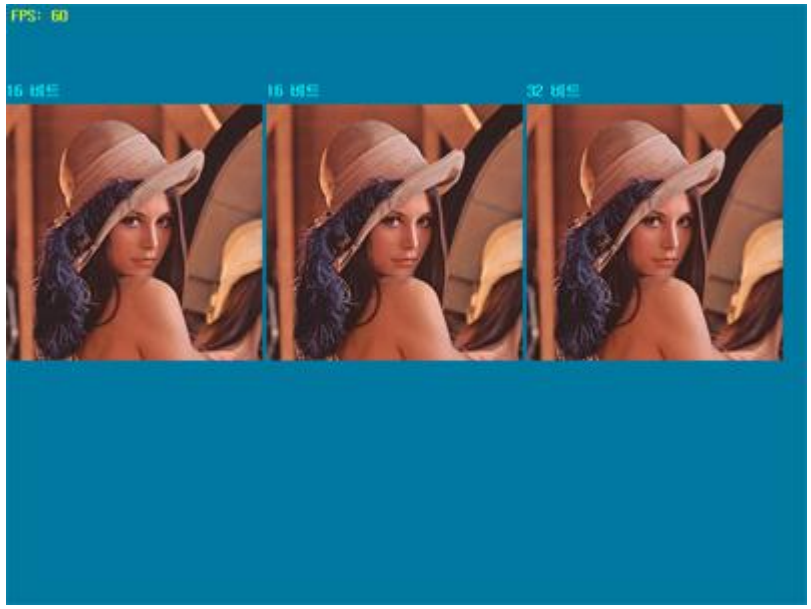
# 2D Game Programming 05

afewhee@gmail.com

# 1. Pixel Base Processing – Color Buffer

- 색상 버퍼(Color Buffer)
  - Direct3D에서 Pixel(색)으로 구성된 버퍼
  - Direct Draw에서는 보통 서피스(Surface)로 부름
  - 32Bit = Alpha(8) + Red(8) + Green(8) + Blue(8)
  - 가장 낮은 값: Blue
    - ←→ 윈도우 GDI는 Red가 가장 낮은 값
  - Texture, Back Buffer에 Color Buffer 존재




- 텍스처에서 사용
  - D3DSURFACE_DESC, D3DLOCKED_RECT 구조체 이용

Ex)

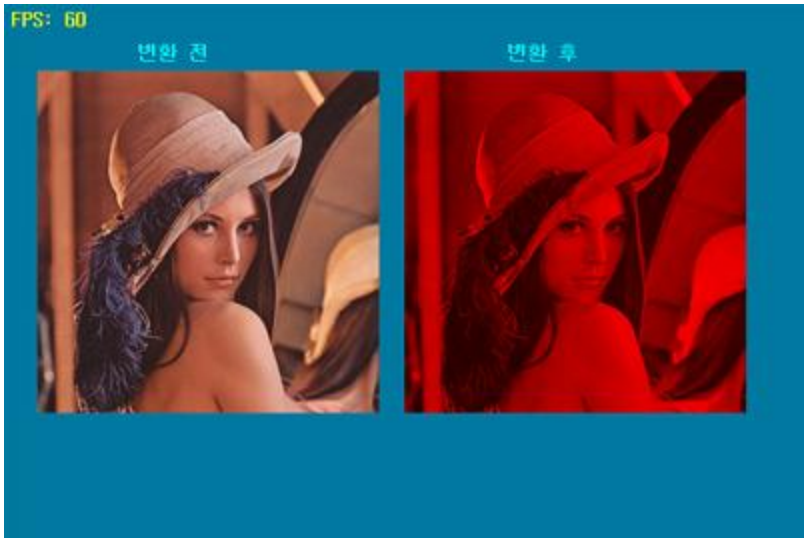
```
LPDIRECT3DTEXTURE9    pTx;
D3DSURFACE_DESC       dsc;
D3DLOCKED_RECT        rc;
...
pTx->GetLevelDesc(0, &dsc);              // 텍스처 정보 얻기
pTx->LockRect(0, &rc, NULL, 0);          // 텍스처 픽셀 정보 얻기
INT  Pitch = rc.Pitch;
nByte = Pitch/dsc.Width;                 // 1픽셀당 바이트 계산
...
pTx->UnlockRect(0);
```

- D3DXCOLOR 구조체
  - DirectX에서 지원하는 색상을 다루기 위해서 지원하는 구조체
  - 쉐이더에서도 별 다른 수정없이 사용할 수 있어서 가장 많이 애용
  - 32Bit 색상은 DWORD형을 주로 사용하는데 이에 대한 캐스팅 연산자 지원

- Color Shift
  - 특정한 색상만 출력
  - 텍스처에서 사용

Ex) Red Shift

```
void McColor_ColorRed(DWORD* pOut
                      , DWORD* pIn
                      , INT iImgW, INT iImgH)
{
    ...
    D3DXCOLOR xclr = pIn[nIdx];
    xclr.r *= 1.0f;
    xclr.g *= 0.0f;
    xclr.b *= 0.0f;
    pOut[nIdx] = xclr;
    ...
}
```



