Stencil

305890
2008년 봄학기
4/16/2007
박경신

Overview

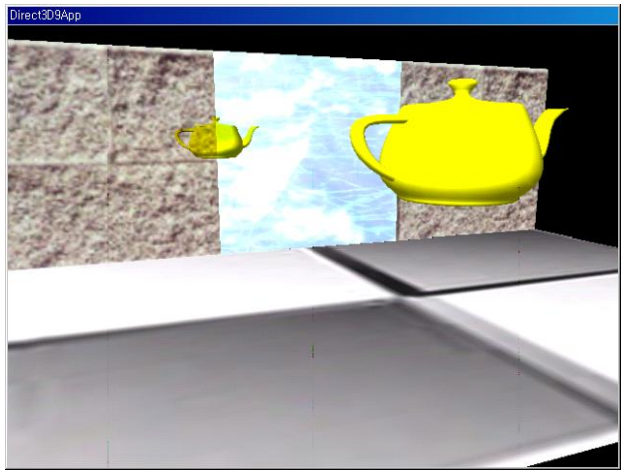
- Using the Stencil Buffer
  - 스텐실 버퍼의 작동 원리
  - 스텐실 버퍼를 만드는 방법.
  - 스텐실 버퍼를 제어하는 방법
- Mirrors
  - 거울을 구현하는 방법
  - 스텐실 버퍼를 이용해 거울이 아닌 표면에 반사가 일어나는 것을 막는 방법
- Planar Shadows
  - 그림자를 렌더링하는 방법
  - 스텐실 버퍼를 이용해 '더블 블렌딩'을 막는 방법

Stencil Buffer

- Stencil Buffer
  - 특수한 효과를 위한 off-screen buffer임
  - Backbuffer, depth buffer와 동일한 해상도임
  - Stencil buffer는 back buffer의 일정 부분이 rendering되지 않도록 함.
  - 대표적인 활용: 거울, 그림자의 구현
  - 예: 벽면에 거울이 있는 경우, 벽면을 제외하고 거울이 있는 영역에 대해서만 반사되는 물체의 drawing을 수행하도록 함.

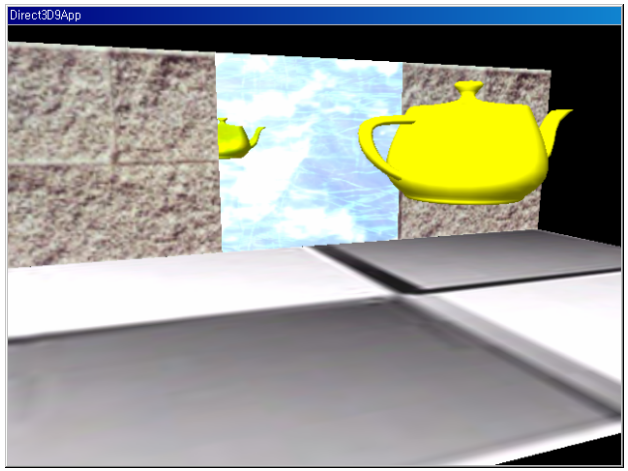
Mirror Effect

- 스텐실 버퍼를 이용하지 않음



Mirror Effect

- 스텐실 버퍼를 이용



Using Stencil Buffer

- Stencil Buffer 이용하기
  - Direct3D초기화 과정에서 stencil buffer를 요청해야 함
  - 이용할 시에 이를 활성화시켜야 함

```
Device->SetRenderState(D3DRS_STENCILENABLE, true); //활성화
…  // stencil관련 작업을 수행
Device->SetRenderState(D3DRS_STENCILENABLE, false);
```

- Stencil buffer의 내용을 clear하기 (back buffer, depth buffer에서의 방법과 동일함)

```
Device->clear(0 /* num of rectangles */, 0 /* rectangles */,
        D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER|D3DCLEAR_STENCIL,
        0xff000000 /* target */,
        1.0f /* depth */,
        0 /* stencil */);
```

Using Stencil Buffer

- Stencil Buffer 요청하기
  - 스텐실 버퍼는 깊이 버퍼와 함께 생성하고 버퍼도 공유함
  - 깊이 버퍼의 포멧을 지정할 때 스텐실 버퍼의 포멧도 함께 지정가능함
- Depth/Stencil buffer format
  - D3DFMT_D24S8: 32-bit depth/stencil buffer, pixel당 24-bit depth buffer/8-bit stencil buffer에 할당함
  - D3DFMT_D24X4S4: 32-bit depth/stencil buffer, pixel당 24-bit depth buffer/4-bit stencil buffer에 할당함. 나머지 4-bit는 이용하지 않음
  - D3DFMT_D15S1: 16-bit depth/stencil buffer; pixel당 15-bit depth buffer/1-bit stencil buffer에 할당함.
  - D3DFMT_D32는 depth buffer에만 32-bit를 할당함 (stencil buffer는 없음)
  - Stencil 지원은 graphics card에 따라 달라질 수 있음 (8-bit stencil을 지원하지 않는 카드도 있음)

