0, sizeof(ColorVertex));
      // draw the triangle to the left with flat shading
      D3DXMatrixTranslation(&WorldMatrix, -1.25f, 0.0f, 0.0f);
      Device->SetTransform(D3DTS_WORLD, &WorldMatrix);
      Device->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_FLAT);
      Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);
      // draw the triangle to the right with gouraud shading
      D3DXMatrixTranslation(&WorldMatrix, 1.25f, 0.0f, 0.0f);
      Device->SetTransform(D3DTS_WORLD, &WorldMatrix);
      Device->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_GOURAUD);
      Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);
      Device->EndScene();
      Device->Present(0, 0, 0, 0);
    }
    return true;
}
```

Lighting

- Light Components
- Materials
- Vertex Normals
- Light Sources
- Lighting 예제

Lighting

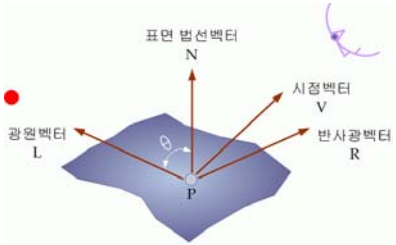
- 광원 (Lighting source)에서 출발
- 물체 표면에서
  - 흡수 (Absorption)
  - 반사 (Reflection)
  - 투과 (Transmission) 또는 굴절 (Refraction)
- 물체를 본다는 것은 우리 눈으로 입사하는 빛에 의함
- 물체색: 광원, 물체, 관찰자 위치, 광원과 물체의 특성에 의해 결정



Lighting Model

- 조명관련 벡터
  - 입사각: 광원벡터와 법선벡터가 이루는 각

$$N \cdot L = \|N\|\|L\|\cos\theta = (1)(1)\cos\theta = \cos\theta$$

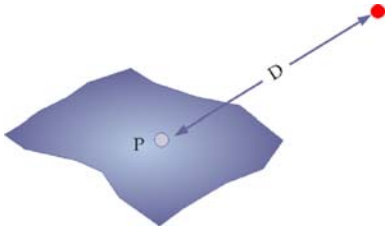


Ambient Reflection

- 광원에 직접 노출되지 않는 면에 밝기를 부여
- 모든 빛의 경로를 추적하기 어려움
  - 면마다 상수 크기의 밝기를 추가
  - 전역 조명모델 효과를 근사적으로 부여

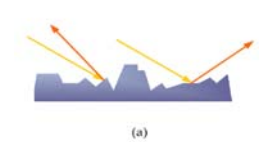
$$Ambient\ Reflection = K_a I_a$$

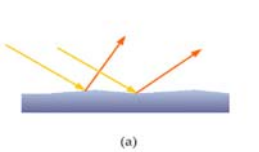
I_a: 광원의 주변광 세기
K_a: 주변광 계수

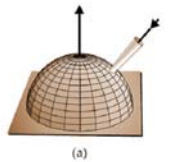


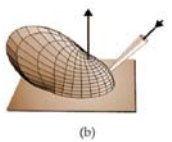
Diffuse Reflection

- 난반사에 해당
- 완벽 확산체 (Perfect Diffuser)와 방향성 확산체 (Directional Diffuser)
  - 방향성 확산체
    - 확산 방향에 시점이 있다면 물체가 더욱 밝게 보여야 함.
  - 완벽 확산체
    - 지역조명 모델의 그래픽 처리를 단순화하기 위해서 완벽 확산체를 가정



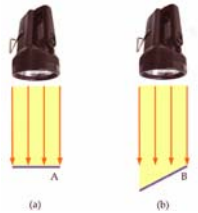






Diffuse Reflection

- 물체면이 서 있는 방향에 따라 다름
  - 램버트 법칙(Lambertian Law)
    - 입사각: 광원벡터, 법선벡터 사이각
    - 면의 밝기는 입사각의 코사인에 정비례



I_d: 광원의 확산광 세기
K_d: 확산광 계수

$$Diffuse\ Reflection \propto \cos\theta$$

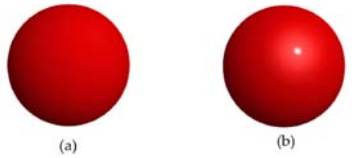
