Combined ID y Depth Shadows

Mariano M. Banquiero, Martin H. Giachetti, Matias N. Leone, Leandro R. Barbagallo
Andres Bursztyn

Proyecto de Investigación "Explotación de GPUs y Gráficos Por Computadora", GIGC - Grupo de Investigación de Gráficos por Computadora, Departamento de Ingeniería en Sistemas de Información, UTN-FRBA, Argentina

{mbanquiero, mgiachetti, mleone, lbarbagallo}@frba.utn.edu.ar
andresb@sistemas.frba.utn.edu.ar

Resumen. Presentamos una técnica para minimizar el artifact del self shadow y surface acne característicos del shadow map, basada en un mapa que combina la profundidad con un ID para cada objeto en la escena. La técnica está diseñada para aplicarse en el contexto de una solución para sombras perceptualmente correctas en tiempos casi interactivos.

Palabras Clave: Real-Time Rendering, Shadow Maps, Depth Shadows, GPU, Shaders, Surface Acne, Depth Bias, Combined ID

1 Introducción

La generación de sombras es un efecto visual de suma importancia en la creación de imágenes por computadora. El cerebro humano es particularmente sensible a la captación de sombras. Aparte de agregar realismo a una escena, las sombras, proveen información espacial entre superficies y objetos y ayudan a transmitir la sensación de profundidad. Existen diversas técnicas para la generación de sombras que se pueden aplicar en un rasterizador y bien soportadas por el hardware de las GPUs, la mayor parte de las cuales están orientadas a ser lo suficientemente rápidas para funcionar en un esquema de real-time. Por el contrario, este trabajo forma parte de un conjunto de soluciones para generar sombras suaves y perceptualmente[1] correctas en tiempos intermedios entre el real-time y el offline.[2]

Estos tiempos pueden variar desde unos segundos hasta algunos minutos, con lo cual encuadran en una categoría intermedia, de tiempos semi-interactivos.

---

[1] A diferencia de otras técnicas que se consideran físicamente correctas como ray tracing, radiosity o photon mapping, usualmente offline.

[2] Numerosas aplicaciones requieren generar imágenes de la mejor calidad posible dentro de un límite de tiempo de unos minutos que el usuario puede esperar.

2 Shadows Maps

La técnica conocida como Shadows Maps [1] consiste en proyectar la escena desde el punto de vista de la fuente de luz. Se genera una textura de profundidades basándose en el valor de Z que es visible desde la luz[3], para luego utilizar esta textura al momento de determinar si un píxel está o no iluminado. La sencillez en la aplicación de la técnica sumado a la gran capacidad para integrarse con el resto del pipeline gráfico hacen que sea una de las técnicas más utilizadas para generar sombras en tiempo real. Desafortunadamente sufre de todos los problemas de aliasing y precisión típicos de las técnicas que trabajan en image space usando texturas para almacenar datos auxiliares. Entre los artifacts más importantes se encuentran el aliasing del shadow map y el self shadows, también conocido como Surface Acne. Este trabajo se focaliza en atacar este último problema en el contexto de una solución para generar sombras suaves con áreas extensas de penumbra y perceptualmente correctas (que constituye la segunda parte de este trabajo.)

3 Artifacts del Shadow Maps: Surface Acne

El algoritmo estándar de shadows map requiere comparar la distancia entre el píxel que se está dibujando y la luz, con la correspondiente distancia[4] previamente almacenada en el shadow map. Si ambos valores son iguales, el píxel está iluminado, caso contrario (el valor en shadow map es más pequeño) está en la sombra [2]. En teoría, si el punto está iluminado, tanto la luz, como la cámara estarían observando el mismo punto. Cuando el punto evaluado está en un área de sombras producido por un obstáculo relativamente lejano, ambas distancias son bastante diferentes. Sin embargo, si el punto está iluminado dichos valores, que deberían ser iguales, difieren en una mínima cantidad debido a una combinación de errores de redondeo, precisión y diferencias en el muestreo del shadow map. El algoritmo asumirá incorrectamente que dicho punto está en el área de sombras. Este artifact se lo llama “Self Shadow”[5] por tratarse de una sombra generada por un polígono sobre sí mismo.

---

[3] Cuando se habla de profundidad o valor de Z se hace referencia al valor z/w del punto transformado por el WorldViewProj de la luz, o sea en el lightspace.

[4] El valor de Z representa la distancia desde el punto a la fuente de luz y es el valor que se almacena en la textura del shadow map.