Stencil Test

- Stencil test
  - 스텐실 버퍼를 사용하여 Back buffer의 일정 부분이 rendering되는 것을 막음. 특정 pixel의 rendering여부를 결정해야 함.
  - (StencilRef & Stencil Mask) CompFunc (StencilBufferValue & StencilMask)
    - StencilRef: app이 정의한 stencil reference 값. Default는 0. 임의의 정수값
    - StencilMask: app이 정의한 stencil mask값. 사용하지 않을 bit들을 감추는 용도. Default 0xffffffff
    - StencilBufferValue: 테스트하려는 현재 pixel의 stencil buffer값
    - 스텐실 테스트 결과가 true이면 스텐실을 통과하여 back buffer에 픽셀을 출력함. False이면 픽셀은 출력되지 않고 깊이 버퍼에도 안 써짐.

Stencil Test Control

- Stencil test에 스텐실 참조 값 (reference value), 마스크 (mask), 비교 연산자(comparison function)를 지정
- Stencil test를 위한 값 지정의 예

```
// enable stencil test
pDevice->SetRenderState(D3DRS_STENCILENABLE, TRUE);
// specify the stencil comparison function
pDevice->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_EQUAL);
// set the comparison reference value
pDevice->SetRenderState(D3DRS_STENCILREF, 0x1);
// specify a stencil mask
pDevice->SetRenderState(D3DRS_STENCILMASK, 0x0000ffff);
```

상위 16-bit를 감춤

Default는 0.
16진수 이용하면 비트 배열 확인 및 AND 연산에 수월

Stencil Test Control

- 비교연산자 (CompFunc)의 지정:

  Device->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_LESS);
- D3DCMPFUNC enum type의 요소:
  - D3DCMP_ALWAYS: default임. 항상 test를 통과하도록 함
  - D3DCMP_NEVER: test가 항상 실패함
  - D3DCMP_LESS/EQUAL/LESSQUAL/GREATER/NOTEQUAL/GREATEREQUAL: lhs < / = / <= / > / != / >= rhs

Stencil Buffer Update

- Stencil test 후 3가지 상황에 따라 스텐실 버퍼의 항목이 갱신되는 방법을 지정
  - 스텐실 테스트가 실패한 경우 stencil buffer갱신 방법을 정의:

    Device->SetRenderState(D3DRS_STENCILFAIL, StencilOperation);
  - 깊이 테스트가 실패한 경우 stencil buffer갱신 방법을 정의 :

    Device->SetRenderState(D3DRS_STENCILZFAIL, StencilOperation);
  - 스텐실 테스트와 깊이 테스트가 모두 성공한 경우 stencil buffer갱신 방법을 정의:

    Device->SetRenderState(D3DRS_STENCILPASS, StencilOperation);

D3DSTENCILOP_KEEP

Stencil Buffer Update

- Stencil Operation
  - 모든 D3DRS_STENCILXXX에 대한 default는 D3DSTENCILOP_KEEP 임.
  - D3DSTENCILOP_KEEP/ZERO/REPLACE/INVERT: stencil buffer항목을 간직, 0으로, StencilRef값으로, 또는 반전.
  - D3DSTENCILOP_INCRSAT/DECRSAT/INCR/DECR: stencil buffer항목을 증가(최대값보다크면 최대값으로), 감소(0미만이면 0으로), 증가(최대값보다 크면 0으로), 또는 감소(0미만이면 최대값으로)

Stencil Write Mask

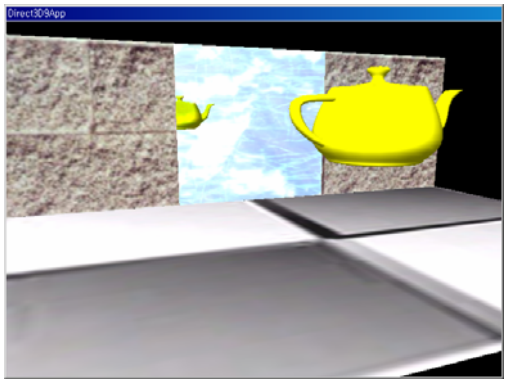
- Stencil Write Mask
  - Stencil buffer에 쓰여지는 모든 값들을 mask함.
  - Default는 0xffffffff

```
Device->SetRenderState(D3DRS_STENCILWRITEMASK, 0x0000ffff);
```

