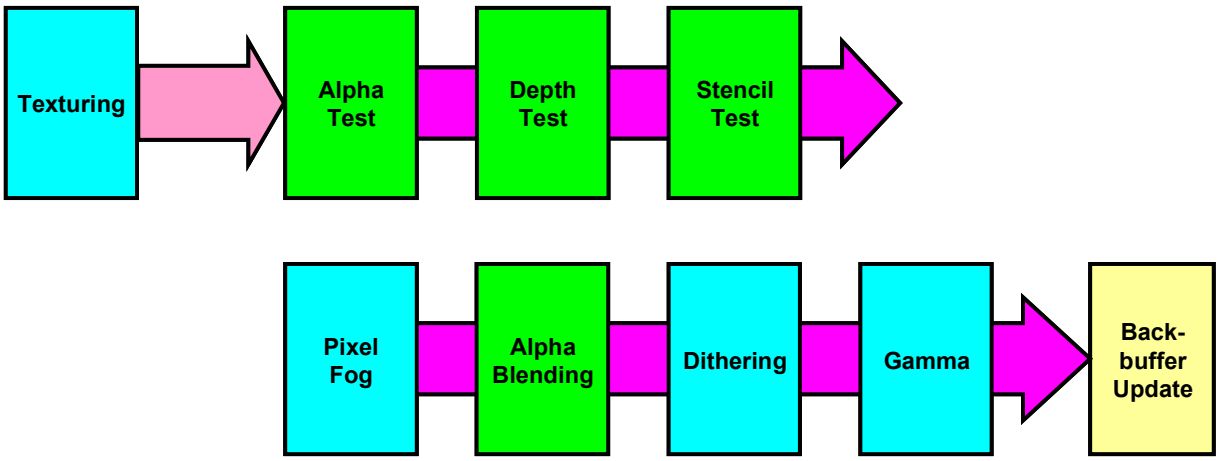
3D Game Programming Basic with Direct3D

8. Alpha Blending

기본 도형 처리가 끝낸 후 텍스처 처리과정을 거쳐 만들어진 픽셀들은 후면 버퍼를 갱신하기 위해 다음 그림과 같이 먼저 테스트를 합니다. 테스트를 통과한 픽셀들은 안개 색상, 알파 블렌딩, 디더링, 감마 교정으로 픽셀을 보정한 다음 비로소 후면 버퍼의 픽셀을 바꾸어 줍니다.



<텍스처 처리 후 그래픽 파이프라인의 처리과정>

알파 테스트(Alpha Test), 깊이 테스트(Depth Test), 스텐실(Stencil Test)가 먼저 오는 이유는 Fog, Alpha Blending, Dithering, Gamma 작업 들도 픽셀을 변경하는 작업이므로 이 작업들을 피하면 렌더링 속도가 증가되기 때문입니다.
간단하게 각 과정을 설명하면 알파 테스트는 픽셀의 알파 값을 주어진 테스트 값과 비교하는 과정입니다. 깊이 테스트는 후면 버퍼의 깊이 버퍼와 비교하는 테스트입니다. 스텐실 테스트는 후면 버퍼의 스텐실 값과 마스킹을 이용하는 테스트입니다.
픽셀 포그는 픽셀에 안개 색상을 적용하는 과정입니다. 알파 블렌딩은 후면버퍼에 저장된 픽셀과 연산을 통해서 새로운 색상을 만들어 내는 단계입니다. 디더링은 주변 색상들을 혼합해서 좀 더 일관성 있는 색상을 만듭니다. 감마 단계는 의 픽셀 색상에 감마 교정을 진행합니다.

순서를 바꿔서 이 중에서 먼저 알파 블렌딩을 살펴보고 그 다음으로 테스트와 나머지 과정에 대해서 알아보겠습니다.

8.1 Alpha Blending

앞서 알파 블렌딩을 간단히 설명했는데 블렌딩은 이전에 다중 텍스처 처리 과정에서도 있었습니다. 다중 텍스처 처리 과정의 블렌딩은 파이프라인에 있는 픽셀인 반면에 알파 블렌딩은 하나는 파이프라인에 있는 픽셀이고 다른 하나는 후면 버퍼 같은 곳에 저장되어 있는 픽셀입니다.

알파는 R, G, B 픽셀의 불투명 정도를 나타내는 픽셀의 무게(Weight) 값입니다. 알파 값이 1.0에 가까우면 불투명이고 0에 가까우면 완전 투명이 됩니다.
알파 블렌딩은 파이프라인의 새로운 픽셀에 가중치를 곱하고, 이미 저장 되어 있는 색상에도 이에 맞는 가중치를 곱한 다음 가산, 감산, 역 감산, 최대 값, 최소 값으로 새로운 픽셀의 색상을 정합니다. 이것을 간단한 수식으로 표현하면 다음과 같습니다.

$$\mathrm{Pixel}_{out} = \mathrm{Pixel}_{src} \otimes \mathrm{Blend\ Factor}_{src} \ \oplus \ \mathrm{Pixel}_{dst} \otimes \mathrm{Blend\ Factor}_{dst}$$

사용된 기호 설명은 다음과 같습니다.