$$Diffuse\ Reflection = K_d I_d \cos\theta = K_d I_d (N \bullet L)$$

Light vector
Normal vector

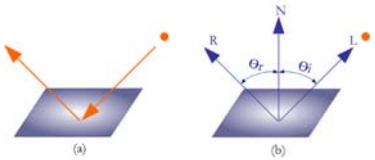
- 면이 서 있는 방향에 따라 차등적 밝기
  - 입체감 부여
  - cf. 주변광

Specular Reflection

- 반질반질한 표면에서 반사되는 빛
  - 정반사에 의함
  - 물체의 색이 아니라 광원의 색
  - cf. 주변광, 확산광: 광원의 색이 물체의 색과 상호작용
- 예: 확산, 확산+경면

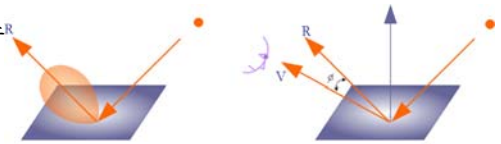


- 기본적으로 입사각과 반사각이 동일
  - 시점이 정확히 반대방향일 때 보임

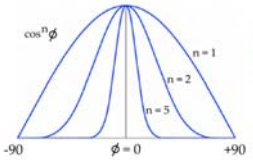


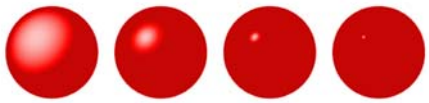
Specular Reflection

- 실제적으로는 Lobe 모습



- 퐁 반사모델 (Phong Illumination Model)
  - 광택계수 (Shininess Coefficient)를 통하여 시점 방향이 정반사 방향에서 벗어남에 따라 반사되는 빛의 세기가 약해지는 속도를 조절함. 결과적으로 생성되는 하이라이트 크기를 결정.

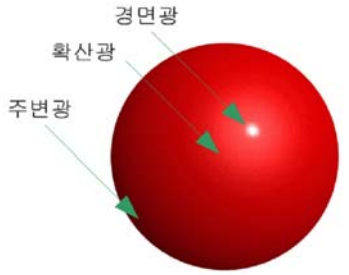




$$Specular\ Reflection = K_s I_s (\cos\phi)^n = K_s I_s (R \bullet V)^n$$

View vector
Reflection vector

Direct Illumination Model



$$I = K_a I_a + K_d I_d (N \bullet L) + K_s I_s (R \bullet V)^n$$

Ambient reflection
Diffuse reflection
Specular reflection

Light Attenuation

- 광원과 물체간의 거리에 따른 밝기 조절을 원할 경우

$$I = K_a I_a + f_{att}(d)\{K_d I_d (N \bullet L) + K_s I_s (N \bullet H)^n\}$$

$$f_{att}(d) = \frac{1}{d^2}$$

$$f_{att}(d) = \frac{1}{k_0 + k_1 d + k_2 d^2}$$

$$f_{att}(d) = \min\left(\frac{1}{k_0 + k_1 d + k_2 d^2}, 1\right)$$

Multiple Light Sources

- 광원이 여러 개 있을 경우

$$I = K_a I_a + \sum_{i=0}^{m-1} f_{att}(d)\{K_d I_d (N \bullet L) + K_s I_s (N \bullet H)^n\}$$

- 방출조명 (Emissive illumination) I_e=E
  - 특정한 물체들은 빛을 반사할 뿐만 아니라 빛을 뿜기도 하는데 이를 방출 조명이라 한다. 단순하게 방출 조명의 색을 더해주면 된다.

$$I = K_a I_a + \sum_{i=0}^{m-1} f_{att}(d)\{K_d I_d (N \bullet L) + K_s I_s (N \bullet H)^n\} + E$$

Lighting Component

- 빛의 구성 요소
  - Ambient light (환경광): 다른 일반 표면에서 반사되어 나오는 빛. 장면을 전반적으로 밝게 함.
  - Diffuse light (난반사광): 특정 방향으로 진행하다가 표면에 닿으면 모든 방향으로 동일하게 반사됨. 관찰자의 위치와 무관함. 가장 일반적인 형태임.
    - f (빛의 방향, 표면의 형태)
  - Specular light (정반사광): 특정 방향으로 진행하다가 표면에 닿으면 한 방향으로 강하게 반사됨. 반짝이는 표면을 모델링할 때 이용됨.
    - f (빛의 방향, 표면의 형태, 관찰자의 시점)
    - 계산량이 많으므로 default로 off임.
    - Device->SetRenderState(D3DRS_SPECULARENABLE, true);

Lighting Component

- 빛의 형은 빛의 컬러를 표현하는 D3DXCOLOR나 D3DCOLORVALUE 구조체로 나타낼 수 있다.

