3D Game Programming 05

afewhee@gmail.com

- 색상
- 조명
- 실습

- Direct3D 쉐이딩 모델
  - Pixel 사이의 색상과 밝고 어둠을 보충해서 채우는 보간(Interpolation-) 방법
  - Flat Shading 모델 → Diffuse로 구현 : 하나의 색상으로 한 면을 처리
  - Gouraud Shading(Henri Gouraud) 모델 → Diffuse로 구현: 색상을 선형적으로 보간. Direct3D의 Default 쉐이딩
  - Phong Shading(Bui Tuong Phong) 모델 → Specular로 구현: 카메라 위치, 정점이 법선 벡터, 광원의 방향으로 빛의 밝기 결정
  - Shader를 사용하면 Gouraud, Phong 쉐이딩 처리가 가능
- 색상
  - 정점의 색상을 Diffuse라 부름
  - Alpha-8bit, Red-8bit, Green-8bit, Blue-8bit → 총 32비트 구조
  - 바이트 순서는 가장 낮은 바이트 Blue → Green → Red → Alpha순
  - ※ OpenGL: r→g →b →a. GDI: r→g→b
  - 정점의 색상은 라이팅과 결합해서 Rasterize의 색상 형성

- 색상 응용
  - 구조체

```
struct VtxD
{
    float3   p;  // Position
    DWORD d;  // Diffuse color

    enum{ FVF=(D3DFVF_XYZ|D3D_DIFFUSE), };  // FVF
};
```

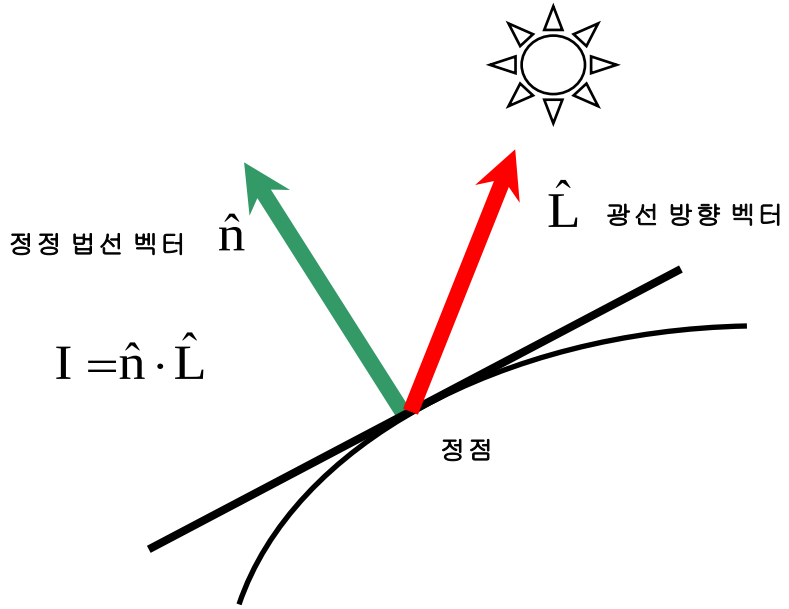
- 쉐이딩 적용
  - Flat Shading
    - pDevice->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_FLAT);
  - Gouraud Shading:
    - Direct3D의 Default 쉐이딩
    - pDevice->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_GOURAUD);