Src(Source): 앞으로 쓰게 될 파이프라인에 있는 픽셀.
Dst(Destination): 후면 버퍼 같은 곳에 이미 저장되어 있는 픽셀이며 갱신의 대상인 픽셀
⊗: R, G, B, A에 대한 각각의 성분 단위 곱셈.
⊕: 블렌딩 연산(Blend Operation)
ADD: 두 픽셀을 더한다. Result = Source + Destination.
SUBSTRACT: 새로운 픽셀에서 저장되어 있는 픽셀을 뺀다. Result = Source - Destination.
REVSUBSTRACT: 저장된 픽셀에서 새로운 픽셀을 뺀다. Result = Destination - Source.
MAX: R,G,B,A 단위로 두 픽셀 중에 최대 값을 선택한다. Result = MAX(Source, Destination)
MIN: R,G,B,A 단위로 두 픽셀 중에 최소 값을 선택한다. Result = min(Source, Destination)

이 블렌딩 연산 중에서 D3DS의 Default는 ADD입니다.

다음은 D3D가 지원하는 블렌딩 인수입니다. Source와 Destination은 이 인수 중에서 하나를 선택해서 사용합니다.

D3DBLEND_ZERO → (0, 0, 0, 0): 결과는 (0, 0, 0, 0)
D3DBLEND_ONE → (1,1,1,1): 결과는 원본과 동일
D3DBLEND_SRCCOLOR → (R_{src}, G_{src}, B_{src}, A_{src}): Source 쪽 색상이 블렌딩 인수
D3DBLEND_INVSRCCOLOR → (1-R_{src}, 1-G_{src}, 1-B_{src}, 1-A_{src}): Source 색상을 반전한 것이 블렌딩 인수
D3DBLEND_SRCALPHA → (A_{src}, A_{src}, A_{src}, A_{src}): Source의 알파를 각각의 성분에 대한 인수로 사용
D3DBLEND_INVSRCALPHA →(1-A_{src}, 1-A_{src}, 1-A_{src}, 1-A_{src}): Source의 1-알파를 인수로 사용

D3DBLEND_DESTCOLOR → (R_{dst}, G_{dst}, B_{dst}, A_{dst}): Destination 색상이 인수
D3DBLEND_INVDESTCOLOR → (1-R_{dst}, 1-G_{dst}, 1-B_{dst}, 1-A_{dst}): 반전된 Destination 색상이 인수
D3DBLEND_DESTALPHA → (A_{dst}, A_{dst}, A_{dst}, A_{dst}): Destination의 알파를 인수로 사용
D3DBLEND_INVDESTALPHA → (1-A_{dst}, 1-A_{dst}, 1-A_{dst}, 1-A_{dst}): 반전된 Destination 알파가 인수

D3DBLEND_SRCALPHASAT →(f, f, f, 1): f = min(A_{src}, 1 - Ad)
D3DBLEND_BOTHSRCALPHA:
  Source Factor→ (1 - A_{src}, 1 - A_{src}, 1 - A_{src}, 1 - A_{src}).
  Dest Factor→(As, As, As, As). 이 인수 지정은 오직 D3DRS_SRCBLEND에만 이용 가능

이 중에서 알파 블렌딩에 대한 D3D의 Default 설정은 Source 인수에는 D3DBLEND_SRCALPHA, Destination은 D3DBLEND_INVSRCALPHA, 블렌딩 연산 (D3DRS_BLENDOP) 은 ADD(D3DBLENDOP_ADD) 로 설정되어 있습니다. 이것은 반투명 오브젝트가 있을 때 부드럽게 색상의 혼합을 만들어 냅니다.

알파 블렌딩을 D3D에서 구현하려면 Source의 블렌딩 인수, Destination 블렌딩 인수를 설정하고 D3D 파이프라인 상태 머신의 알파 블렌딩 처리를 활성화해야 합니다. 물론 렌더링이 끝나면 상태 머신의 알파 블렌딩은 비활성화(FALSE) 시켜야 합니다. 이것을 의사 코드 만들면 다음과 같습니다.

```
pDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_"블렌딩 인수");
pDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_"블렌딩 인수");
pDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, TRUE);
…
pDevice->DrawPrimitive(…);
pDevice->SetRenderState(D3DRS_ALPHABLENDING, FALSE);
```