상위 16-bit를 감춤

Sample: StencilMirrors

- 거울 (mirror) 구현
  - 임의의 평면에 물체가 반사되는 방법 파악
  - 거울 영역에만 반사를 그려야 함 – 스텐실 버퍼이용



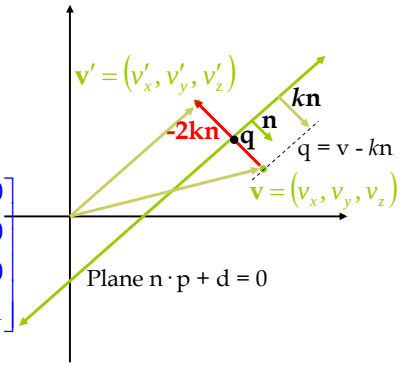
Reflection

- 거울 평면 (n, d)에 대해 점 $v=(v_x, v_y, v_z)$의 반사 포인트 $v'=(v_x', v_y', v_z')$를 계산하는 방법

$$\begin{aligned}\mathbf{v}' &= \mathbf{v} - 2k\mathbf{n} \\ &= \mathbf{v} - 2(\mathbf{n}\cdot\mathbf{v} + d)\mathbf{n} \\ &= \mathbf{v} - 2[(\mathbf{n}\cdot\mathbf{v})\mathbf{n} + d\mathbf{n}]\end{aligned}$$

$$\mathbf{v}' = \mathbf{v}R$$

$$\mathbf{R} = \begin{bmatrix} -2n_xn_x+1 & -2n_yn_x & -2n_zn_x & 0 \\ -2n_xn_y & -2n_yn_y+1 & -2n_zn_y & 0 \\ -2n_xn_z & -2n_yn_z & -2n_zn_z+1 & 0 \\ -2n_xd & -2n_yd & -2n_zd & 1 \end{bmatrix}$$



k: v에서 plane로의 signed shortest distance
n이 unit vector인 경우, $k = \mathbf{n}\bullet\mathbf{v} + d$

Reflection

- D3DX 라이브러리는 R과 같은 임의의 평면에 대한 반사 행렬을 만들어내는 함수 제공

```
D3DXMATRIX *D3DXMatrixReflect(
                    D3DXMATRIX *pOut,
                    CONST D3DXPLANE *pPlane);
```

  - 내부적으로 plane을 정규화한 후, 반사행렬 R을 계산함.
- 세 가지 특수한 경우
  - 표준 좌표 평면(*yz*, *xz*, *xy* 평면)에 대한 반사 변환 행렬

$$\mathbf{R}_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example: StencilMirror Overview

1. 바닥, 벽, 거울, 주전자 등 전체 장면을 보통 때와 마찬가지로 rendering 함.
   - 주전자 반사는 아직 포함되지 않음

2. 스텐실 버퍼를 0 값으로 clear 함.
3. 스텐실 버퍼에만 거울을 구성하는 기본 도형(face)을 rendering 함.
   - 스텐실 테스트가 항상 성공하도록 하고 테스트가 성공하면 stencil buffer 항목을 1로 지정하도록 함. (거울에 해당하는 pixel들만 1의 값을 가짐)

4. 반사된 주전자를 후면 버퍼와 스텐실 버퍼로 rendering 함.
   - 스텐실 테스트 통과한 부분만 후면 버퍼에 렌더링. Stencil buffer항목이 1인 경우에만 stencil test를 pass하도록 지정함. 따라서 반사된 주전자가 거울에만 rendering됨.

Example: StencilMirror

```
#include "d3dUtility.h"

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

IDirect3DVertexBuffer9* VB = 0;
IDirect3DTexture9* FloorTex= 0;
IDirect3DTexture9* MirrorTex= 0;

D3DMATERIAL9 FloorMtrl = d3d::WHITE_MTRL;
D3DMATERIAL9 WallMtrl = d3d::WHITE_MTRL;
D3DMATERIAL9 MirrorMtrl = d3d::WHITE_MTRL;

ID3DXMesh* Teapot = 0;
D3DXVECTOR3 TeapotPosition(0.0f, 3.0f, -7.5f);
D3DXMATERIAL9 TeapotMtrl = d3d::YELLOW_MTRL;
```

Example: StencilMirror

```
Struct Vertex {
    Vertex() {}
    Vertex(float x, float y, float z, float nx, float ny, float nz, float u, float v) {
        _x = x; _y = y; _z = z;
        _nz = nx, _ny = ny; _nz = nz;
        _u = u; _v = v;
    }
    float _x, _y, _z;
    float _nx, _ny, _nz;
    float _u, _v;
    static const DWORD FVF;
};
Const DWORD Vertex::FVF =
    D3DFVF_XYZ|D3DFVF_NORMAL|D3DFVF_TEX1;
```