[5] Self Shadow en sí mismo no es un error, ya que es perfectamente posible que un cuerpo genere sombras sobre sí mismo siempre y cuando sea cóncavo. Artifact self shadow o erroneous self shadow hace referencia a cuando un polígono produce sombra sobre el mismo polígono (o sobre alguno muy próximo), en una configuración de superficie “convexa”, y debido a un error de proyección o de precisión.

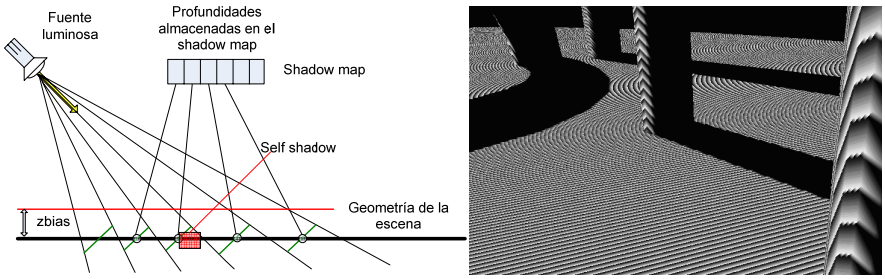

Fig 1: Izquierda: Esquema de Self Shadow. Derecha: Surface Acne producido por error de sampling.

4 Trabajos relacionados

Para resolver el artifact la solución más sencilla es introducir un valor SHADOW_EPSILON en la comparación, denominado Zbias.

```
// Z almacenado en el shadow map
z_smap = tex2D( g_samShadow, ShadowTexC);
// Z calculado en el píxel
z_pixel = vPosLight.z / vPosLight.w;
if(z_map + SHADOW_EPSILON < z_pixel )
    punto en sombra
else
    punto iluminado
```

Si bien ajustando manualmente el Zbias se puede en teoría eliminar el problema, en la práctica esto no resulta sencillo y así diversas soluciones se han propuesto para determinar el Zbias que ajuste mejor a los parámetros actuales del shadow map [3][4].

Un bias demasiado pequeño no resuelve el problema y uno demasiado grande produce otro artifact todavía más notorio: los objetos dejan de producir sombra sobre los lugares más próximos, lo cual es particularmente notable cuando tocan el piso, dando la sensación que están “volando”, defecto que se lo conoce como “Peter Panning”.

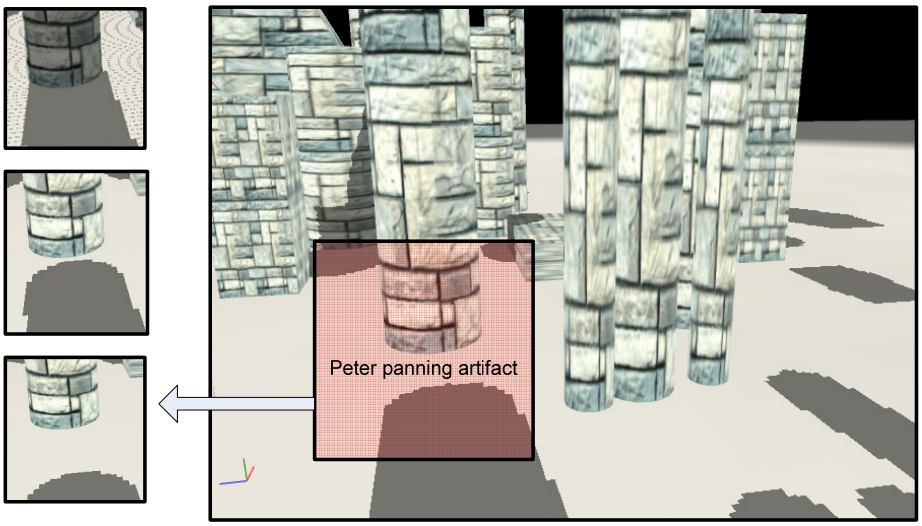

Fig 2: Un Zbias demasiado grande produce que la sombra de la columna desaparezca cerca de la unión con el piso dando la sensación que está volando o despegada del nivel del suelo.

Una forma de resolver el problema de precisión de los float es utilizar lo que se llama un mapa de IDs [5]. En este caso un ID por polígono reemplaza a la profundidad en el shadow map. La comparación se realiza a nivel de IDs que como son enteros no presentan ningún problema de precisión eliminando la necesidad de utilizar un Zbias. Sin embargo el self shadow se sigue presentando entre polígonos próximos entre sí, los cuales se comportan mejor cuando se analizan según su profundidad. Como se puede observar hay casos en los que conviene usar la profundidad para la comparación mientras que en otros casos es conveniente usar el ID. Esto motiva la siguiente solución.