8.1.1 알파 블렌딩 연습
알파 블렌딩을 연습하기 위해서 화면에 직사각형 만듭시다. 직사각형이 색상을 가져야 하므로 정점 구조체는 다음과 같이 Diffuse 색상이 있어야 합니다.

```
struct VtxD
{
        D3DXVECTOR3     p;
        DWORD           d;
…
        enum { FVF = (D3DFVF_XYZ|D3DFVF_DIFFUSE)};
```

};

정점의 적당한 값을 카메라 거리에 비례해서 설정합니다. m3d_07_1blend01_rect.zip 예제는 카메라가 붙어있지만 여러분은 꼭 카메라를 만들 필요는 없습니다.
이 예제는 정점 구조체에 대한 정점 버퍼를 만들지 않고 시스템 메모리를 직접 이용하고 있습니다. 이런 경우에는 DrawPrimitiveUP() 함수로 렌더링을 해야 하는 것은 다 알고 있는 내용입니다.

```
m_pVtx[0] = VtxD(-6.f,  5.f, 5.f,    D3DXCOLOR(1,0,0,0));
m_pVtx[1] = VtxD( 6.f,  5.f, 5.f,    D3DXCOLOR(1,0,0,1));
m_pVtx[2] = VtxD( 6.f, -5.f, 5.f,    D3DXCOLOR(1,0,0,1));
m_pVtx[3] = VtxD(-6.f, -5.f, 5.f,    D3DXCOLOR(1,0,0,0));
```

위와 같이 정점 데이터를 만들었는데 눈 여겨 볼 것은 색상입니다. 색상은 전부 붉은 색이지만 왼쪽에 있는 정점은 알파가 0이고, 오른쪽에 있는 정점은 전부 1입니다.

이렇게 정점을 만들고 다음과 같이 화면에 렌더링 할 때 단지 상태 머신의 알파 블렌딩 활성화만 시켜도 다음 그림처럼 왼쪽에서 오른쪽으로 완전 투명에서 불투명으로 변화하는 붉을 색을 볼 수 있습니다.

```
m_pDev->SetRenderState(D3DRS_ALPHABLENDENABLE, TRUE);

m_pDev->SetTexture(0, NULL);
m_pDev->SetFVF(VtxD::FVF);
m_pDev->DrawPrimitiveUP(D3DPT_TRIANGLEFAN, 2, m_pVtx, sizeof(VtxD));
```


<알파 블렌딩: Diffuse 이용 m3d_07_1blend01_rect.zip>

D3D 가상 머신의 어떤 값도 설정 안 했다면 위와 같이 구현이 되며 이것은 다음과 같이 Source에 SRCALPHA 인수를, Destination에 INVSRCALPHA를 연결한 것과 같은 결과를 만듭니다.

```
m_pDev->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
m_pDev->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);
m_pDev->SetRenderState(D3DRS_ALPHABLENDENABLE, TRUE);
...
m_pDev->DrawPrimitiveUP(D3DPT_TRIANGLEFAN, 2, m_pVtx, sizeof(VtxD));
```

이 방법을 수식으로 표현하면 Source * SrcAlpha + Dest * (1-SrcAlpha)로 선형보간이 됩니다. 이것은 즉, Source의 색상과 Destination을 알파를 이용해서 자연스럽게 섞는 방법인 것입니다.

DXSDK에 있는 옆 그림과 같은 텍스처를 매핑 해 봅시다.
텍스처 매핑을 위해서 정점 구조체를 U,V가 설정될 수 있도록 다음과 같이 변경합니다.

```
struct VtxDUV1
{
        D3DXVECTOR3     p;      // 위치
        DWORD           d;      // 색상
        FLOAT           u,v;    // U, V 좌표

        enum { FVF = (D3DFVF_XYZ|D3DFVF_DIFFUSE|D3DFVF_TEX1)};
};
```

정점의 4 위치에 텍스처 매핑 좌표를 설정합니다.

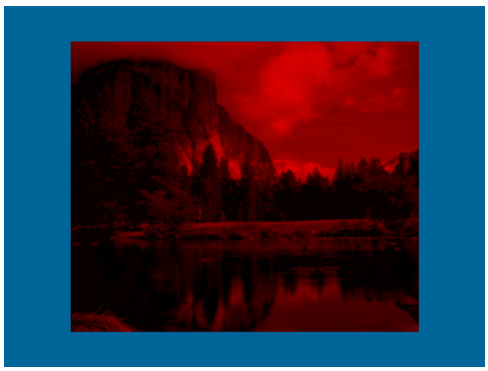
```
m_pVtx[0] = VtxDUV1(-6.f,  5.f, 5.f,    0.f, 0.f,  D3DXCOLOR(1,0,0,0));
m_pVtx[1] = VtxDUV1( 6.f,  5.f, 5.f,    1.f, 0.f,  D3DXCOLOR(1,0,0,1));
m_pVtx[2] = VtxDUV1( 6.f, -5.f, 5.f,    1.f, 1.f,  D3DXCOLOR(1,0,0,1));
m_pVtx[3] = VtxDUV1(-6.f, -5.f, 5.f,    0.f, 1.f,  D3DXCOLOR(1,0,0,0));
```