Example: StencilMirror

```
bool Setup() {
    WallMtrl.Specular = d3d::WHITE * 0.2f;
    D3DXCreateTeapot(Device, &Teapot, 0);  // create teapot
    Device->CreateVertexBuffer(24 *sizeof(Vertex), 0, Vertex::FVF,
                                    D3DPOOL_MANAGED, &VB, 0);
    Vertex *v = 0;
    VB->Lock(0, 0, (void**)&v, 0);
    v[0] = Vertex(-7.5f, 0.0f, -10.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f);
    v[1] = Vertex(-7.5f, 0.0f,    0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f);
    v[2] = Vertex( 7.5f, 0.0f,    0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f);
    …




    v[23] = Vertex( 2.5f, 0.0f,   0.0f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f);
    VB->Unlock();
```

Example: StencilMirror

```
    // load textures, set filters
    D3DXCreateTextureFromFile(Device, "checker.jpg", &FloorTex);
    D3DXCreateTextureFromFile(Device, "brick0.jpg", &WallTex);
    D3DXCreateTextureFromFile(Device, "ice.bmp", &MirrorTex);
    Device->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
    Device->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
    Device->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);
    // light
    D3DXVECTOR3 lightDir(0.707f, -0.707f, 0.707f);
    D3DXCOLOR color(1.0f, 1.0f, 1.0f, 1.0f);
    D3DLIGHT9 light = d3d::InitDirectionalLight(&lightDir, &color);
    Device->SetLight(0, & light);
    Device->LightEnable(0, true);
    Device->SetRenderState(D3DRS_NORMALIZENORMALS, true);
    Device->SetRenderState(D3DRS_SPECULARENABLE, true);
    // set camera
    // set projection matrix
    return true;
}
```

Example: StencilMirror

```
bool Display(float timeDelta) {
    if (Device) {
        static float radius = 20.0f;
        if (::GetAsyncKeyState(VK_LEFT) & 0x8000f)
          TeapotPosition.x -= 3.0f * timeDelta;
        if (::GetAsyncKeyState(VK_RIGHT) & 0x8000f)
          TeapotPosition.x += 3.0f * timeDelta;
        if (::GetAsyncKeyState(VK_UP) & 0x8000f)
          radius -= 2.0f * timeDelta;
        if (::GetAsyncKeyState(VK_DOWN) & 0x8000f)
          radius += 2.0f * timeDelta;
        static float angle = (3.0f * D3DX_PI) / 2.0f;
        if (::GetAsyncKeyState('A') & 0x8000f) angle -= 0.5f * timeDelta;
        if (::GetAsyncKeyState('S') & 0x8000f) angle += 0.5f * timeDelta;
```

Example: StencilMirror

```
        D3DXVECTOR3 position(cosf(angle)*radius, 3.0f, sinf(angle)*radius);
        D3DXVECTOR3 target(0.0f, 0.0f, 0.0f);
        D3DXVECTOR3 up(0.0f, 1.0f, 0.0f);
        D3DXMATRIX V;
        D3DXMatrixLookAtLH(&V, &position, &target, &up);
        Device->SetTransform(D3DTS_VIEW, &V);
        // draw scene
        Device->Clear(0, 0, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER|
                    D3DCLEAR_STENCIL, 0xff000000, 1.0f, 0L);
        Device->BeginScene();
        RenderScene();
        RenderMirror();
        Device->EndScene();
        Device->Present(0, 0, 0, 0);
    }
    return true;
}
```

Example: StencilMirror

```
void RenderScene() {
    // draw teapot
    Device->SetMaterial(&TeapotMtrl);
    Device->SetTexture(0, 0);
    D3DXMATRIX W;
    D3DXMatrixTranslation(&W, TeapotPosition.x, TeapotPosition.y,
                                TeapotPosition.z);
    Device->SetTransform(D3DTS_WORLD, &W);
    Teapot->DrawSubset(0);

    D3DXMATRIX I;
    D3DXMatrixIdentity(&I);
    Device->SetTransform(D3DTS_WORLD, &I);
    Device->SetStreamSource(0, VB, 0, sizeof(Vertex));
    Device->SetFVF(Vertex::FVF);
```