```
D3DXCOLOR redAmbient(1.0f, 0.0f, 0.0f, 1.0f);
D3DXCOLOR blueDiffuse(0.0f, 0.0f, 1.0f, 1.0f);
D3DXCOLOR whiteSpecular(1.0f, 1.0f, 1.0f, 1.0f);
```

- Light color를 표현할 때 alpha값은 이용되지 않음.

Materials

- Materials – D3DMATERIAL9
  - 물체의 재질에 따라서 색이 다르게 보임.
  - 예: 빨간 공은 빨간색만 반사하고 그 외의 색은 모두 흡수함.

```
typedef struct _D3DMATERIAL9 {
    D3DCOLORVALUE Diffuse, Ambient, Specular, Emissive;
    float Power;
} D3DMATERIAL9;
```

  - Diffuse/Ambient/Specular: 표면이 반사하는 난반사광 /환경광/ 정반사광
  - Emissive: 전반적인 표면의 컬러를 더하는데 사용됨. 물체 자체가 빛을 발하는 것처럼 좀 더 밝은 물체 효과를 만들어 냄.
  - Power: 정반사광의 sharpness를 지정하며, 높은 값일 수록 highlight가 강조됨.

Materials

- 예: 빨간 공의 재질을 표현

```
D3DMATERIAL9 red;
::ZeroMemory(&red, sizeof(red));
red.Diffuse = D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f);
red.Ambient = D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f);
red.Specular = D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f);
red.Emissive = D3DXCOLOR(0.0f, 0.0f, 0.0f, 1.0f); // no emission
red.Power = 5.0f;
```

Materials

- d3dUtility.h에 전역 재질 상수 추가

```
namespace d3d
{
    …
    D3DMATERIAL9 InitMtrl (D3DXCOLOR a, D3DXCOLOR d, D3DXCOLOR s, D3DXCOLOR
    e, float p);
    const D3DMATERIAL WHITE_MTRL = InitMtrl(WHITE, WHITE, WHITE, BLACK, 8.0f);
    const D3DMATERIAL RED_MTRL = InitMtrl(RED, RED, RED, BLACK, 8.0f);
    const D3DMATERIAL GREEN_MTRL = InitMtrl(GREEN, GREEN, GREEN, BLACK, 8.0f);
    const D3DMATERIAL BLUE_MTRL = InitMtrl(BLUE, BLUE, BLUE, BLACK, 8.0f);
    const D3DMATERIAL YELLOW_MTRL = InitMtrl(YELLOW, YELLOW, YELLOW, BLACK,
    8.0f);
}
```

Materials

- d3dUtility.cpp에 InitMtrl 재질 함수 추가

```
D3DMATERIAL9 d3d::InitMtrl (D3DXCOLOR a, D3DXCOLOR d, D3DXCOLOR s,
D3DXCOLOR e, float p)
{
        D3DMATERIAL9 mtrl;
        mtrl.Ambient = a;
        mtrl.Diffuse = d;
        mtrl.Specular = s;
        mtrl.Emissive = e;
        mtrl.Power = p;
        return mtrl;
}
```

Materials

- Material 속성 지정
  - Vertex 구조체에서 지정하지 않고, SetMaterial로 현재의 Material을 지정

```
IDirect3DDevice9::SetMaterial(CONST D3DMATERIAL9* pMaterial)
```

  - 예: 물체를 서로 다른 재질을 이용하여 랜더링

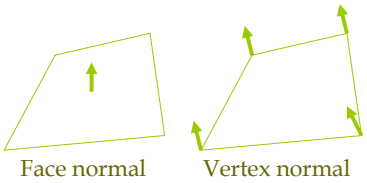
```
D3DMATERIAL9 blueMaterial, redMaterial;
// 재질 구조체를 구성