이제 렌더링을 위해서 다시 알파 블렌딩을 활성화하고 텍스처를 샘플러에게 연결하고 렌더링 합니다.

```
m_pDev->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
m_pDev->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);
m_pDev->SetRenderState(D3DRS_ALPHABLENDENABLE, TRUE);
m_pDev->SetTexture(0, m_pTex);
```

```
m_pDev->SetFVF(VtxDUV1::FVF);
m_pDev->DrawPrimitiveUP(D3DPT_TRIANGLEFAN, 2, m_pVtx, sizeof(VtxDUV1));
```

아마 다음 그림과 같이 알파 블렌딩이 적용 안된 상태로 출력 되는 것이 대부분입니다. 이럴 때 제일 먼저 살펴보아야 할 것이 다중 텍스처 처리(Multi-Texturing)을 살펴 봐야 합니다.



지금은 0번 단계에서 만 처리하므로 0 번 단계(0-Stage)의 알파 연산을 다음 코드와 같이 확인해야 합니다.

```
DWORD v;
m_pDev->GetTextureStageState(0, D3DTSS_ALPHAARG1, &v);  // Default는 Texture
m_pDev->GetTextureStageState(0, D3DTSS_ALPHAARG2, &v);  // Default는 Current
m_pDev->GetTextureStageState(0, D3DTSS_ALPHAOP, &v);    // Default는 SelectArg1
```

최초에는 코드의 주석처럼 ALPHAOP가 SelectArg1으로 되어 있고 SelectArg1은 Texture로 되어 있습니다. 따라서 알파 없는 불투명 텍스처만 그리게 되어 알파 블렌딩이 적용이 안된 것입니다.

이 것을 다음과 같이 수정합니다.

```
m_pDev->SetTextureStageState(0, D3DTSS_ALPHAARG1, D3DTA_TEXTURE);
m_pDev->SetTextureStageState(0, D3DTSS_ALPHAARG2, D3DTA_DIFFUSE);
m_pDev->SetTextureStageState(0, D3DTSS_ALPHAOP,  D3DTOP_MODULATE);
```

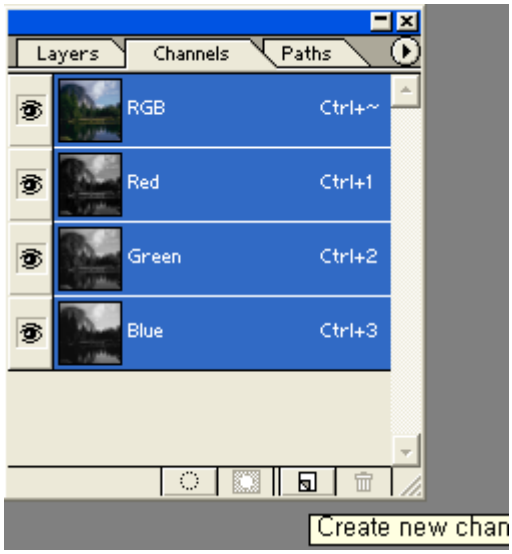
이것은 텍스처의 알파와 정점의 Diffuse의 알파 값을 서로 곱하는 명령입니다. 더하는 것이 좋아 보일 수 있지만 지금처럼 불투명 텍스처는 알파가 1이 되어 그 결과도 1을 넘을 수 있어서 반투명을 만들어 내지 못합니다. 따라서 곱하기가 제일 좋습니다.

이렇게 알파만 설정할 것이 아니라 R, G, B 색상에 대해서도 같은 코드를 만들어 주는 것이 좋습

니다. 이 부분은 생략하겠습니다.
이제 다시 실행하면 다음 그림처럼 목적했던 장면을 만들 수 있습니다.

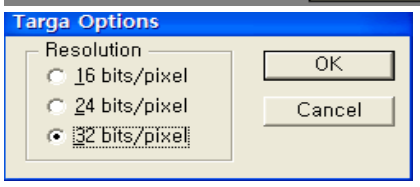

<m3d_07_1blend02_no_alpha.zip>

지금까지 불투명한 텍스처를 반투명한 정점에 매핑 해서 렌더링 해봤습니다. 다음으로 알파가 있는 텍스처를 적용해 보겠습니다.
보통의 이미지를 가지고 알파가 있는 이미지를 만들기 위해서는 포토샵과 같은 그래픽 툴이 필요합니다. (아니면 프로그램을 만들어야 합니다.)
이미지 포맷 중에 BMP, JPG 등은 Alpha 가 없습니다. 알파를 가지고 있는 포맷은 TGA, PNG가 대표적입니다.
TGA 포맷은 흔히 Targa(Truevision TGA)로 부르며 이 포맷은 무 손실 (비)압축 포맷으로 특히 압축되지 않은 형식은 게임에서 가장 많이 사용되는 형태 중에 하나입니다.
PNG는 Potable Network Graphics로 무 손실 압축 포맷입니다. 이 포맷도 게임 작업에서 가장 많이 사용되는 포맷입니다.