# 3. Pixel Base Processing - Mono Color

- 간단한 단색 공식
  - 최종 색상 = (붉은색 + 초록색 + 파랑색)
- 경험적인 단색 공식
  - 최종 색상 = 붉은색 * 0.299 + 초록색 * 0.587 + 파랑색 * 0.1

Ex)

```
void McColor_TransColorMono(DWORD* pOut
                          , DWORD* pIn
                          , INT iImgW
                          , INT iImgH)
    ...
    int nIdx = y*iImgW + x;
    D3DXCOLOR c = pIn[nIdx];

    //I = 0.30R + 0.59G + 0.11B
    float I = c.r * 0.3f + c.g * 0.59f + c.b * 0.11f;
    pOut[nIdx] = D3DXCOLOR(I, I, I, c.a);
    ...
}
```

# 4. Pixel Base Processing - Inverse




- 영상 필름과 비슷한 효과
  - 쉐이더, OpenGL에서는 색상의 크기를 [0,1] 값을 사용

- 색상 반전
  - 반전된 붉은색 = 1.0f – 붉은색
  - 반전된 초록색 = 1.0f – 초록색
  - 반전된 파랑색 = 1.0f – 파랑색

Ex)

```
void McColor_TransColorInverse(DWORD* pOut
                               , DWORD* pIn
                               , INT iImgW
                               , INT iImgH)
{
    ...
    D3DXCOLOR xclr = pIn[nIdx];

    xclr.r = 1- xclr.r;
    xclr.g = 1- xclr.g;
    xclr.b = 1- xclr.b;
    pOut[nIdx] = xclr;
    ...
}
```

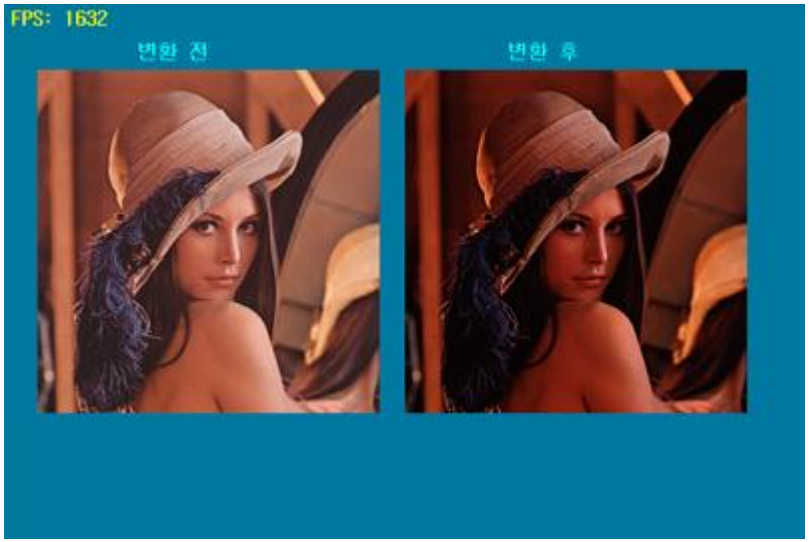
- 색상의 강도(Intensity)를 지수함수를 적용
  - 색상 강도(intensity) = x (1/r) , x=[0, 1.0]
  - 지수 함수 ➔ Pow()

Ex)

```
void McColor_TransColorGamma(DWORD* pOut
                             , DWORD* pIn
                             , INT iImgW
                             , INT iImgH
                             , D3DXCOLOR xGamma)
{
...
    D3DXCOLOR xClr = pIn[nIdx];

    xClr.r = powf(xClr.r, 1.0f/xGamma.r);
    xClr.g = powf(xClr.g, 1.0f/xGamma.g);
    xClr.b = powf(xClr.b, 1.0f/xGamma.b);

    pOut[nIdx] = xClr;
...
}
```



- 비트 플래너 슬라이싱
  - 색상을 비트 단위로 처리 시간을 빠르게 하기 위해 영상이 어느 정도 보존 될 때까지 레벨을 두어 비트 연산으로 해상도를 낮추는 작업
  - 보통 레벨은 2의 승수로 정함