Device->SetMaterial(&blueMaterial);
drawSphere();          // blue sphere
Device->SetMaterial(&redMaterial);
drawSphere();          // red sphere
```

Vertex Normal

- Normal
  - 빛의 계산을 위해서 face normal이 아닌 vertex normal을 사용
- Normal 표현을 위한 Vertex struct 수정
  - Color를 제거하고 그 자리에 normal을 추가함

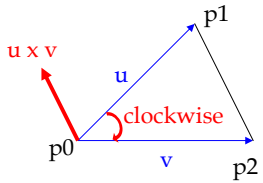
```
struct Vertex {
    float _x, _y, _z;          // XYZ
    float _nx, _ny, _nz;       // normal
    static const DWORD FVF;
};
Const DWORD Vertex::FVF = D3DFVF_XYZ | D3DFVF_NORMAL;
```



Vertex Normal

- Normal 계산
  - 3개의 vertex p0, p1, p2로 이루어지는 triangle의 normal 계산

```
void ComputeNormal(D3DXVECTOR3* p0, D3DXVECTOR3* p1,
    D3DXVECTOR3* p2, D3DXVECTOR* out) {
    D3DXVECTOR3 u = *p1 - *p0;
    D3DXVECTOR3 v = *p2 - *p0;
    D3DXVec3Cross(out, &u, &v);
    D3DXVec3Normalize(out, out);
}
```



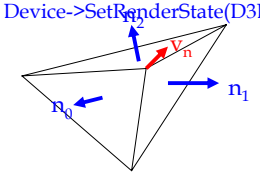
Vertex Normal

- Normal 계산
  - 더 부드러운 normal을 얻는 방법: vertex v를 공유하는 모든 face들의 normal의 평균을 그 vertex의 normal로 함.

$$v_n = \frac{1}{3}(n_0 + n_1 + n_2)$$

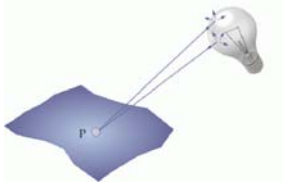
  - 주의: 변환 단계에서의 vertex normal의 왜곡 현상 방지를 위해, 변환 단계 이후에 Direct3D가 모든 normal들을 다시 정리하도록 지정하는 것이 안전하다.

```
Device->SetRenderState(D3DRS_NORMALIZENORMALS, true);
```

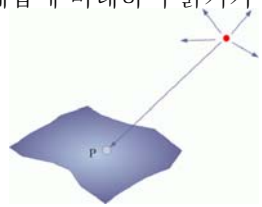


Light Source

- 면적광원 (Area Light Source)
  - 옴니라이트 (Omni Light), 빛이 모든 (Omni) 방향으로 방사형 (Radial Direction)으로 진행

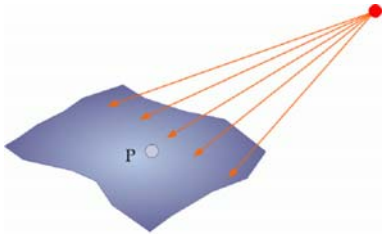


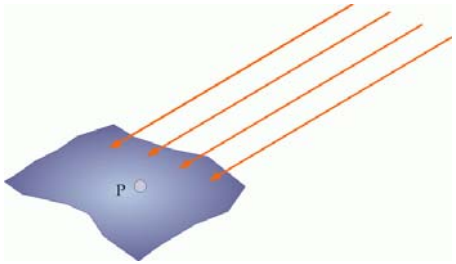
- 점광원 (Point Light Source)
  - 한 점을 중심으로 주변으로 퍼져나가는 빛
  - 빛이 반사될 표면과의 거리의 제곱에 비례하여 밝기가 감쇠



Light Source

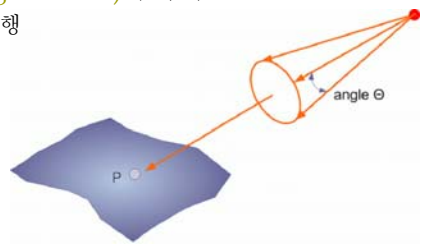
- 위치성 광원 (Positional Light Source)
  - 옴니라이트 (Omni Light), 빛이 모든 (Omni) 방향으로 방사형 (Radial Direction)으로 진행
  - 광원의 위치가 중시됨. 근거리 광원
- 방향성 광원 (Directional Light Source)
  - 빛이 물체면을 향하여 일정한 방향으로 진행
  - 거리에 상관없이 빛의 방향이 중시됨. 원거리 광원





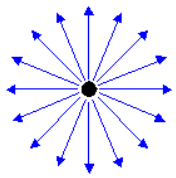
Light Source

- 점적광원 (Spot Light Source)
  - 점광원의 특수한 형태로 원뿔과 같이 일정한 범위로 빛을 발하는 광원
  - 광원의 위치, 빛을 발하는 중심 방향과 범위의 설정이 필요함
  - 위치성 광원 (Positional Light Source)과 유사
    - 빛이 방사형으로 진행
    - 광원이 유한 거리에 존재
  - 방향성 광원 (Directional Light Source)과 유사
    - 하나의 방향을 향해서만 진행



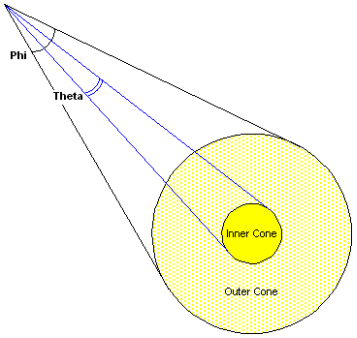
D3D Light Sources

- Direct3D에서 지원하는 light source
  - Point light
  - Directional light
  - Spot light



Point light

Directional light



Spot light

Light Sources

- Light source 구조체: D3DLIGHT9

