Stencil

305890
Spring 2010
5/7/2010
Kyoung Shin Park

Overview

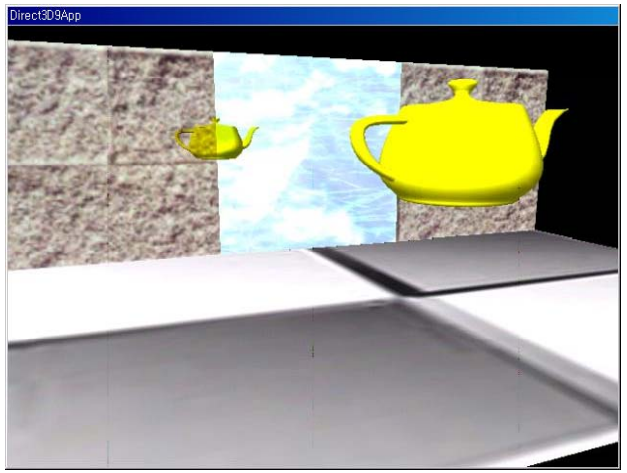
- Using the Stencil Buffer
- Mirrors
  - Implementing mirror using a stencil buffer
- Planar Shadows
  - Preventing double blending using a stencil buffer

Stencil Buffer

- Stencil Buffer
  - Stencil buffer is an off-screen buffer to achieve special effects
  - Stencil buffer has a same resolution as back buffer and depth buffer
  - Stencil buffer works as a stencil and allows us to block rendering to certain parts of the back buffer
  - Used for mirrors and shadows
    - For example, when implementing a mirror we simply need to reflect a particular object across the plane of the mirror; however, we only want to draw the reflection into a mirror.

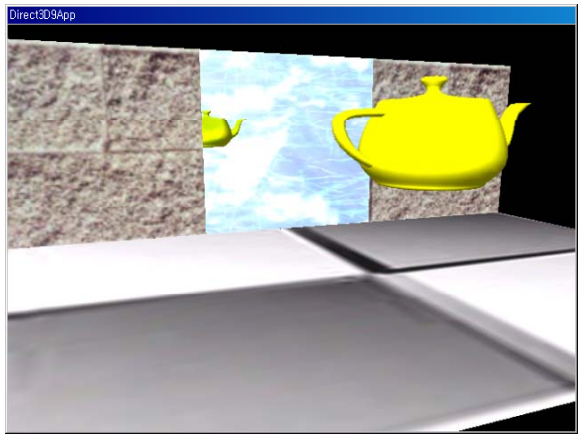
Mirror Effect

- Teapot being reflected without using the stencil buffer



Mirror Effect

- Block the reflected teapot from being rendered unless it is being drawn in the mirror, by using stencil buffer



Using Stencil Buffer

- Using a stencil buffer
  - Request a stencil buffer at the time we create the depth buffer
  - Enable the stencil buffer

```
_device->SetRenderState(D3DRS_STENCILENABLE, true);
... // do stencil work
_device->SetRenderState(D3DRS_STENCILENABLE, false);
```

- Clear a stencil buffer (same as back buffer & depth buffer)

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

Using Stencil Buffer

- Requesting a stencil buffer
  - A stencil buffer can be created at the time we create the depth buffer
  - Specify the format of the stencil buffer and depth buffer
- Depth/Stencil buffer format
  - D3DFMT_D24S8: create a 32-bit depth/stencil buffer (24-bit depth buffer/8-bit stencil buffer per pixel)
  - D3DFMT_D24X4S4: 32-bit depth/stencil buffer, (24-bit depth buffer/4-bit stencil buffer per pixel, 4-bit will not be used)
  - D3DFMT_D15S1: 16-bit depth/stencil buffer (15-bit depth buffer/1-bit stencil buffer per pixel)
  - D3DFMT_D32: 32-bit depth buffer (no stencil buffer)
  - The support of stenciling varies among the various graphics cards. (Some cards may not support an 8-bit stencil buffer)