TGA, PNG는 다른 이미지의 픽셀이 손실되지 않은 관계로 게임 그래픽 작업용으로 활용하기에 가장 좋으며 나머지 BMP, JPG는 사용 안 하는 것이 좋습니다. BMP는 32비트도 저장이 가능하지만 그래픽 툴에서 지원이 안 되는 경우도 있습니다. JPG는 손실 압축이 되어 이미지의 변형이 생겨 알파 적용이 잘 안 될 수 도 있어서 지금은 PNG나 TGA 포맷을 게임 그래픽 작업용으로 많이 사용합니다.

이미지를 포토샵에서 열고 옆의 그림처럼 채널(Channels)을 선택하면 이 이미지의 RGB 그리고 알파 채널을 볼 수 있습니다.



만약 알파 채널이 없다면 휴지통 옆의 버튼을 눌러 새로운 채널을 만들어야 합니다. 이렇게 채널을 만들면 검정색 또는 흰색으로 채널을 전부 채워진 상태가 됩니다. 브러시를 이용해서 흰색(불투명) 또는 검정색(완전 투명)을 선택한 후에 적절하게 그리고 나서 저장할 때 TGA포맷으로 저

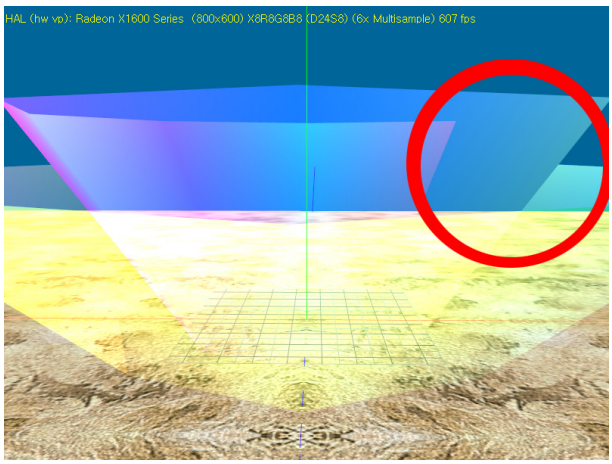
장합니다. 이때 TGA 해상도 옵션이 화면에 나타나면 32bit를 선택하고 저장합니다.
이렇게 알파 채널이 있는 TGA파일을 적용하면 다음 그림과 같은 장면을 만들 수 있습니다.


<알파 텍스처: m3d_07_1blend03_alpha.zip>

PNG 파일로 알파 채널 만드는 것은 많이 알려져 있어서 생략하겠습니다.
이전에는 다중 처리에서 색상을 혼합하는 법을 배웠고, 지금은 알파를 혼합하는 법을 배웠습니다. 게임에서 정점의 Diffuse와 텍스처의 알파를 혼합해서 사용하는 경우가 많으므로 특별히 연습이 더 필요합니다. 또한 이렇게 알파를 처리하는 법을 알았다면 지금까지의 내용 가지고도 2D 게임에서 사용하는 스프라이트를 3D로 완전하게 만들 수 있습니다. 이것은 과제로 남겨 놓겠습니다.

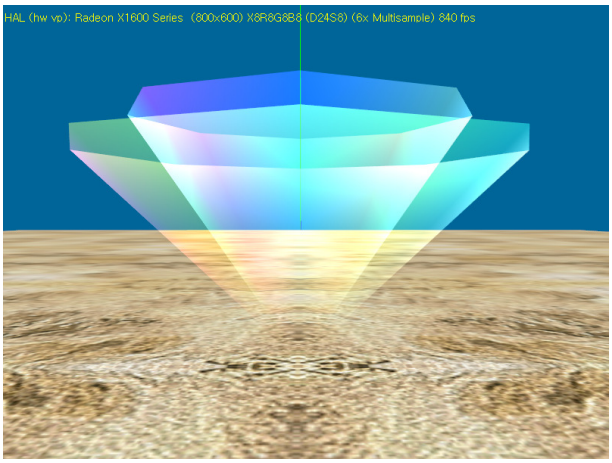
8.2 Z - Write

오브젝트를 반 투명으로 만들어서 렌더링 하면 화면에 근사한 장면을 많이 만들어 낼 수 있습니다.
여러분이 알파 블렌딩을 이용해서 다음과 같이 폴리곤을 화면에 출력했다고 가정 합시다.
괜찮은 장면 같지만 붉은 색의 동그라미를 자세히 보면 삼각형이 출력하다가 말았습니다.