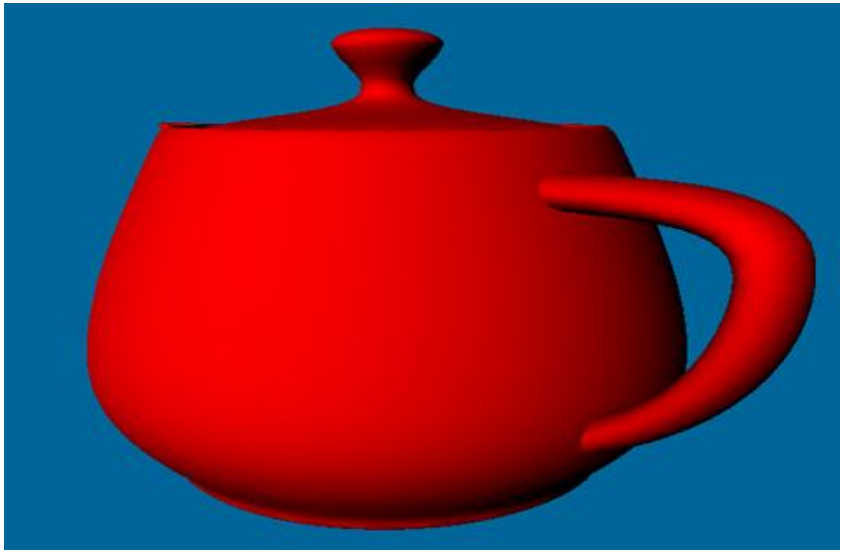
- 조명(Lighting)
  - Computer Graphics에서는 정확한 빛에 대한 물리법칙을 적용하지 않고 현실 세계의 빛에 대한 효과를 근사(Approximation)해서 처리

- 빛의 반사에 대한 모델
  - Lambert 확산
    - 빛의 반사에 대한 세기(Intensity)를 반사 면의 법선 벡터와 라이팅의 방향에 대한 내적으로 처리
  - 퐁의 반사
    - 퐁의 반사는 복잡한 전자기학 법칙을 단순화 시킨 모델로 시선 벡터와 반사되는 라이팅의 방향을 내적(Dot Product)한 값에 적당한 승 수(Power: 멱)를 적용해서 반사의 세기를 설정

- 분산 조명 효과(Diffuse Reflection: 난반사)
  - DirectX의 고정파이프 라인에서 램버트 확산
  - 반사의 세기 = 정점의 위치에서 바라보는 광원의 방향과 정점의 법선 벡터의 내적
  - 반사에 대한 색상의 밝기(Intensity) = Dot(N, L)
  - 여러 조명이 있을 경우 각각의 내적 값을 더함.

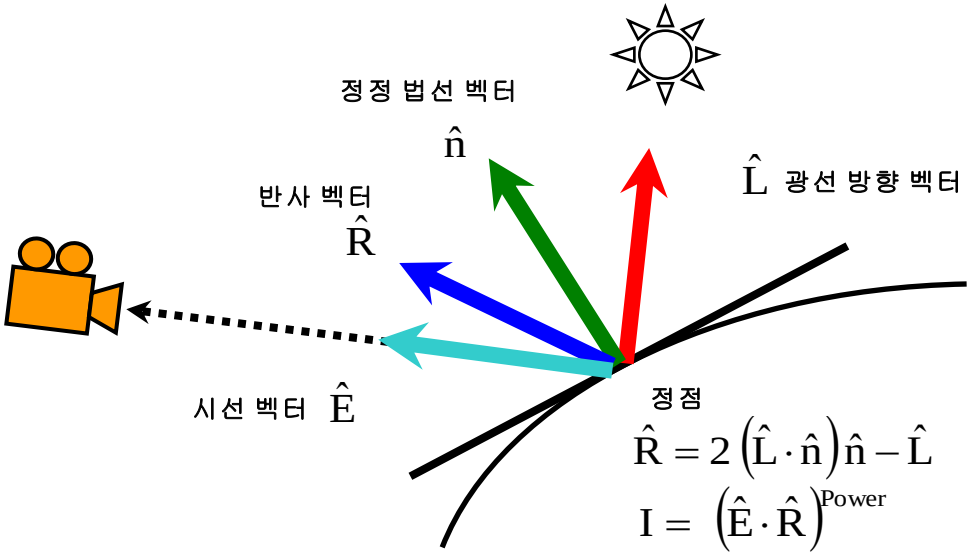
$$I = \sum_i \hat{n} \cdot \hat{L}_i \times (VertexDiffuse \otimes L_{Diffuse-i}Color)$$