Example: StencilMirror

```
    // draw floor
    Device->SetMaterial(&FloorMtrl);
    Device->SetTexture(0, FloorTex);
    Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);
    // draw wall
    Device->SetMaterial(&WallMtrl);
    Device->SetTexture(0, WallTex);
    Device->DrawPrimitive(D3DPT_TRIANGLELIST, 6, 4);
    // draw mirror
    Device->SetMaterial(&MirrorMtrl);
    Device->SetTexture(0, MirrorTex);
    Device->DrawPrimitive(D3DPT_TRIANGLELIST, 18, 2);
}
```

Example: StencilMirror

```
void RenderMirror() {                                   스텐실버퍼활성화, 관련된 렌더값지정
    // draw mirror quad to stencil buffer ONLY
    Device->SetRenderState(D3DRS_STENCILENABLE, true);
    // stencil test가 항상 성공하도록 함
    Device->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_ALWAYS);
    Device->SetRenderState(D3DRS_STENCILREF, 0x1);
    Device->SetRenderState(D3DRS_STENCILMASK, 0xffffffff);
    Device->SetRenderState(D3DRS_STENCILWRITEMASK, 0xffffffff);
    // depth test가 실패하면 pixel이 가려졌음을 의미 렌더링할 필요 없음.
    Device->SetRenderState(D3DRS_STENCILZFAIL, D3DSTENCILOP_KEEP);
    Device->SetRenderState(D3DRS_STENCILFAIL, D3DSTENCILOP_KEEP);
    // depth/stencil test가 성공하면 stencil 참조 값을 0x1로 함.
    Device->SetRenderState(D3DRS_STENCILPASS, D3DSTENCILOP_REPLACE);
    // depth/back buffer에 쓰는 것을 방지함.              스텐실버퍼에 거울 렌더링
    Device->SetRenderState(D3DRS_ZWRITEENABLE, false);
    Device->SetRenderState(D3DRS_ALPHABLENDENABLE, true);
    // blending시 back buffer가 바뀌지 않게 함.
    Device->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_ZERO);
    Device->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_ONE);
```

Example: StencilMirror

```
    // stencil buffer에 거울을 그린다.
    Device->SetStreamSource(0, VB, 0, sizeof(Vertex));
    Device->SetFVF(Vertex::FVF);
    Device->SetMaterial(&MirrorMtrl);
    Device->SetTexture(0, MirrorTex);
    D3DXMATRIX I;
    D3DXMatrixIdentity(&I);
    Device->SetTransform(D3DTS_WORLD, &I);
    Device->DrawPrimitive(D3DPT_TRIANGLELIST, 18, 2);
    // re-enable depth write
    Device->SetRenderState(D3DRS_ZWRITEENABLE, true);
    // 거울에 비친 주전자만 그린다                      스텐실버퍼내에 거울에 해당하는 픽셀은
    // stencil값이 0x1인 경우에만 pass함                 0x1값을 가짐.거울로 렌더링될 부분을 표시
    Device->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_EQUAL);
    // stencil test가 pass이면 stencil buffer값을 계속 유지함
    Device->SetRenderState(D3DRS_STENCILPASS, D3DSTENCILOP_KEEP);
```

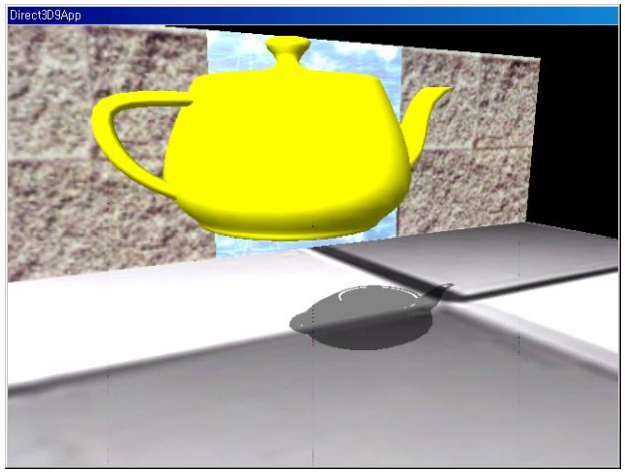
Example: StencilMirror

```
    // position reflection                              주전자의 반사행렬
    D3DXMATRIX W, T, R;
    D3DXPLANE plane(0.0f, 0.0f, 1.0f, 0.0f);   // xy plane
    D3DXMatrixReflect(&R, &plane);
    D3DXMatrixTranslation(&T, TeapotPosition.x, TeapotPosition.y,
                          TeapotPosition.z);
    W = T * R;

    // clear depth buffer and blend reflected teapot with mirror
    // 깊이버퍼 소거                                    반사된 주전자를 거울과 블랜드
    Device->Clear(0, 0, D3DCLEAR_ZBUFFER, 0, 1.0f, 0);
    // 동시에 반사된 주전자를 거울과 블랜드(blending) 해줘야함
    Device->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_DESTCOLOR);
    Device->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_ZERO);
```