이것은 디바이스가 z 값을 비교해서 후면 버퍼의 내용을 갱신하기 때문입니다. 후면 버퍼의 입장에서 앞으로 갱신할 픽셀의 깊이 값(Depth Value)가 자신이 가지고 있는 것보다 크면 장면을 그릴 이유가 없다고 판단 픽셀을 안 바꾸게 되면 속도에 이득이 있는 것은 당연합니다.

그래서 장면을 그릴 때 카메라의 위치에서 z축으로 가까운 오브젝트부터 z축으로 멀리 떨어진 오브젝트를 그리게 하면 렌더링 속도가 빨라지



게 됩니다. 그런데 이것은 오로지 불투명일 때 맞는 이야기 입니다. 반투명일 때는 이와 반대로 해야 후면 버퍼의 픽셀을 계속 갱신을 하게 되어 알파 블렌딩이 효과적으로 적용이 됩니다. 그렇다고 해도 위의 문제가 완전히 해결이 된 것은 아닙니다. 만약 한 폴리곤에서 겹겹이 쌓여 있는 폴리곤의 경우에는 DrawPrimitive()에서 한꺼번에 그리므로 순서를 결정할 수 없는 문제가 생깁니다.
그래서 찾은 방법이 Z-Test를 하지 않고 그리는 것입니다.

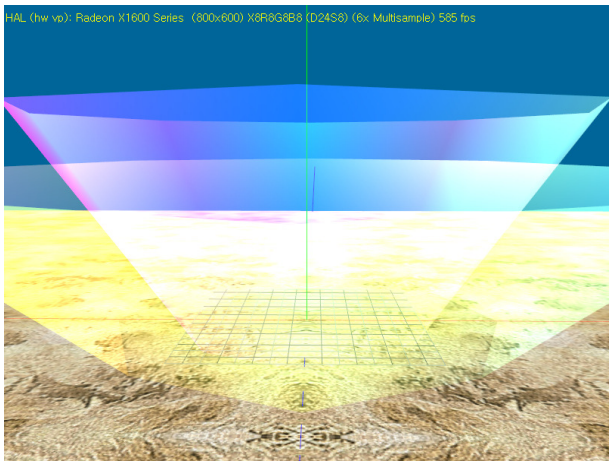
```
pDevice->SetRenderState(D3DRS_ZENABEL, FALSE);
```

이렇게 하면 깊이 테스트를 안 하기 때문에 무조건 갱신해서 아래와 같이 만들 수 있지만 문제는 오브젝트를 그릴 대 마다 이전 픽셀들을 무조건 덮어 버리게 되어 해결 책이 아니게 됩니다.

현재까지 가장 적절한 방법은 먼저 불투명 오브젝트를 카메라에서 가까운 거리부터 그리고 그 다음으로 반투명 오브젝트들을 먼 쪽부터 그리도록 하는데 이 때 후면 버퍼의 깊이 값을 쓰지 않고 넘어 방법이 있습니다.

```
m_pDev->SetRenderState( D3DRS_ZWRITEENABLE, FALSE);
```

이렇게 하면 새로운 픽셀은 항상 깊이 테스트를 거의 통과 하게 되어 다음 그림과 같은 효과를 만들 수 있습니다.



D3DRS_ZWRITEENABLE-FALSE 이 방법은 이 후 파티클에서 알파 블렌딩과 더불어 상태 머신을 설정할 때 사용되니 꼭 기억하기 바랍니다.
D3DRS_ZENABEL-FALSE 는 2D에서 많이 사용합니다. 2D의 경우 렌더링이 무조건 이루어 져야 하므로 3D로 2D Sprite를 만들 때 깊이 테스트를 하지 못하도록 설정을 합니다.
물론 설정을 했으면 렌더링 후에 다시 원래대로(TRUE)로 바꾸어놓아야 합니다.

<m3d_07_1blend03_zwrite.zip>

8.2 Alpha Test

앞서 테스트란 목적 버퍼의 픽셀을 현재의 픽셀로 대치할 것인가에 대한 평가라 했습니다. 고정 함수 파이프라인에서 테스트에 사용되는 함수들은 비교 함수 입니다. 이 비교 함수들은 알파 테스트, 깊이 테스트, 스텐실 테스트에서 같이 사용되는 함수들입니다.

알파 테스트의 경우 "새로운 알파 값" (비교 함수) "참조 값" 으로 해석하면 됩니다.