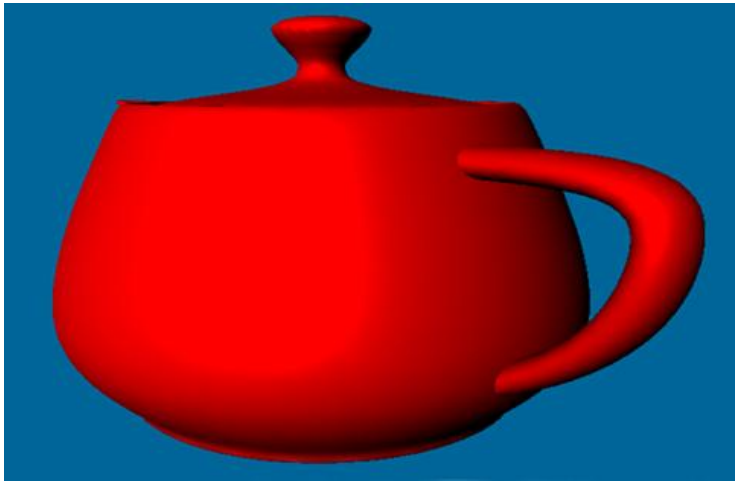
- Specular (정반사) 조명 효과
  - DirectX의 고정파이프 라인에서 퐁의 반사
  - 반사의 세기 = 정점의 위치에서 광원의 반사 방향과 정점에서 카메라 위치(시선 방향)에 대한 방향의 내적
  - 반사에 대한 색상의 밝기(Intensity) = Dot(E, L) : E 정점에서 시선 방향
  - 여러 조명이 있을 경우 각각의 내적 값을 더함.

$$I = \sum_i \left(\hat{E} \cdot \hat{R}_i\right)^{Power} \times (Specular \otimes L_{Specular-i} Color)$$



- Direct3D의 조명
  - 반사의 세기 = Ambient + Diffuse + Specular

$$I = \text{Ambient} \otimes \sum_i L_{\text{Ambient}-i} + \sum_i \hat{n} \cdot \hat{L}_i \times (\text{VertexDiffuse} \otimes L_{\text{Diffuse}-i}) + \sum_i \left(\hat{E} \cdot \hat{R}_i\right)^{\text{Power}} \times (\text{Specular} \otimes L_{\text{specular}-i})$$



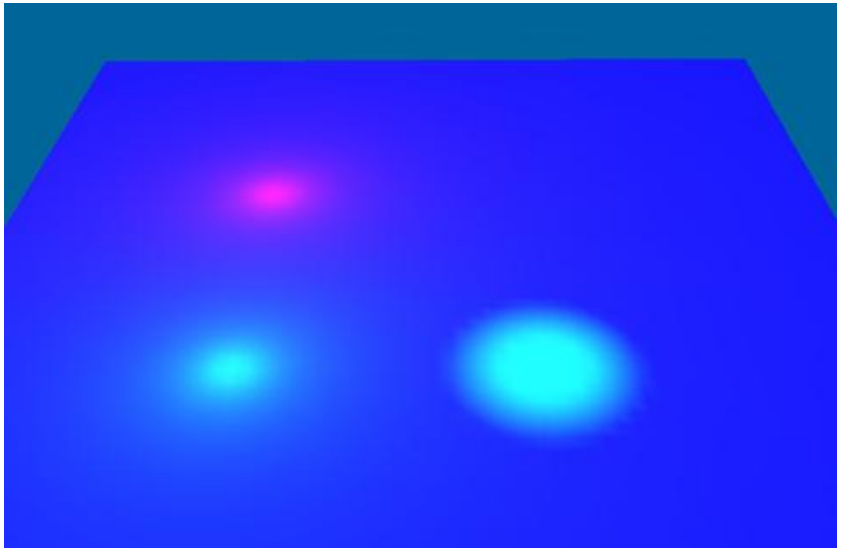
- 조명의 3요소
  - 현실 세계의 빛을 3요소로 만들어 흉내 냄
  - 정점이 많을수록 효과가 뛰어남
  - Ambient - 현실 세계의 안개처럼 전체적으로 밝음을 정함
  - Diffuse - 빛과 상호작용하는 물체에서 빛이 나는 효과
  - Specular - 유리면, 금속 면과 같이 특별한 영역이 더욱 밝게 빛나는 효과
    - 정반사광의 power 부분은 재질에서 설정