Stencil Test

- Stencil test
  - We can use the stencil buffer to block rendering to certain areas of the back buffer. Decide to block a particular pixel from being written is decided by the stencil test.
  - (StencilRef & StencilMask) CompFunc (StencilBufferValue & StencilMask)
    - StencilRef: stencil reference value set with D3DRS_STENCILREF render state (0 by default).
    - StencilMask: stencil mask value to mask bit in both the StencilRef and StencilBufferValue variables (0xffffffff by default)
    - StencilBufferValue: stencil buffer value for the current pixel we are stencil testing
    - **IF (StencilRef & StencilMask) CompFunc (StencilBufferValue & StencilMask) == true THEN accept pixel**
      ELSE reject pixel

Stencil Test Control

- Set a comparison operation

  _device->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_LESS);

- D3DCMPFUNC enum type:
  - D3DCMP_NEVER: stencil test always fails
  - D3DCMP_LESS/EQUAL/LESSQUAL/GREATER/NOTEQUAL/GREATEREQUAL: lhs < / = / <= / > / != / >= rhs
  - D3DCMP_ALWAYS: stencil test always succeeds (the pixel is always drawn) – default
  - D3DCMP_FORCE_DWORD: Not used, but included to make enumerated type instance 32 bits

Stencil Test Control

- Stencil test control:

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

Mask 16 high bits

Stencil Buffer Update

- Updating the stencil buffer after stencil test
  - The stencil test fails for the ijth pixel, we define how to update the ijth entry in the stencil buffer:

    _device->SetRenderState(D3DRS_STENCILFAIL, StencilOperation);
  - The depth test fails for the ijth pixel, we define how to update the ijth entry in the stencil buffer:

    _device->SetRenderState(D3DRS_STENCILZFAIL, StencilOperation);
  - The stencil test and stencil test succeed for the ijth pixel, we define how to update the ijth entry in the stencil buffer:

    _device->SetRenderState(D3DRS_STENCILPASS, StencilOperation);

D3DSTENCILOP_KEEP

Stencil Buffer Update

- StencilOperation can be one of the following:
  - D3DSTENCILOP_KEEP: specifies to not change the stencil buffer; i.e., keep the value currently there
  - D3DSTENCILOP_ZERO/REPLACE/INVERT: specifies to set the stencil buffer entry to 0, replace it with StencilRef, or invert the bits of the stencil buffer
  - D3DSTENCILOP_INCRSAT/DECRSAT/INCR/DECR: specifies to increment the stencil buffer entry (clamp the entry to that maximum), decrement (clamp the entry to 0), increment (wrap to 0), decrement (wrap to the maximum allowed value)

Stencil Write Mask

- Stencil Write Mask
  - We can set a write mask that masks off bits of any value we write to the stencil buffer (default value is 0xffffffff)

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

StencilMirror Demo

- StencilMirror Demo
  - Reflection matrix
  - Using stencil buffer to draw reflection on the mirror surface

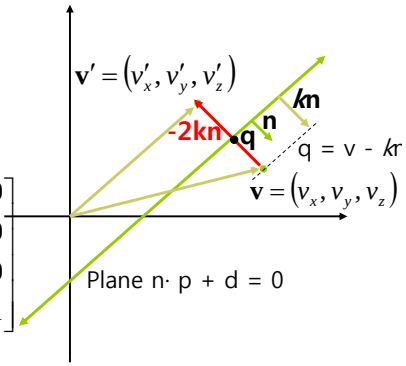
Reflection

- Compute a reflection point $v'=(v_x', v_y', v_z')$ of a point $v=(v_x, v_y, v_z)$ about an arbitrary plane (n, d)

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

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

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



k: the signed shortest distance from v to the plane
k = n•v + d when n=unit vector

Reflection

- D3DX library provides the reflection matrix function

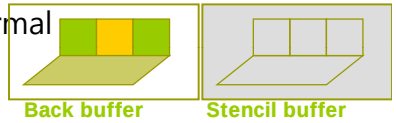
```
D3DXMATRIX *D3DXMatrixReflect(D3DXMATRIX *pOut,
                              CONST D3DXPLANE *pPlane);
```

- Reflections about the 3 standard coordinate planes – the *yz-plane*, *xz-plane*, *xy-plane*

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

Example: StencilMirror Demo

1. Render the entire scene as normal
   - But, not the teapot's reflection



2. Clear the stencil buffer to 0
3. Render the primitives that make up the mirror into the stencil buffer only.
   - Set the stencil test to always succeed and specify that the stencil buffer entry should be replaced with 1 if the test passes. (i.e., only pixels on the mirror surface has 1)

4. Render the reflected teapot to the back buffer and stencil buffer.
   - We only will render to the back buffer if the stencil test passes. (we set the stencil test to only succeed if the value in the stencil buffer is a 1). Then, the teapot will only be rendered to areas that have a 1 in their corresponding stencil buffer entry.

