Stencil

305890
Spring 2014
5/27/2014
Kyoung Shin Park

Overview

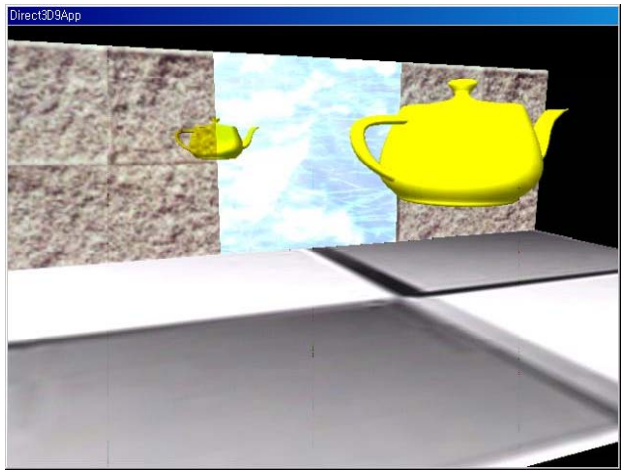
- Using the Stencil Buffer
- Mirrors
  - Implementing mirror using a stencil buffer
- Planar Shadows
  - Preventing double blending using a stencil buffer

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.

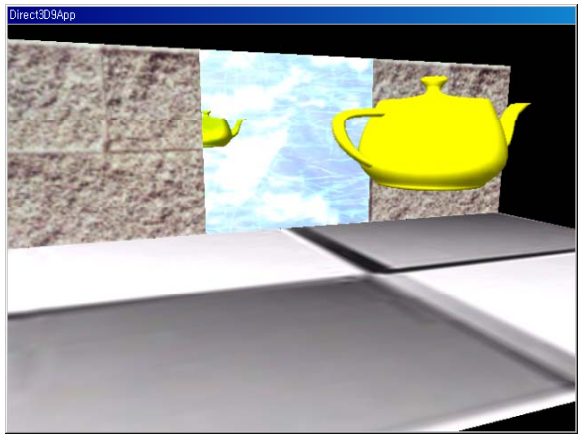
Mirror Effect

- 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

```
DepthStencilState dss = new DepthStencilState();
dss.StencilEnable = true;
... // do stencil work
```

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

```
GraphicsDevice.Clear(
    ClearOptions.Target|ClearOptions.DepthBuffer|ClearOptions.Stencil,
    Color.CornflowerBlue /* 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
  - DepthFormat.Depth24Stencil8: create a 32-bit depth/stencil buffer (24-bit depth buffer/8-bit stencil buffer per pixel)
  - DepthFormat.Depth24 : 24-bit depth buffer
  - DepthFormat.Depth16 : 16-bit depth buffer
  - DepthFormat.None : Do not create a depth buffer

```
graphics.PreferredDepthStencilFormat = DepthFormat.Depth24Stencil8;
```

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
  dss.StencilFunction = CompareFunction.Always;
- CompareFunction enum type:
  - Always: stencil test always succeeds (the pixel is always drawn)
  - Equal/Greater/GreaterEqual/Less/LessEqual/NotEqual: lhs =, >, >=, <, <=, != rhs
  - Never: stencil test always fails

Stencil Test Control

- Stencil test control:

```
// enable stencil test
DepthStencilState checkMirror = new DepthStencilState();
{
StencilEnable = true;
// specify the stencil comparison function
StencilFunction = CompareFunction.Equal;
// set the comparison reference value
ReferenceStencil = 1;
// set the stencil operation to perform if the stencil test passes
StencilPass = StencilOperation.Keep;
}
```

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:
    StencilFail = StencilOperation.Keep;
  - The stencil test and stencil test succeed for the ijth pixel, we define how to update the ijth entry in the stencil buffer:
    StencilPass = StencilOperation.Keep;

Stencil Buffer Update

- StencilOperation can be one of the following:
  StencilOperation.Decrement : decrements the stencil buffer entry
  StencilOperation.DecrementSaturation : decrements & clamps to 0
  StencilOperation.Increment : increments the stencil buffer entry
  StencilOperation.IncrementSaturation : increments & clamps to max
  StencilOperation.Invert : inverts the bits in stencil buffer entry
  StencilOperation.Keep : do not update the stencil buffer entry
  StencilOperation.Replace : replace the stencil buffer entry with reference value
  StencilOperation.Zero : sets the stencil buffer entry to 0

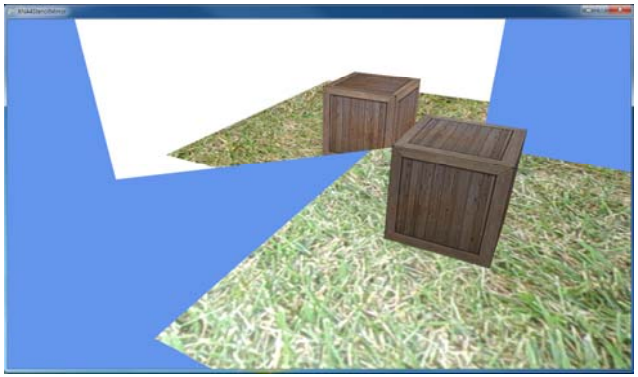
Stencil Write Mask

- Stencil Write Mask
  - We can set a write mask that masks off bits of any value we write to the stencil buffer