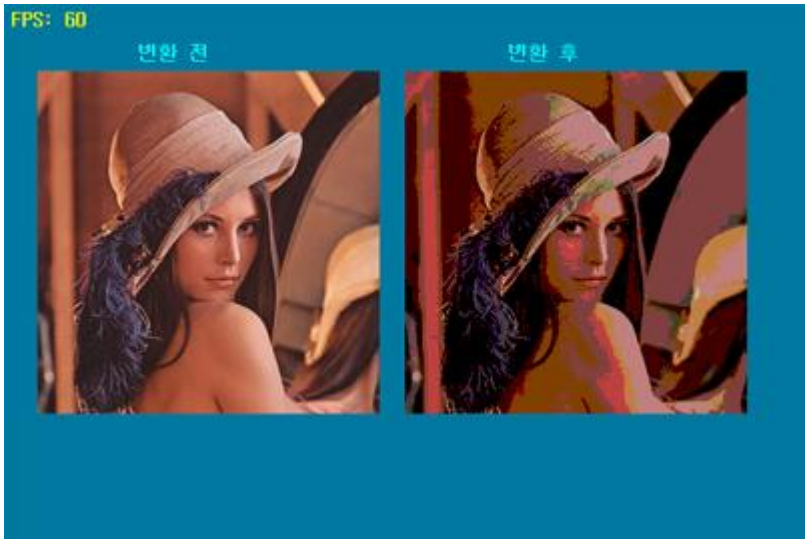
Ex)

```
void McColor_TransColorBitPlanner(DWORD* pOut
                        , DWORD* pIn
                        , INT iImgW
                        , INT iImgH
                        , INT nBit)
{
...
    D3DXCOLOR xclr = pIn[nIdx];
    DWORD r = DWORD(xclr.r * 255);
    DWORD g = DWORD(xclr.g * 255);
    DWORD b = DWORD(xclr.b * 255);

    r >>= (8-nBit);
    g >>= (8-nBit);
    b >>= (8-nBit);

    r < < = (8-nBit);
    g < < = (8-nBit);
    b < < = (8-nBit);

    pOut[nIdx] = D3DXCOLOR(r/255.f, g/255.f, b/255.f, 1.f);
...
}
```



- Adjacency Pixel Processing
  - 마스킹(Masking): 연산을 빠르게 수행하기 위해서 인접 픽셀에 적용하기 위한 비중(Weight) 값
  - 종류: Embossing, Blurring, Sharpening, Edge
  - 수학 함수 or 미분 방정식을 근사(Approximation)한 값

- 양각 효과(Embossing)

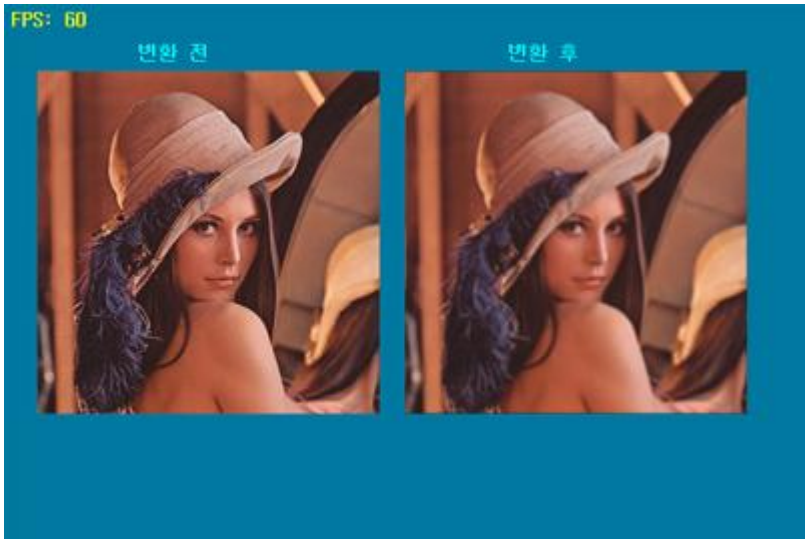
```
FLOAT       g_MaskSouthEast[][3] =
{
    {  2,  1,  0},
    {  1,  0, -1},
    {  0, -1, -2},
};
```

- Blurring
  - 일종의 Broadening ←→Sharpening
  - 적분형태
  - 3X3, 5X5, 7X7 Masking이 존재

Ex)

```
FLOAT g_MaskBlurRect3X3[][3] =
{
    {  1/9.f,  1/9.f,  1/9.f},
    {  1/9.f,  1/9.f,  1/9.f},
    {  1/9.f,  1/9.f,  1/9.f},
};

FLOAT g_MaskBlurCircular5X5[][5] =
{
    {    0.f,  1/21.f,  1/21.f,  1/21.f,    0.f},
    { 1/21.f,  1/21.f,  1/21.f,  1/21.f, 1/21.f},
    { 1/21.f,  1/21.f,  1/21.f,  1/21.f, 1/21.f},
    { 1/21.f,  1/21.f,  1/21.f,  1/21.f, 1/21.f},
    {    0.f,  1/21.f,  1/21.f,  1/21.f,    0.f},
};
```



- Sharpening
  - ←→Sharpening
  - 미분형태

Ex)

```
FLOAT g_MaskSharpening1[][3] =
{
    {  1, -2,  1},
    { -2,  5, -2},
    {  1, -2,  1},
};

FLOAT g_MaskSharpening2[][3] =
{
    {  0, -1,  0},
    { -1,  5, -1},
    {  0, -1,  0},
};
```

- 3개 이상의 캐릭터와 이에 대한 그림자를 표현하시오.