Sombras proyectadas

Autor: Diego G. Ruiz

Versión: 1.0 (draft)

Tabla de Contenidos

Sombras proyectadas

Introducción

Por el modo en que funciona la iluminación en Direct3D, sabemos que no se generan sombras. Los vértices son iluminados independientemente en función de su normal y en función del material activo y las luces activas; pero nada sabe Direct3D de oclusión de luz a un objeto por parte de otro, por ello si deseamos que nuestras escenas posean sombras deberemos generarlas nosotros matemáticamente.

El primer método que veremos se denominado sombras proyectadas, es muy sencillo pero tiene algunas contras. En principio la sombra será generada dibujando nuevamente el modelo proyectado sobre un plano, por lo tanto ya podemos estipular que:

La sombra no es volumétrica

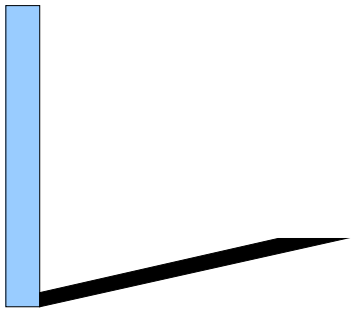

Figura 1. La sombra de un objeto

En la figura 1 se puede apreciar la sombra de un objeto proyectada sobre el plano XZ, dicha sombra no es interrumpida por ningún otro objeto por lo tanto es correcta.

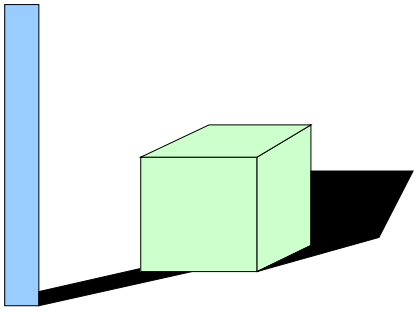

Figura 2. La sombra del primer objeto no se proyecta sobre el segundo

En la figura 2, se puede observar el problema que ocasiona el método de creación de sombras que estamos introduciendo en este documento. La sombra del primer objeto debería proyectarse sobre el segundo como muestra la siguiente figura, pero no lo hace debido a que es chata y se encuentra proyectada sobre un plano pre-especificado.

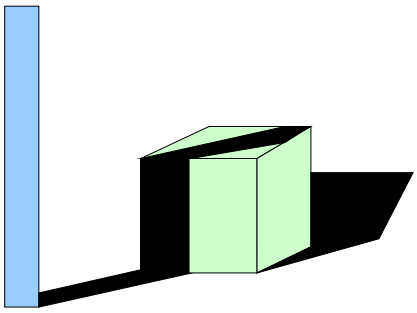

Figura 3. La sombra de un objeto se proyecta sobre otro

Sombras muy delimitadas

La segunda deficiencia del sistema que aprenderemos a calcular en este documento, tiene que ver con el contraste en los bordes de la sombra, son muy marcados. Un efecto más realista tendría que suavisar dichos bordes.

El método

El método de sombras proyectadas se basa en la proyección de un objeto sobre un plano, para ello lo que haremos será construir una matriz que proyecte vértices sobre el plano deseado y luego reprocesaremos el objeto en cuestión introduciendo dicha matriz al pipeline.

Para construir dicha matriz tendremos en cuenta dos elementos:

- El plano de proyección (usualmene XZ)
- La posición de la fuente de iluminación.

Paso 1

Deberemos realizar el producto punto entre el vector dirección, formado entre la posición de la luz y un punto del plano, y el vector normal al plano.

Siendo **luzpos** la posición de la luz, **planpto** un punto del plano, **plannorm** el vector normal al plano (normalizado).

```
vecDir = luzpos - planpto
fDot = vecDir . plannorm
```

Paso 2

Haciendo calculado el producto punto, construiremos la matriz de "sombra" (proyección sobre un plano) del siguiente modo:

fDot - luzpos.x - plano.a	0 - luzpos.y - plano.a	0 - luzpos.z - plano.a	0 - plano.a
0 - luzpos.x - plano.b	fDot - luzpos.y - plano.b	0 - luzpos.z - plano.b	0 - plano.b
0 - luzpos.x - plano.c	0 - luzpos.y - plano.c	fDot - luzpos.z - plano.c	0 - plano.c
0 - luzpos.x - plano.d	0 - luzpos.y - plano.d	0 - luzpos.z - plano.d	fDot - plano.d

Paso 3

Ahora deberemos introducir la matriz de proyección de sombra en el pipeline. Para esto, podríamos multiplicar la matriz de mundo utilizada con la calculada.

Paso 4

Sólo falta algo para mandar a procesar los vértices de nuestro objeto, si lo hacemos ahora se dibujará el objeto proyectado pero mostrando la textura relacionada a él, lo que deseamos aquí es que el objeto se dibuje en negro. Para esto podremos modificar el factor de modulación de la textura, más concretamente cambiarlo a negro.

El código

Creación de la matriz de sombra:

```
void CreateShadowMatrix(Plane * pPlane, Vector3 * pLightPos, Matrix * pMatOut)
{

	// Realizo el producto punto entre la posición de la luz y
	// el plano
	float fDot = D3DXPlaneDotCoord(pPlane, pLightPos);

	// Construyo matriz
	// Columna 1
	pMatOut->_11 = fDot - pLightPos->x * pPlane->a;
	pMatOut->_21 = 0.0f - pLightPos->x * pPlane->b;
	pMatOut->_31 = 0.0f - pLightPos->x * pPlane->c;
	pMatOut->_41 = 0.0f - pLightPos->x * pPlane->d;

	// Columna 2
```

```
        pMatOut->_12 = 0.0f - pLightPos->y * pPlane->a;
        pMatOut->_22 = fDot - pLightPos->y * pPlane->b;
        pMatOut->_32 = 0.0f - pLightPos->y * pPlane->c;
        pMatOut->_42 = 0.0f - pLightPos->y * pPlane->d;

        // Columna 3
        pMatOut->_13 = 0.0f - pLightPos->z * pPlane->a;
        pMatOut->_23 = 0.0f - pLightPos->z * pPlane->b;
        pMatOut->_33 = fDot - pLightPos->z * pPlane->c;
        pMatOut->_43 = 0.0f - pLightPos->z * pPlane->d;

        // Columna 4
        pMatOut->_14 = 0.0f - pPlane->a;
        pMatOut->_24 = 0.0f - pPlane->b;
        pMatOut->_34 = 0.0f - pPlane->c;
        pMatOut->_44 = fDot - pPlane->d;
}
```

Modificación del factor de modulación de la textura:

```
// Especico modulación como operación: textura x factor
m_pD3DDevice->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_MODULATE);
m_pD3DDevice->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
m_pD3DDevice->SetTextureStageState(0, D3DTSS_COLORARG2, D3DTA_TFACTOR);

// Cambio el factor a color negro
m_pD3DDevice->SetRenderState(D3DRS_TEXTUREFACTOR, 0x00000000);
```

Conclusiones

El método de creación de sombras visto es sencillo, posee algunas desventajas pero aún así es mucho más realista que no colocar nada o colocar un círculo negro bajo nuestro personaje. Es apto para gran cantidad de juegos, como juegos de deportes donde no existen objetos que corten el camino de la sombra al plano, que es el piso. Juegos donde tal vez se aprecien más sus debilidades son los que manipulan la luz como método fundamental para lograr efectos (ej: juegos tipo Resident Evil, Splinter Cell, etc.).

Una posible mejora consiste en dibujar la sombra mediante un stencil buffer; para más información leer el documento "Stencil Buffer".