- 조명의 3가지 종류
  - 점 광원 (Point Light)
    - 백열 전구와 같이 하나의 점에서 사방으로 골고루 빛이 방출되는 광원
    - 광원의 위치가 중요
  - 평행 광원 (Directional Light)
    - 멀리 떨어진 태양처럼 평행하게 빛이 비추는 광원
    - 광원의 방향이 중요
  - 점적 광원 (Spot Light)
    - 무대 조명의 원뿔처럼 빛이 퍼지는 광원
    - 원뿔처럼 형성 되도록 Theta , Phi가 존재

- 재질
  - 조명과 연산. 반사의 색상을 결정
  - Ambient, Vertex Diffuse, Vertex Specular는 default의 경우 D3DMATERIAL9 구조체를 설정해야 함
    - pDevice->SetMaterial();

- Direct3D 조명 구조체

```
typedef struct _D3DLIGHT9 {
    D3DLIGHTTYPE Type;          // 조명의 종류. Point, Directional, Spot Light
    D3DCOLORVALUE Diffuse;      // 난반사광. 반사 Object의 Diffuse와 연산
    D3DCOLORVALUE Specular;     // 정반사광. 반사 Object의 Specular와 연산
    D3DCOLORVALUE Ambient;      // 주변광. 주어진 Ambient 값과 연산
    D3DVECTOR Position;         // 광원의 위치. 점광원, 점점 광원에 필요
    D3DVECTOR Direction;        // 빛의 방향. 평행광원에 필요
    float Range;                // 빛의 도달 거리
    float Falloff;              // 빛이 감쇠되는 정도
    float Attenuation0;
    float Attenuation1;
    float Attenuation2;
    float Theta;
    float Phi;
} D3DLIGHT9;
```



- Direct3D 재질 구조체

```
typedef struct _D3DMATERIAL9 {
    D3DCOLORVALUE Diffuse;      // 조명의 Diffuse와 연산
    D3DCOLORVALUE Ambient;      // 주어진 Ambient와 연산
    D3DCOLORVALUE Specular;     // 조명의 Specular와 연산
    D3DCOLORVALUE Emissive;     // 자체 방출 색상
    float Power;                // Specular의 Power
} D3DMATERIAL9;
```

- 프로그래밍 방법
  - 라이팅 구조체 설정
    - D3DLIGHT9
  - 라이팅 설정
    - pDevice->SetLight(nIndex, &Light)
    - 총 8개까지 라이팅 설정 가능
  - 라이팅 활성화
    - pDevice->LightEnable(nIndex, TRUE);
  - 렌더링 가상 머신의 라이팅 적용
    - pDevice->SetRenderState(D3DRS_LIGHTING, TRUE);
  - 재질 설정
    - D3DMATERIAL9
    - pDevice->SetMaterial(&d3dmaterial);

- 주의사항
  - 라이팅을 활성화 시켜놓고 적용이 안되면 검은 색으로 표현됨
    - 디바이스의 Clear 색상을 검정색으로 하면 라이팅이 적용 또는 문제를 파악하기 어려움
  - 라이팅이 적용되려면 정점에 법선 벡터가 꼭 필요
    - FVF = ... D3DFVF_NORMAL...
  - 법선 벡터가 정규화 되지 않으면 정규화 과정을 거치도록 디바이스에 설정
    - pDevice->SetRenderState(D3DRS_NORMALIZENORMALS, TRUE);
    - 보통 안하는 것이 좋음
  - 빛과 반응하는 재질 설정도 반드시 필요
    - pDevice->SetMaterial();
  - 전체 화면이 어둡지 않도록 최소한의 밝기가 유지되도록 Ambient 값을 설정
    - pDevice->SetRenderState(D3DRS_AMBIENT, 0x000F0F0F);

- 8x8, 16x16, 32x32, 64x64 격자로 구성된 Block을 정점들을 UP를 이용해서 Index 방식으로 화면에 출력하시오.
  - Block의 Cell의 Width = 32;
  - 높이 설정은 임의로 합니다.

- 위의 32x32 Block을 동적으로 4개를 생성해서 이들을 연속적인 지형으로 출력해 보시오.