비교 함수들은 _D3DCMPFUNC알파에 다음과 같이 선언 되어 있습니다.

```
typedef enum _D3DCMPFUNC {
    D3DCMP_NEVER               = 1,
    D3DCMP_LESS                = 2,
    D3DCMP_EQUAL               = 3,
    D3DCMP_LESSEQUAL           = 4,
    D3DCMP_GREATER             = 5,
    D3DCMP_NOTEQUAL            = 6,
    D3DCMP_GREATEREQUAL        = 7,
    D3DCMP_ALWAYS              = 8,
} D3DCMPFUNC;
```

NEVER: 테스트를 절대 통과 못합니다. 버퍼에 새로운 픽셀을 갱신 안 합니다.
LESS: New<참조. 새로운 픽셀의 투명도가 높으면(알파 값이 작으면) 테스트를 통과합니다.
EQUAL: New == 참조. 새로운 픽셀과 버퍼에 저장된 픽셀의 알파 값이 같은 픽셀 통과합니다.
LESSEQUAL: New ≤ 참조. 새로운 픽셀의 알파 값이 같거나 작은 경우에만 픽셀은 통합니다.
GREATER: New > 참조. 새로운 픽셀의 알파 값이 같거나 작은 경우에만 픽셀은 통합니다.
NOTEQUAL: New != 참조. 알파 값이 같지 않은 픽셀 만 통과합니다.
GREATEREQUAL: New ≥ 참조. 알파가 크거나 (불투명도가 높은) 같으면 테스트를 통과합니다.
D3DCMP_ALWAYS: 항상 통과해서 새로운 픽셀로 버퍼를 갱신합니다.

알파 테스트를 D3D에 적용하려면 다음과 같이 상태 머신의 알파 테스트를 활성화 시키고 비교 함수와 테스트에 대한 "참조 값"을 설정 해야 합니다.

```
pDevice->SetRenderState(D3DRS_ALPHATESTENABLE, TRUE);
pDevice->SetRenderState(D3DRS_ALPHAFUNC, "비교 함수");
pDevice->SetRenderState(D3DRS_ALPHAREF, "참조 값");
```

물론 렌더링을 끝낸 후에 알파 테스트를 비활성화 시켜야 하는 것은 당연 합니다.

알파 테스트를 구현하기 위해서 먼저 디바이스에 설정되어 있는 알파 테스트에 대한 값들을 확인해 봅니다. 이것은 렌더링에서 알파 테스트가 제대로 되지 않을 경우 제일 먼저 확인해 보는 방법입니다.

```
DWORD v;
m_pDev->GetRenderState(D3DRS_ALPHAFUNC, &v);
m_pDev->GetRenderState(D3DRS_ALPHAREF, &v);
```

D3D는 디폴트로 알파 테스트는 비활성화 되어 있고 비교 함수는 ALWAYS, 참조 값은 0으로 설정되어 있어서 알파 테스트를 활성화 시켜도 아무런 변화는 없습니다.

m3d_07_2alphatest.zip은 알파 테스트를 연습하기 위한 예제 입니다. 컴파일 하고 실행하면 다음과 같은 화면으로 시작됩니다.


<m3d_07_2alphatest.zip 실행 화면>

숫자 키를 1~8까지 눌러보면 각 비교 함수에 대한 차이를 볼 수 있습니다. 또한 우측의 키 패드에 있는 '+' 키와 '-' 키를 눌러서 참조 값을 변경시킬 수 있습니다. 다음 그림들은 참조 값을 128로 하고 Less ~ GreaterEqual까지 테스트한 화면입니다.


-Less-


-Equal-


-Less Equal-

-Greater-

-Not Equal-

-Greater Equal-

<알파 테스트의 비교 함수 연습>

주의할 것은 알파 테스트와 알파 블렌딩은 아무런 상관이 없습니다. 테스트의 목적은 버퍼의 갱신 여부를 판별하는 것이기 때문입니다.

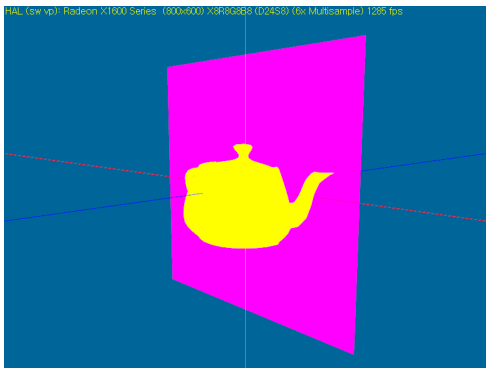
8.3 깊이 테스트(Depth Test)