5 Combined ID-Depth Shadows map.

La solución propuesta es una combinación del ID y la profundidad en la misma textura que llamaremos ID-DepthMap. El ID-DepthMap tendrá un formato de D3DFMT_G32R32F [6]: en los 32 bits del canal G almacenará el ID y en los 32 bits del canal R la profundidad.

Para generar los IDs se aplica la estrategia de un ID único para cada malla [6] (o para cada subset). De esta forma no hace falta propagar el ID en una coordenada de textura en cada vértice como se hace en la técnica estándar. Además elimina el problema del self shadow dentro de la misma malla, (ya que todo el objeto tiene el

[6] En DirectX D3DFMT_G32R32F es un formato de textura que permite almacenar 2 float de 32 bits en los canales R y G.

mismo ID). Desafortunadamente también introduce un nuevo artifact: al eliminar el self shadow también elimina la posibilidad de que un objeto produzca sombra sobre sí mismo cuando esa sombra es real. Y por último todavía persiste el problema de self shadow entre polígonos próximos entre sí pero pertenecientes a diferentes objetos, por ejemplo la unión entre una pared y el piso. Estas dos últimas situaciones se resuelven usando la profundidad almacenada en el ID-DepthMap y constituye la motivación de esta técnica.

```
void PixShadow(  float2 Depth : TEXCOORD0,
                 float idFace: TEXCOORD1,
                 out float4 Color : COLOR )
{
    // En R devuelvo el ID del objeto
    Color.r = idFace;
    // En G devuelvo la profundidad
    Color.g = Depth.x / Depth.y;
    Color.ba = 1;  // no se usa
}
```

Entonces, mientras que la comparación por el ID permite resolver en gran medida el self shadow, la profundidad se puede usar para refinar la primera aproximación basada sólo en IDs. Para trabajar con la profundidad se determinan manualmente dos valores de épsilon:

- Un épsilon pequeño, llamado EPSILON, que actúa como Zbias estándar tal como se aplica habitualmente en la comparación de profundidades. (equivalente SHADOW_EPSILON en el ejemplo anterior)

- Un epsilon grande, llamado EPSILON_FAR que representa una distancia a partir de la cual no hay ninguna posibilidad de confundir dos polígonos distintos como si fueran el mismo objeto.

Luego, si la profundidad desde la perspectiva de la fuente luminosa es bastante inferior a la profundidad de referencia, o sea es menor que EPSILON_FAR, el punto se encuentra en el área de sombras, independientemente del ID (que puede llegar a ser el mismo, en el caso que se trate de un objeto cóncavo que produce sombra real sobre sí mismo).

5.1 Ejemplo en objeto cóncavo.

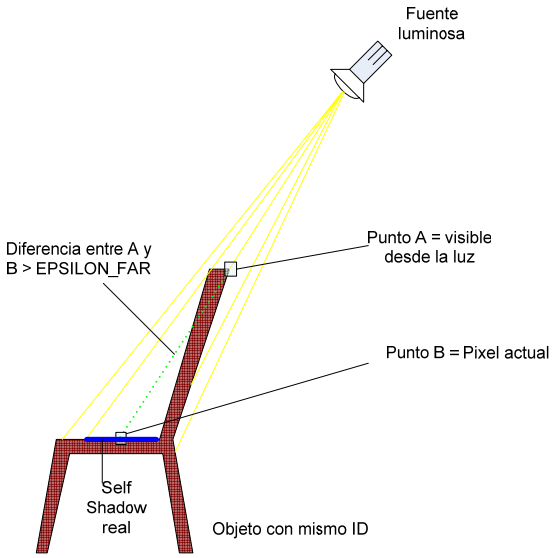

Fig 3: Un objeto cóncavo puede producir sombra real sobre sí mismo. En ese caso tanto el receiver como el blocker corresponden al mismo ID de objeto.

Entonces todas las diferencias grandes en las distancias se manejan con el EPSILON_FAR que permite resolver los casos de sombras típicos en los que el occluder[7] está bien alejado del receiver. Esta parte de la comparación soporta el self shadow real (es decir cuerpos cóncavos que producen sombras sobre sí mismos) y no genera ningún tipo de artifact. Cuando la diferencia es inferior a EPSILON_FAR, pero superior al EPSILON entra en juego el ID estándar. Si el ID proyectado desde la luz coincide con el ID que se está analizando el punto está iluminado. Hasta aquí tampoco hay ningún tipo de artifact, ya que la comparación es entre números enteros y no hay problemas de precisión o de redondeo.