스텐실버퍼에 거울 렌더링

Example: StencilMirror

```
void MirrorDemo::drawReflectedTeapot() {
    gd3dDevice->SetRenderState(D3DRS_STENCILENABLE, true);
    // stencil test가 항상 성공하도록 함
    gd3dDevice->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_ALWAYS);
    gd3dDevice->SetRenderState(D3DRS_STENCILREF, 0x1);
    gd3dDevice->SetRenderState(D3DRS_STENCILMASK, 0xffffffff);
    gd3dDevice->SetRenderState(D3DRS_STENCILWRITEMASK, 0xffffffff);
    // depth test가 실패하면 pixel이 가려졌음을 의미 렌더링할 필요 없음.
    gd3dDevice->SetRenderState(D3DRS_STENCILZFAIL, D3DSTENCILOP_KEEP);
    gd3dDevice->SetRenderState(D3DRS_STENCILFAIL, D3DSTENCILOP_KEEP);
    // depth/stencil test가 성공하면 stencil 참조 값을 0x1로 함.
    gd3dDevice->SetRenderState(D3DRS_STENCILPASS, D3DSTENCILOP_REPLACE);
    // depth/back buffer에 쓰는 것을 방지함.
    gd3dDevice->SetRenderState(D3DRS_ZWRITEENABLE, false);
    // blending시 back buffer가 바뀌지 않게 함.
    gd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, true);
    gd3dDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_ZERO);
    gd3dDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_ONE);
```

Example: StencilMirror

```
// draw mirror to stencil only
drawMirror();
// re-enable depth writes
gd3dDevice->SetRenderState(D3DRS_ZWRITEENABLE, true);
// stencil값이 0x1인 경우에만 pass함
gd3dDevice->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_EQUAL);
// stencil test가 pass이면 stencil buffer값을 계속 유지함
gd3dDevice->SetRenderState(D3DRS_STENCILPASS, D3DSTENCILOP_KEEP);
// Build reflection matrix
D3DXMATRIX R;
D3DXPLANE plane(0.0f, 0.0f, 1.0f, 0.0f); // xy-plane
D3DXMatrixReflect(&R, &plane);
// Set the original teapot world matrix
D3DXMATRIX oldTeapotWorld = mTeapotWorld;
// Add reflection transformation
mTeapotWorld = mTeapotWorld * R;
```

스텐실버퍼내에 거울에 해당하는 픽셀은 0x1값을 가짐.거울로 렌더링될 부분을 표시

주전자의 반사행렬

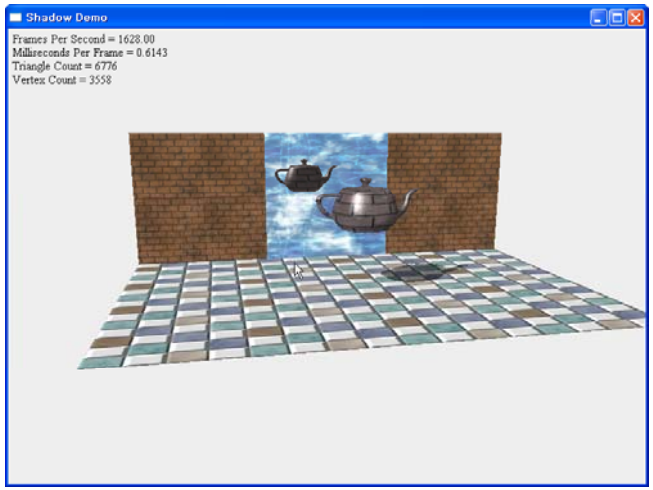
Example: StencilMirror

```
    // reflect light vector also
    D3DXVECTOR3 oldLightVecW = mLightVecW;
    D3DXVec3TransformNormal(&mLightVecW, &mLightVecW, &R);
    mFX->SetValue(mhLightVecW, &mLightVecW, sizeof(D3DXVECTOR3));
    // disable depth buffer and render the reflected teapot
    gd3dDevice->SetRenderState(D3DRS_ZENABLE, false);
    gd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, false);
    // 반사될 때 물체의 전후면이 뒤바뀌나 winding order는 바뀌지 않아서
    //culling을 변경해야 하고, 반사된 주전자를 렌더링해야 한다.
    gd3dDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_CW);
    drawTeapot();
    // restore original teapot world matrix and light vector
    mTeapotWorld = oldTeapotWorld;
    mLightVecW = oldLightVecW;
    // restore render states
    gd3dDevice->SetRenderState(D3DRS_ZENABLE, true);
    gd3dDevice->SetRenderState(D3DRS_STENCILENABLE, false);
    gd3dDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);
}
```