```
typedef struct _D3DLIGHT9 {
    D3DLIGHTTYPE Type;            // light source type (point, spot, directional)
    D3DLIGHTVALUE Diffuse;        // diffuse, specular, ambient
    D3DLIGHTVALUE Specular;
    D3DLIGHTVALUE Ambient;
    D3DVECTOR Position; // light position. directional light인 경우 무시
    D3DVECTOR Direction; // light direction. spot light인 경우 무시
    float Range;    // 빛이 완전히 소멸할 때까지 진행할 수 있는 거리.
    float Falloff;    // spot light에서 안쪽 원뿔과 바깥쪽 원뿔 간의 빛의 세기 차이.
    float Attenuation0;// 빛의 세기가 거리에 따라 약해지는 정도.
    float Attenuation1;
    float Attenuation2;
    float Theta;      // spot light에서 안쪽 원뿔의 각도를 radian값으로 지정
    float Phi;        // spot light에서 바깥쪽 원뿔의 각도를 radian값으로 지정
} D3DLIGHT9;
```

Light Sources

- D3DLIGHT9 구조체
  - Type: 광원의 타입을 지정. D3DLIGHT_POINT, D3DLIGHT_SPOT, D3DLIGHT_DIRECTIONAL
  - Diffuse/Specular/Ambient: 광원이 발산하는 난반사/정반사/환경광
  - Position: 광원의 위치 (in world space). Directional light인 경우 무시
  - Direction: 광원의 방향 (in world space). Spot light인 경우 무시. Direction의 길이(length)는 0이 아니어야 함.
  - Range: 빛이 완전히 소멸할 때까지 진행할 수 있는 최대 거리. Directional light인 경우 무시
  - Falloff: spot light에서 안쪽의 원뿔과 바깥쪽의 원뿔 간 빛의 세기차이. 일반적으로 1.0으로 지정.
  - Attenuation0/1/2: 거리에 따라 빛의 세기가 약해지는 정도. Directional light에서는 무시. 0에서 ∞까지의 값. Directional light이 아닌 경우 세 값을 0으로 하면 안됨. Attenuation = $1/(a_0+a_1*D+a_2*D^2)$
  - Theta/Phi: spot light에서 안쪽/바깥쪽 원뿔의 radian 단위 $0 < \theta < \phi < \pi$

Light Sources

- d3dUtility.h에 간단한 광원의 초기화를 위한 함수 추가

```
namespace d3d
{
    …
    D3DLIGHT9 InitDirectionalLight(D3DXVECTOR3* direction,
                                   D3DXCOLOR* color);

    D3DLIGHT9 InitPointLight(D3DXVECTOR3* position,
                             D3DXCOLOR* color);

    D3DLIGHT9 InitSpotLight(D3DXVECTOR3* position,
                            D3DXVECTR3* direction,
                            D3DXCOLOR* color);
}
```

Light Sources

- d3dUtility.cpp에 광원초기화 함수 추가