```
StencilWriteMask = 0;
```

StencilMirror Demo

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



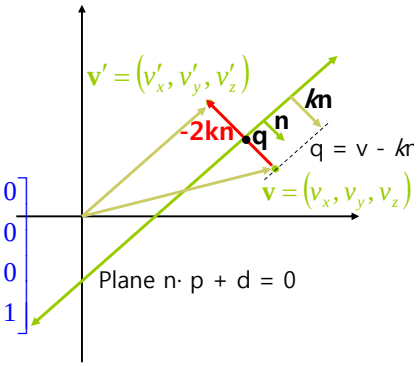
Reflection

- 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)

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

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

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



k: 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

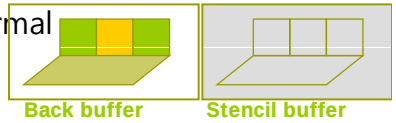
```
D3DXMATRIX *D3DXMatrixReflect(D3DXMATRIX *pOut,
                              CONST D3DXPLANE *pPlane);
```

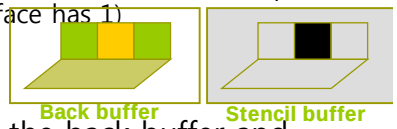
- Reflections about the 3 standard coordinate planes – the y*z-plane*, *xz-plane*, *xy-plane*

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

Example: StencilMirror Demo

1. Render the entire scene as normal
   - But, not the box 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.

Example: StencilMirror

```
DepthStencilState addIfMirror = new DepthStencilState()
{
    StencilEnable = true,
    StencilFunction = CompareFunction.Always,
    StencilPass = StencilOperation.Increment
};

DepthStencilState checkMirror = new DepthStencilState()
{
    StencilEnable = true,
    StencilFunction = CompareFunction.Equal,
    ReferenceStencil = 1,
    StencilPass = StencilOperation.Keep
};
```

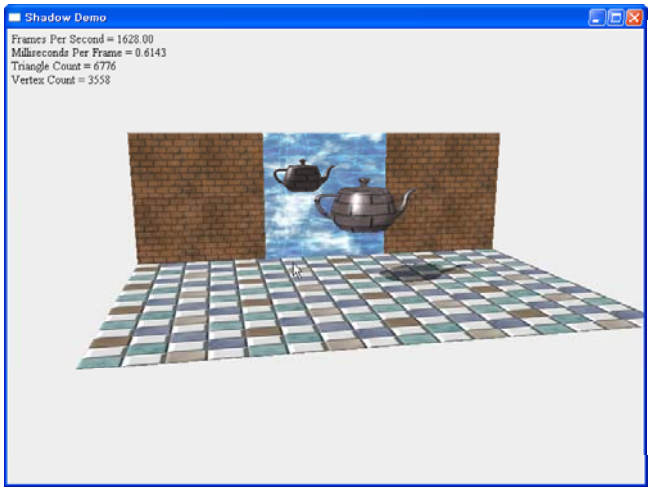

Example: StencilMirror

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(ClearOptions.Target | ClearOptions.DepthBuffer |
    ClearOptions.Stencil,Color.CornflowerBlue,1,0);
    GraphicsDevice.DepthStencilState = DepthStencilState.Default;
    GraphicsDevice.RasterizerState = RasterizerState.CullCounterClockwise;

    //draw the crate & the ground normally
    myBox.Draw(World, View, Projection, crateTexture);
    grid.Draw(Matrix.Identity, View, Projection, grassTexture);

    //draw the mirror, remember we are drawing to both color and stencil buffer
    colorEffect.Parameters["WVP"].SetValue(Matrix.Identity * View * Projection);
    myMirror.Draw(colorEffect);
   //set the stencil buffer to check if we are drawing on the surface of the mirror
    GraphicsDevice.DepthStencilState = addIfMirror;
    myMirror.Draw(colorEffect);
```