Por último si la diferencia de profundidad es inferior al épsilon pequeño se asume que la fuente de luz y el observador están proyectando el mismo punto, independientemente de los IDs, y por lo tanto el punto está iluminado. Es decir si las caras son diferentes, pero la profundidad es muy similar, se interpreta como dentro del error de sampling del shadow map y se considera iluminado. Esto resuelve los problemas de acne y self shadows en las uniones entre polígonos diferentes.

[7] En el contexto de las sombras se llama occluder, blocker o shadow caster al objeto que produce sombras, sobre otro que las recibe llamado receiver.

5.2 Ejemplo en escena de test.

En el siguiente ejemplo la escena consta de una silla (con dos IDs: las varillas de madera y el asiento blanco de tela) y un piso. Se puede ver la sombra lejana proyectada por la silla en el piso a una distancia en la región del EPSILON_FAR. La sombra de la pata de la silla sobre el piso, cerca del punto de unión, está en una distancia menor al EPSILON_FAR pero superior al EPSILON pequeño. Sin embargo por tratarse de distintos IDS el algoritmo interpreta correctamente esa sombra. Un caso similar es el de las varillas que proyectan sombras sobre el asiento. Por último la sombra sobre las varillas se produce dentro de la región < EPSILON, y corresponde a los polígonos de la parte posterior de la silla que producen sombras sobre los que están en la parte delantera.

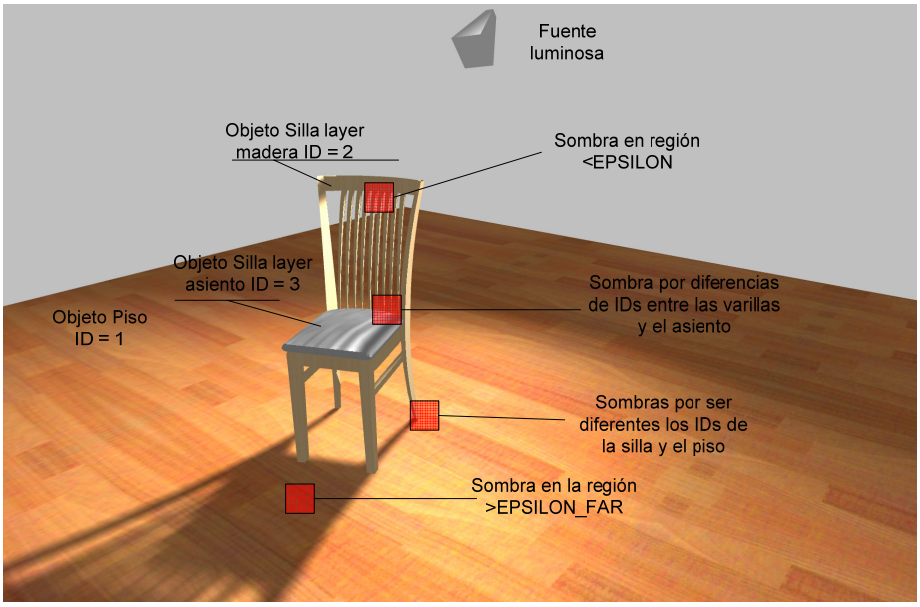

Fig 4: Silla que proyecta sombras en las 3 áreas definidas por EPSILON y EPSILON_FAR.

5.3 Ejemplo de Self Shadow Real

El siguiente ejemplo muestra un caso concreto de un objeto cóncavo que produce una sombra real sobre sí mismo (que no es un artifact) y como se resuelve gracias al uso de EPSILON_FAR. Todavía queda un pequeño artifact producto del Zbias cuando la distancia no es superior a EPSILON_FAR.