```
D3DLIGHT9 d3d::InitDirectionalLight(D3DXVECTOR3* direction,
D3DXCOLOR* color)
{
    D3DLIGHT9 light;
    ::ZeroMemory(&light, sizeof(light));
    light.Type = D3DLIGHT_DIRECTIONAL;
    light.Ambient = *color * 0.6f;
    light.Diffuse = *color;
    light.Specular = *color * 0.6f;
    light.Direction = *direction;
    return light;
}
```

Light Sources

- d3dUtility.cpp에 광원초기화 함수 추가

```
D3DLIGHT9 d3d::InitPointLight(D3DXVECTOR3* position, D3DXCOLOR*
color)
{
        D3DLIGHT9 light;
        ::ZeroMemory(&light, sizeof(light));
        light.Type = D3DLIGHT_POINT;
        light.Ambient = *color * 0.6f;
        light.Diffuse = *color;
        light.Specular = *color * 0.6f;
        light.Position = *position;
        light.Range = 1000.0f;
        light.Falloff = 1.0f;
        light.Attenuation0 = 1.0f;
        light.Attenuation1 = light.Attenuation2 = 0.0f;
        return light;
}
```

Light Sources

- d3dUtility.cpp에 광원초기화 함수 추가

```
D3DLIGHT9 d3d::InitSpotLight(D3DXVECTOR3* position, D3DXVECTOR3*
direction, D3DXCOLOR* color)  {
        D3DLIGHT9 light;
        ::ZeroMemory(&light, sizeof(light));
        light.Type = D3DLIGHT_SPOT;
        light.Ambient = *color * 0.0f;
        light.Diffuse = *color;
        light.Specular = *color * 0.6f;
        light.Position = *position;
        light.Direction = *direction;
        light.Range = 1000.0f;
        light.Falloff = 1.0f;
        light.Attenuation0 = 1.0f;
        light.Attenuation1 = light.Attenuation2 = 0.0f;
        light.Theta = 0.4f; light.Phi = 0.9f;
        return light;
}
```

Light Sources

- 예: +x방향으로 평행하고 흰색인 directional light source를 생성

```
D3DXVECTOR3 dir(1.0f, 0.0f, 0.0f);
D3DXCOLOR c = d3d::WHITE;
D3DLIGHT9 dirLight = d3d::InitDirectionalLight(&dir, &c);
```

- D3DLIGHT9 instance 초기화 후, Direct3D에 등록해야 함.

```
Device->SetLight(0,      // 지정할 광원 리스트 내의 요소, 0~최대광원
                 &light);  // 설정하려는 D3DLIGHT9 구조체의 주소
```

- 등록 후, light을 on/off하고자 할 때

```
Device->LightEnable(0, true);     // light on
Device->LightEnable(0, false);    // light off
```

Example: Lit Pyramid

- 이 예제는 vertex normal, material, directional light 을 만들고 활성화하는 방법을 보여준다.
  - 조명을 활성화 한다.
  - 각 물체의 재질을 만들고 해당 물체를 렌더링하기 전에 재질을 지정한다.
  - 한 개 이상의 광원을 만들고 광원을 지정한 후 이를 활성화한다.
  - 정반사광과 같은 부가적인 조명 상태를 활성화한다.

Example: Lit Pyramid

```
#include "d3dUtility.h"

IDirect3DDevice9* Device = 0;
const int Width = 640;
const int Height = 480;
IDirect3DVertexBuffer9* Pyramid = 0;

struct Vertex {
    Vertex() {}
    Vertex(float x, float y, float z, float nx, float ny, float nz) {
        _x = x; _y = y; _z = z;
        _nx = nx; _ny = ny; _nz = nz;
    }
    float _x, _y, _z;
    float _nx, _ny, _nz;
    static const DWORD FVF;
};
const DWORD Vertex::FVF = D3DFVF_XYZ | D3DFVF_NORMAL;
```

Example: Lit Pyramid