Example: StencilMirror

```
    // finally draw the reflected teapot
    Device->SetTransform(D3DTS_WORLD, &W);
    Device->SetMaterial(&TeapotMtrl);
    Device->SetTexture(0, 0);
    // 반사될 때 물체의 전후면이 뒤바뀌나 두르기 순서는 바뀌지않아서
    // 후면 추리기 조건을 변경해야 한다.
    Device->SetRenderState(D3DRS_CULLMODE, D3DCULL_CW);
    Teapot->DrawSubset(0);

    // restore render states
    Device->SetRenderState(D3DRS_ALPHABLENDENABLE, false);
    Device->SetRenderState(D3DRS_STENCILENABLE, false);
    Device->SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);
}
```

Shadow



Shadow

- 평면에 놓이는 그림자
  - 어느 곳에서 빛이 비추고 있는지 인식할 수 있도록 함
  - 장면의 사실감을 높임
- 그림자 볼륨
  - 매우 사실적, 수준 높은 개념
  - DirectX SDK에 그림자 볼륨의 예제 프로그램
- 구현 방법
  1. 3차원 수학 – 물체를 평면에 떨어뜨림
  2. 50% 투명한 검은 재질을 이용하여 렌더링
  3. "더블 블렌딩" 이라는 약간의 부작용 방지를 위해 스텐실 버퍼 이용

Directional Light Shadow

- 광선과 평면의 교차

Ray $\mathbf{r}(t)=\mathbf{p}+t\mathbf{L}$

Plane $\mathbf{n}\cdot\mathbf{p}+d=0$

Intersection Point

$\mathbf{s}=\mathbf{p}+t\mathbf{L}$

$\mathbf{n}\cdot\mathbf{s}+d=0$

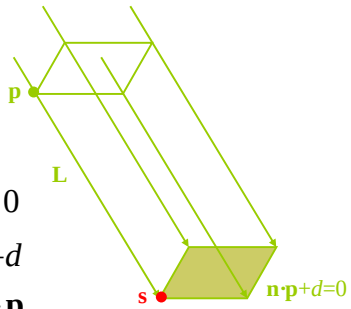
$\Rightarrow \mathbf{n}\cdot(\mathbf{p}+t\mathbf{L})+d=0$

$\mathbf{n}\cdot\mathbf{p}+t(\mathbf{n}\cdot\mathbf{L})=-d$

$t(\mathbf{n}\cdot\mathbf{L})=-d-\mathbf{n}\cdot\mathbf{p}$

$t=\dfrac{-d-\mathbf{n}\cdot\mathbf{p}}{\mathbf{n}\cdot\mathbf{L}}$

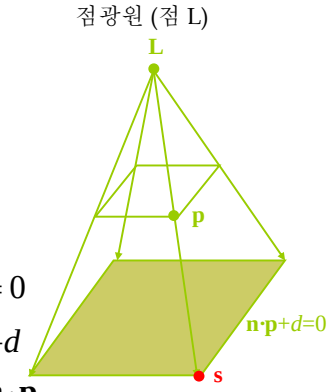
평행광원 (벡터 L)



$\therefore \mathbf{s}=\mathbf{p}+\left[\dfrac{-d-\mathbf{n}\cdot\mathbf{p}}{\mathbf{n}\cdot\mathbf{L}}\right]\mathbf{L}$

Point Light Shadow

- 광선과 평면의 교차



Ray $\mathbf{r}(t)=\mathbf{p}+t(\mathbf{p}-\mathbf{L})$

Plane $\mathbf{n}\cdot\mathbf{p}+d=0$

Intersection Point

$\mathbf{s}=\mathbf{p}+t(\mathbf{p}-\mathbf{L})$

$\mathbf{n}\cdot\mathbf{s}+d=0$

$\Rightarrow \mathbf{n}\cdot(\mathbf{p}+t(\mathbf{p}-\mathbf{L}))+d=0$

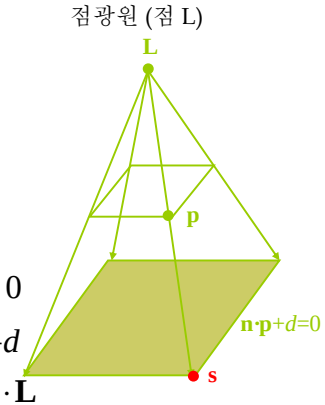
$\mathbf{n}\cdot\mathbf{p}+t(\mathbf{n}\cdot(\mathbf{p}-\mathbf{L}))=-d$