Example: StencilMirror

```
void MirrorDemo::drawScene() {
    gd3dDevice->Clear(0, 0, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER|
                        D3DCLEAR_STENCIL, 0xffeeeeee, 1.0f, 0);
    mFX->SetTechqnie(mhTech);
    mFX->SetValue(mhLightVecW, &mLightVecW, sizeof(D3DXVECTOR3));
    mFX->SetValue(mhDiffuseLight, &mDiffuseLight, sizeof(D3DXCOLOR));
    mFX->SetValue(mhAmbientLight, &mAmbientLight, sizeof(D3DXCOLOR));
    mFX->SetValue(mhSpecularLight, &mSpecularLight, sizeof(D3DXCOLOR));
    // 중간 생략..
    drawRoom();
    drawMirror();
    drawTeapot();
    drawReflectedTeapot();
    gd3dDevice->EndScene();
    gd3dDevice->Present(0, 0, 0, 0);
}
```

Planar Shadow



Shadow

- Shadow
  - Shadows aid in our perception of where light is being emitted in a scene;
  - Ultimately make the scene more realistic
- Planar shadow implementation
  1. Find the shadow an object casts to a plane
  2. Then, render the polygons that describe the shadow with a black material at 50% transparency
  3. Employ the stencil buffer to prevent "double blending" while rendering the shadow

Directional Light Shadow

- Ray and plane intersection

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

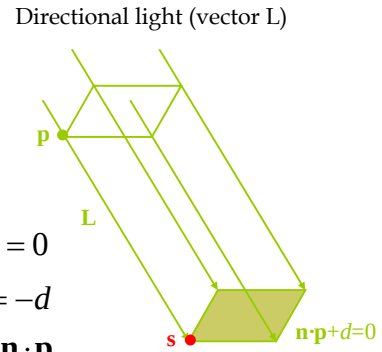
Plane $\mathbf{n}\cdot\mathbf{p}+d=0$

Intersection Point

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

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

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



Point Light Shadow

- Ray and plane intersection

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

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

Intersection Point

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

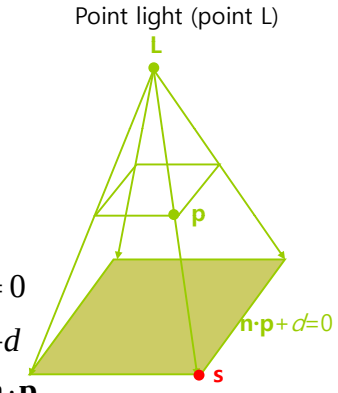
$\mathbf{n}\cdot\mathbf{s}+d=0$

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

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

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

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



Point Light Shadow

- Ray and plane intersection

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

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

Intersection Point

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

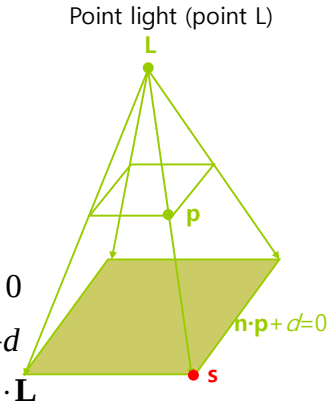
$\mathbf{n}\cdot\mathbf{s}+d=0$

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

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

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

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



Shadow Matrix

- Shadow matrix
  - Plane: n· p + d = 0 → 4D vector (n_x, n_y, n_z, d)
  - Light source vector/point → 4D vector (L_x, L_y, L_z, L_w)
    - *If* L_w=0, **L** is a directional light source
    - *If* L_w=1, **L** is a point light source