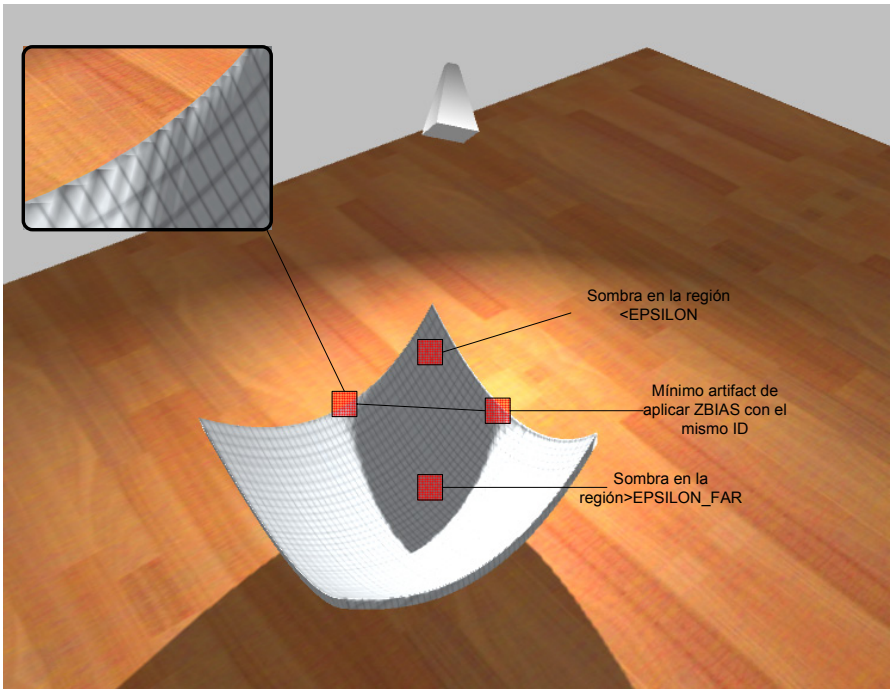

Fig 5: Ejemplo de self shadow Real de un objeto cóncavo.

5.4 Comparación combinada

```
// Acceso al Shadow Map
float4 tx = tex2Dlod(g_samShadow,TexCoord);
// Obtener Shadow map face y depth
int face_smap = round(tx.r);
float depth_smap = tx.g;
// Comparacion combinada
if( depth_smap + EPSILON_FAR < depth  ||
    (face_smap!=id_face && depth_smap + EPSILON < depth))
        // El punto esta en la sombra
```

5.5 Tabla Resúmen

Diferencia entre distancias	Método de Comparación	¿Hay sombra?
> EPSILON_FAR	Por Profundidad	Siempre
> EPSILON	Por IDs	Si los IDs son ≠
≤ EPSILON	Por Profundidad	Nunca

6 Conclusiones

La simple introducción de un ID global para toda una malla o un subset, combinado con la profundidad del shadow map estándar y el EPSILON_FAR minimiza notablemente los artifacts producidos por el Zbias manual. A su vez es muy fácil de integrar en un pipeline ya existente, ya que sólo requiere pasar solo un ID como variable gobal al shader y mínimos cambio al píxel shader del shadow map usual.

7 Trabajo futuro

Todavía hay un mínimo error de muestreo en los puntos que son <= EPSILON. Dentro de ese entorno los puntos se consideran iluminados, aun cuando pertenezcan a otro ID. Eso produce una mínima pérdida de sombra (artifact que se denomina peter panning). Como el EPSILON se puede ajustar para que sea bien pequeño (gracias al ID y el EPSILON_FAR) este artifact tiende a ser mínimo. Sin embargo, gran parte del problema reside en que el EPSILON constante es una mera aproximación, en realidad se trata de una función que depende de la relación entre las matrices de proyección de la luz y el observador. Se podría estudiar la forma de generar el EPSILON variable dentro del vertex shader o del píxel shader para ajustarlo mejor a cada situación.

8 Referencias

1. L. Williams: Casting curved shadows on curved surfaces. SIGGRAPH Comput. Graph., vol. 12, nº 3, pp. 270-274, (1978).
2. E. Eisemann, M. Wimmer, U. Assarsson y M. Schwartz, Real-Time Shadows, Taylor & Francis, (2011).
3. Lengyl, E.. Tweaking a vertex's projected depth value. In Game Programing Gens, pp 361-365. Charles River Media, Hingham, MA. (2000)
4. Schüler c. Eliminating surface acne with gradient shadow mapping. In Shader X: adevanced Render techniques. pp 289-297.Charles River Media, Hingham, MA. (2005)
5. Hourcade, J-C. and Nicolas, A.. Algorithms for antialiased cast shadows. Computer and Graphics, 9,3, 259-265. (1975)
6. Dietrich, S.. Practical priority buffer shadows. In Game Programming Gems 2. pp 481-487. Charles River Media, Hingham, MA. (2001)