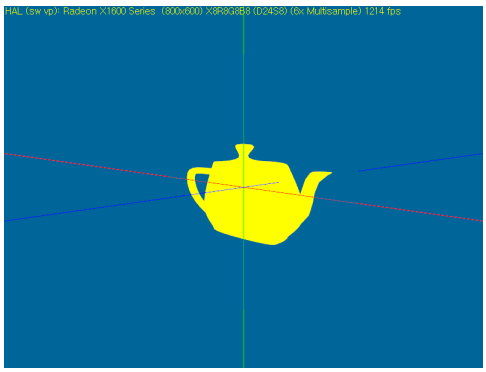
깊이 테스트도 알파 테스트와 마찬가지로 픽셀을 버퍼에 쓸 것인가 말 것인가를 결정하는 것입니다. 앞서 D3D는 비교 함수를 깊이, 알파, 스텐실 테스트에서 같이 사용한다고 했습니다.
깊이 테스트는 알파 테스트의 참조 값 대신에 깊이 값을 이용합니다. 이 깊이 값(Depth Value)은 정규 변환, 뷰포트 변환을 거친 Z 값입니다. 깊이 테스트는 알파 테스트의 참조 값은 없습니다. 따라서 깊이 테스트 활성화, 비교 함수 만 설정하면 됩니다.

```
pDevice->SetRenderState(D3DRS_ZWRITEENABLE,  TRUE);
pDevice->SetRenderState(D3DRS_ZFUNC,  "비교 함수");
```

물론 당연히 D3D는 깊이 테스트가 디폴트로 활성화 되어 있습니다. 디폴트 비교 함수는 LESSEQUAL로 깊이 값이 작거나 같은 경우에만 테스트를 통과하게 되어 있습니다. 또한 ZENABLE-FALSE는 항상 깊이 테스트를 무시하는 경우이므로 비교 함수가 ALWAYS와 같은 경우 입니다.

보통 주전자와 평면을 같이 그리면 다음 그림의 왼쪽처럼 화면에 나오는 것이 일반적입니다.





<m3d_07_3depthtest.zip. 깊이 테스트를 변형해서 잘린 주전자 표현>

그런데 깊이 테스트를 변경하면 오른쪽 그림처럼 주전자가 잘린 모양을 만들어 낼 수 도 있습니다. m3d_07_3depthtest.zip의 void CMcScene::Render() 함수 안을 보면 다음과 같은 코드를 볼 수 있습니다.

```
m_pDev->SetRenderState(D3DRS_COLORWRITEENABLE, 0);      // 색상을 안 쓴다.
m_pDev->SetRenderState(D3DRS_FILLMODE, D3DFILL_SOLID);
m_pDev->SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);
m_pDev->SetRenderState(D3DRS_ZWRITEENABLE, TRUE);
m_pDev->SetFVF(VtxD::FVF);
m_pDev->DrawPrimitiveUP(D3DPT_TRIANGLEFAN, 2, m_Plan, sizeof(VtxD));

m_pDev->SetRenderState(D3DRS_ZWRITEENABLE, TRUE);
m_pDev->SetRenderState(D3DRS_COLORWRITEENABLE, 0XF);
m_pDev->SetRenderState(D3DRS_ZFUNC,  D3DCMP_GREATER);
m_pDev->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);
m_pDev->SetRenderState(D3DRS_LIGHTING, FALSE);
m_pDev->SetTransform(D3DTS_WORLD, &mtWld);
m_Teapot->DrawSubset(0);
```

D3D의 후면 버퍼는 Color, Depth, Stencil 세 개의 버퍼로 구성되어 있습니다. 이중 색상 버퍼에 아무 것도 쓰지 못하게 하려면 상태 COLORWRITEENABLE 에 상태 값 0으로 합니다. 반대로 색상을 다시 쓸 수 있도록 하려면 TRUE가 아닌 0xF로 해야 합니다. 이것은 D3D가 색상 갱신을 R, G, B 각각 따로 활성화 비활성화 할 수 있기 때문입니다.

위의 코드 6줄 까지는 색상을 쓰지 않고 깊이만 버퍼에 기록이 됩니다. 그 다음에는 주전자를 그리는데 주전자의 깊이 비교 값은 Greater로 주전자의 깊이 값이 큰 것만 기록을 하게 됩니다. 그런데 이미 평면의 깊이 값이 깊이 버퍼에 저장된 상태이므로 이 값으로 비교 합니다. 따라서 잘려진 주전자가 출력 되는 것입니다.

지금까지 알파 블렌딩, 알파 테스트, 깊이 테스트에 대해서 알아보았습니다. 깊이 테스트는 다음 장의 스텐실 테스트와 많이 결합해서 사용하므로 같이 공부하는 것이 좋습니다.

실습 과제) 2장의 이미지를 준비하고 알파 블렌딩 인수, 알파 블렌딩 연산을 비교할 수 있는 툴 프로그램을 제작하시오.