Example: StencilMirror

```
    //set stencil for draw on mirror surface only
    GraphicsDevice.DepthStencilState = checkMirror;
    GraphicsDevice.Clear(ClearOptions.DepthBuffer, Color.CornflowerBlue, 1, 0);
    GraphicsDevice.RasterizerState = RasterizerState.CullClockwise;
    //draw the crate in the mirror
    textureEffect.Parameters["tex"].SetValue(crateTexture);
    textureEffect.Parameters["WVP"].SetValue(World * Reflect * View * Projection);
    myBox.Draw(textureEffect);
    //draw the ground in the mirror
    textureEffect.Parameters["tex"].SetValue(grassTexture);
    textureEffect.Parameters["WVP"].SetValue(Reflect * View * Projection);
    grid.Draw(textureEffect);

    base.Draw(gameTime);
}
```

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

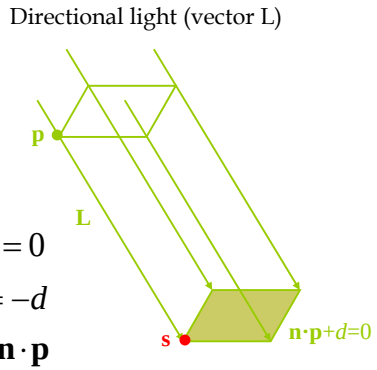
Plane $\mathbf{n}\cdot\mathbf{p}+d=0$

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

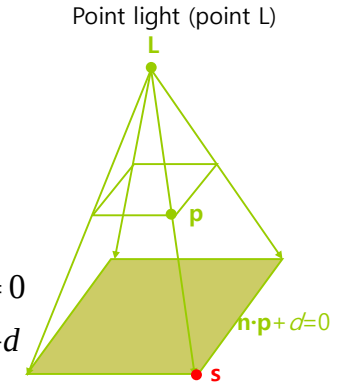
Plane $\mathbf{n}\cdot\mathbf{p}+d=0$

Intersection Point

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

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

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



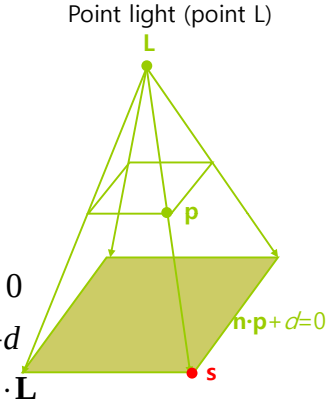
Point Light Shadow

- 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



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

$$\mathbf{n}\cdot(\mathbf{L}+t(\mathbf{p}-\mathbf{L}))+d=0$$
$$\mathbf{n}\cdot\mathbf{L}+t(\mathbf{n}\cdot(\mathbf{p}-\mathbf{L}))=-d$$
$$t(\mathbf{n}\cdot\mathbf{p}-\mathbf{n}\cdot\mathbf{L})=-d-\mathbf{n}\cdot\mathbf{L}$$
$$t=\frac{-d-\mathbf{n}\cdot\mathbf{L}}{\mathbf{n}\cdot\mathbf{p}-\mathbf{n}\cdot\mathbf{L}} \quad \therefore\ \mathbf{s}=\mathbf{L}+\left[\frac{-d-\mathbf{n}\cdot\mathbf{L}}{\mathbf{n}\cdot\mathbf{p}-\mathbf{n}\cdot\mathbf{L}}\right](\mathbf{p}-\mathbf{L})$$

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

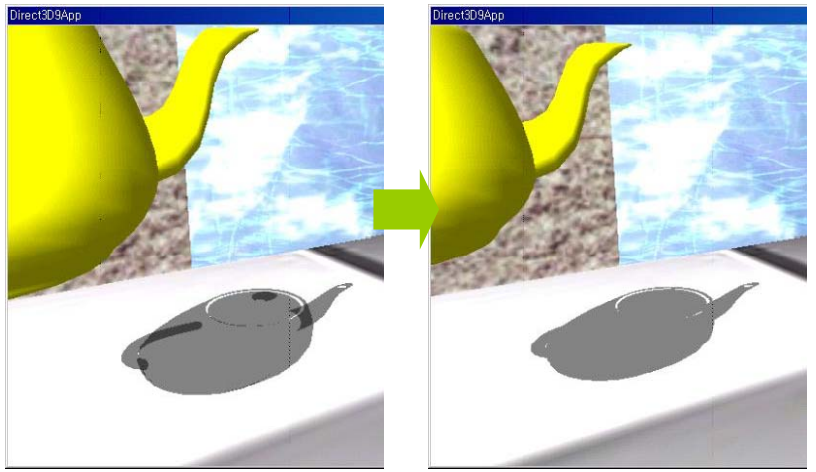
```
D3DXMATRIX *D3DXMatrixShadow(D3DXMATRIX *pOut,
                             CONST D3DXVECTOR4 *pLight, // light
                             CONST D3DXPLANE *pPlane); // shadow plane
```

  - 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

Reference

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