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

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

- D3DX library provides the shadow matrix function

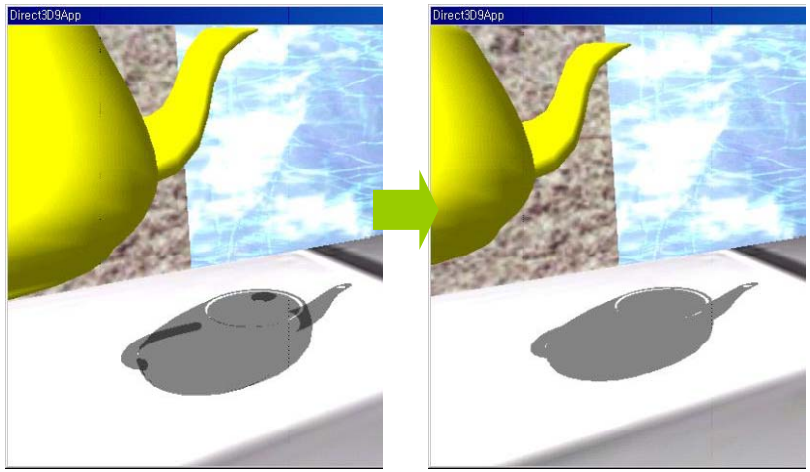
```
D3DXMATRIX *D3DXMatrixShadow(D3DXMATRIX *pOut,
                             CONST D3DXVECTOR4 *pLight, // light
                             CONST D3DXPLANE *pPlane); // shadow plane
```

  - pLight *w*=0, directional light
    w=1, point light
  - This function normalize plane, and then compute the dot product of light and plane, and then compute the shadow matrix.

```
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

- Double blending



Double Blending

- Double blending
  - When we flatten out the geometry of an object onto the plane to describe its shadow, it is possible that two or more of the flattened triangles will overlap.
  - When we render the shadow with transparency (using blending), these areas that have overlapping triangles will get blended multiple times and thus appear darker.
- Preventing double blending artifacts
  - Using the stencil buffer, we set the stencil test to only accept pixels the first time they are rendered.
  - As we render the shadow's pixel to the back buffer, we will mark the corresponding stencil buffer entries.
  - Then, if we attempt to write a pixel to an area that has already been rendered to (marked in the stencil buffer), stencil test fails

Example: StencilShadow

```
void ShadowDemo::drawTeapotShadow() {
    // stencil buffer는 0으로 clear 됐다고 가정
    gd3dDevice->SetRenderState(D3DRS_STENCILENABLE, true);
    // stencil buffer의 해당값이 0인 경우에만 back buffer에 그림자를 렌더링
    gd3dDevice->SetRenderState(D3DRS_STENCILFUNC, D3DCMP_EQUAL);
    gd3dDevice->SetRenderState(D3DRS_STENCILREF, 0x0);
    gd3dDevice->SetRenderState(D3DRS_STENCILMASK, 0xffffffff);
    gd3dDevice->SetRenderState(D3DRS_STENCILWRITEMASK, 0xffffffff);
    gd3dDevice->SetRenderState(D3DRS_STENCILZFAIL, D3DSTENCILOP_KEEP);
    gd3dDevice->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);
    // offset shadow up slightly for no z-fighting with shadow and ground
    D3DXMATRIX eps;
    D3DXMatrixTranslation(&eps, 0.0f, 0.001f, 0.0f);
    // save the original teapot world matrix
    D3DXMATRIX oldTeapotWorld = mTeapotWorld;
    // add shadow projection transform
    mTeapotWorld = mTeapotWorld * S * eps;
    // alpha blend the shadow
    gd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, true);
    gd3dDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
    gd3dDevice->SetRenderState(D3DRS_DESTBLEN, D3DBLEND_INVSRCALPHA);
```

Example: StencilShadow

```
    // 그림자를 50%투명도의 검은 재질로 지정하여 렌더링
    drawTeapot();
    // restore settings
    mTeapotWorld = oldTeapotWorld;
    gd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, false);
    gd3dDevice->SetRenderState(D3DRS_STENCILENABLE, false);
}
```

Reference

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