$t(\mathbf{n}\cdot\mathbf{p}-\mathbf{n}\cdot\mathbf{L})=-d-\mathbf{n}\cdot\mathbf{p}$

$t=\dfrac{-d-\mathbf{n}\cdot\mathbf{p}}{\mathbf{n}\cdot\mathbf{p}-\mathbf{n}\cdot\mathbf{L}} \quad \therefore \mathbf{s}=\mathbf{p}+\left[\dfrac{-d-\mathbf{n}\cdot\mathbf{p}}{\mathbf{n}\cdot\mathbf{p}-\mathbf{n}\cdot\mathbf{L}}\right](\mathbf{p}-\mathbf{L})$

Point Light Shadow

- 광선과 평면의 교차



Ray $\mathbf{r}(t)=\mathbf{L}+t(\mathbf{p}-\mathbf{L})$

Plane $\mathbf{n}\cdot\mathbf{p}+d=0$

Intersection Point

$\mathbf{s}=\mathbf{L}+t(\mathbf{p}-\mathbf{L})$

$\mathbf{n}\cdot\mathbf{s}+d=0$

$\Rightarrow \mathbf{n}\cdot(\mathbf{L}+t(\mathbf{p}-\mathbf{L}))+d=0$

$\mathbf{n}\cdot\mathbf{L}+t(\mathbf{n}\cdot(\mathbf{p}-\mathbf{L}))=-d$

$t(\mathbf{n}\cdot\mathbf{p}-\mathbf{n}\cdot\mathbf{L})=-d-\mathbf{n}\cdot\mathbf{L}$

$t=\dfrac{-d-\mathbf{n}\cdot\mathbf{L}}{\mathbf{n}\cdot\mathbf{p}-\mathbf{n}\cdot\mathbf{L}} \quad \therefore \mathbf{s}=\mathbf{L}+\left[\dfrac{-d-\mathbf{n}\cdot\mathbf{L}}{\mathbf{n}\cdot\mathbf{p}-\mathbf{n}\cdot\mathbf{L}}\right](\mathbf{p}-\mathbf{L})$

Shadow Matrix

평행 광원 그림자 → 평행 투영 이용

점 광원 그림자 → 원근 투영 이용

- 변환 행렬
  - 투영면 $n\cdot p + d = 0$ → 4D 벡터 (n_x, n_y, n_z, d)
  - 조명의 방향이나 위치 → 4D vector (L_x, L_y, L_z, L_w)
    - L_w=0 이면, **L** 은 평행 조명의 방향
    - L_w=1 이면, **L** 은 점 조명의 위치

$s = pS \qquad \mathbf{S}=\begin{bmatrix} n_xL_x+k & n_xL_y & n_xL_z & n_xL_w \\ n_yL_x & n_yL_y+k & n_yL_z & n_yL_w \\ n_zL_x & n_zL_y & n_zL_z+k & n_zL_w \\ dL_x & dL_y & dL_z & dL_w+k \end{bmatrix}$

where $k=(n_x,n_y,n_z,d)\cdot(L_x,L_y,L_z,L_w)=n_xL_x+n_yL_y+n_zL_z+dL_w$

Shadow Matrix

- DirectX는 그림자 행렬을 만들어 내는 함수 제공

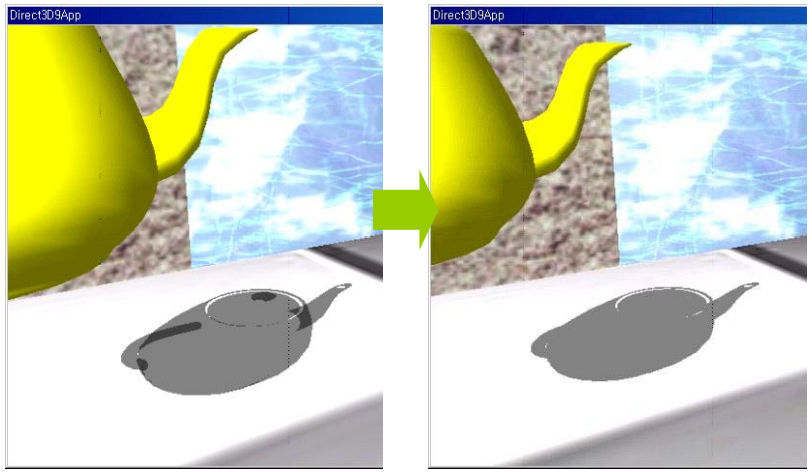
```
D3DXMATRIX *D3DXMatrixShadow(
                D3DXMATRIX *pOut,
                CONST D3DXVECTOR4 *pLight, // light
                CONST D3DXPLANE *pPlane); // shadow plane
```