```
bool Setup() {
    Device->SetRenderState(D3DRS_LIGHTING, true);
    Device->CreateVertexBuffer(12*sizeof(Vertex), D3DUSAGE_WRITEONLY,
                               Vertex::FVF, D3DPOOL_MANAGED, &Pyramid, 0);
    …
    Vertex* v;
    Pyramid->Lock(0, 0, (void**) &v, 0);
    v[0] = Vertex(-1.0f, 0.0f, -1.0f, 0.0f, 0.707f, -0.707f);        // 전면
    v[1] = Vertex( 0.0f, 1.0f,  0.0f, 0.0f, 0.707f, -0.707f);
    v[2] = Vertex( 1.0f, 0.0f, -1.0f, 0.0f, 0.707f, -0.707f);
    v[3] = Vertex(-1.0f, 0.0f,  1.0f, -0.707f, 0.707f, 0.0f);        // 왼쪽측면
    v[4] = Vertex( 0.0f, 1.0f,  0.0f, -0.707f, 0.707f , 0.0f);
    v[5] = Vertex( 1.0f, 0.0f, -1.0f, -0.707f, 0.707f , 0.0f);
    v[6] = Vertex( 1.0f, 0.0f, -1.0f, 0.707f, 0.707f, 0.0f);         // 오른쪽측면
    v[7] = Vertex( 0.0f, 1.0f,  0.0f, 0.707f, 0.707f , 0.0f);
    v[8] = Vertex( 1.0f, 0.0f, 1.0f, 0.707f, 0.707f , 0.0f);
    v[9] = Vertex( 1.0f, 0.0f, 1.0f, 0.0f, 0.707f, 0.707f);          // 후면
    v[10] = Vertex( 0.0f, 1.0f, 0.0f, 0.0f, 0.707f, 0.707f);
    v[11] = Vertex(-1.0f, 0.0f, 1.0f, 0.0f, 0.707f, 0.707f);
    Pyramid->Unlock();
```

Example: Lit Pyramid

```
    D3DMATERIAL9 mtrl;              // create and set the material
    mtrl.Ambient = d3d::WHITE;
    mtrl.Diffuse = d3d::WHITE;
    mtrl.Specular = d3d::WHITE;
    mtrl.Emissive = d3d::BLACK;
    mtrl.Power = 5.0f;
    Device->SetMaterial(&mtrl);

    D3DLIGHT9 dir;                          // setup a directional light
    ::ZeroMemory(&dir, sizeof(dir));
    dir.Type = D3DLIGHT_DIRECTIONAL;
    dir.Diffuse = d3d::WHITE;
    dir.Specular = d3d::WHITE * 0.3f;
    dir.Ambient = d3d::WHITE * 0.6f;
    dir.Direction = D3DXVECTOR3(1.0f, 0.0f, 0.0f); // +x방향으로 빛을 발산

    Device->SetLight(0, &dir);          // set light
    Device->LightEnable(0, true);               // light enable
    Device->SetRenderState(D3DRS_NORMALIZENORMALS, true);
    Device->SetRenderState(D3DRS_SPECULARENABLE, true);
```

Example: Lit Pyramid

```
    D3DXVECTOR3 pos(0.0f, 1.0f, -3.0f); // viewing transformation
    D3DXVECTOR3 target(0.0f, 0.0f, 0.0f);
    D3DXVECTOR3 up(0.0f, 1.0f, 0.0f);
    D3DXMATROX V;
    D3DXMatrixLookAtLH(&V, &pos, &targetm &up);
    Device->SetTransform(D3DTS_VIEW, &V);

    D3DXMATRIX proj;                        // projection
    D3DXMatrixPerspectiveFovLH(&proj, D3DX_PI *0.5f, (float)Width/(floatHeight, 1.0f,
    1000.0f);
    Device->SetTransform(D3DTS_PROJECTION, &proj);
    return true;
}

void Cleanup() {
    d3d::Release<IDirect3DVertexBuffer9*>(Pyramid);
}
```

Example: Lit Pyramid

```
void Display(float timeDelta) {
    if (Device) {
        D3DMATRIX yRot;
        static float y = 0.0f;
        D3DXMatrixRotationY(&yRot, y);
        r += timeDelta;
        if (y>=6.28f) y = 0.0f;
        Device->SetTransform(D3DTS_WORLD, &yRot);
        Device->Clear(0, 0, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER, 0x00000000, 1.0f, 0);
        Device->BeginScene();
        Device->SetStreamSource(0, Pyramid, 0, sizeof(Vertex));
        Device->SetFVF(Vertex::FVF);
        Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 4);
        Device->EndScene();
        Device->Present(0, 0, 0, 0);
    }
    return true;
}
```

부가적인 예제들

- Directional Light
  - D3DX objects & directional light 데모
- Point Light
  - D3DX objects & point light 데모
- Spot Light
  - D3DX objects & spot light 데모