  - pLight의 w=0일 경우, 평행 조명 (directional light)
    w=1일 경우, 점 조명 (point light)
  - 내부적으로 plane을 정규화한 후, light위치와 평면과의 dot product를 계산한 후, 그림자 행렬 S를 계산함.

```
P = normalize(Plane)
L = light
D = dot(P, L)
P.a *L.x + D    P.a*L.y      P.a*L.z      P.a*L.w
P.b *L.x        P.b*L.y+D    P.b*L.z      P.b*L.w
P.c *L.x        P.c*L.y      P.c*L.z+D    P.c*L.w
P.d *L.x        P.d*L.y      P.d*L.z      P.d*L.w+D
```

Double Blending 해결

- 더블 블렌딩 (Double blending)



Double Blending 해결

- Double Blending
  - 물체의 기하 정보를 평면에 flatten하게 만들면 여러 개의 flatten된 삼각형들이 겹쳐나게 됨.
  - 이들을 반투명으로 그림자 렌더링하면 겹쳐진 영역들이 더욱 어둡게 나타남.
- 해결방법
  - Stencil buffer를 사용하여 처음으로 렌더링되는 픽셀들만 받아들이도록 스텐실 테스트를 구성함.
  - 그림자 pixel이 후면버퍼에 렌더링 되는 동안 스텐실 버퍼에 표시를 남겨 놓음.
  - 이미 렌더링된 (스텐실 버퍼에 표시된) 그림자 영역에 다시 픽셀을 쓰려고 하면, 스텐실 테스트가 실패하므로 blending되지 않음.

Example: StencilShadow

```
bool Display(float timeDelta) {
    ….
    RenderScene();
    RenderShadow();
    …
}
void RenderShadow()
{
    // stencil buffer는 0으로 clear 됐다고 가정
    Device->SetRenderState(D3DRS_STENCILENABLE, true);
    // stencil buffer의 해당값이 0인 경우에만 back buffer에 그림자를 렌더링
    Device->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_EQUAL);
    Device->SetRenderState(D3DRS_STENCILREF, 0x0);
    Device->SetRenderState(D3DRS_STENCILMASK, 0xffffffff);
    Device->SetRenderState(D3DRS_STENCILWRITEMASK, 0xffffffff);
    Device->SetRenderState(D3DRS_STENCILZFAIL, D3DSTENCILOP_KEEP);
    Device->SetRenderState(D3DRS_STENCILFAIL, D3DSTENCILOP_KEEP);
    // 두 번째 이후 실패하도록 하기 위해서 1을 증가시킴
    Device->SetRenderState(D3DRS_STENCILPASS, D3DSTENCILOP_INCR);
```

Example: StencilShadow

```
    // 주전자의 그림자 행렬
    D3DVECTOR4 lightDirection(0.707f, -0.707f, 0.707f, 0.0f);
    D3DPLANE groundPlane(0.0f, -1.0f, 0.0f, 0.0f);
    D3DXMATRIX S;
    D3DXMatrixShadow(&S, &lightDirection, &groundPlane);
    D3DXMATRIX T;
    D3DXMatrixTranslation(&T, TeapotPosition.x, TeapotPosition.y,
                                  TeapotPosition.z);
    D3DXMATRIX W = T * S;
    Device->SetTransform(D3DTS_WORLD, &W);

    // alpha blend the shadow
    Device->SetRenderState(D3DRS_ALPHABLENDENABLE, true);
    Device->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
    Device->SetRenderState(D3DRS_DESTBLEN, D3DBLEND_INVSRCALPHA);
```

Example: StencilShadow

```
    // 그림자를 50%투명도의 검은 재질로 지정
    D3DMATERIAL9 mtrl = d3d::InitMtrl(d3d::BLACK, d3d::BLACK,
                                      d3d::BLACK, d3d::BLACK, 0.0f);
    mtrl.Diffuse.a = 0.5f;     // 50% transparency
    // 그림자를 렌더링할 때 바닥면과 z-buffer fighting하지 않도록 z-buffer를 끈다.
    Device->SetRenderState(D3DRS_ZENABLE, false);
    Device->SetMaterial(&mtrl);
    Device->SetTexture(0, 0);
    Teapot->DrawSubset(0);

    Device->SetRenderState(D3DRS_ZENABLE, true);
    Device->SetRenderState(D3DRS_ALPHABLENDENABLE, false);
    Device->SetRenderState(D3DRS_STENCILENABLE, false);
}
```

Reference

- http://www.opengl.org/resources/features/StencilTalk/