PROGRAMACIÓN 3D

Máster en Programación de Videojuegos

TEMA 5: SOMBRAS

Juan Mira Núñez

U-tad
CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL

Índice

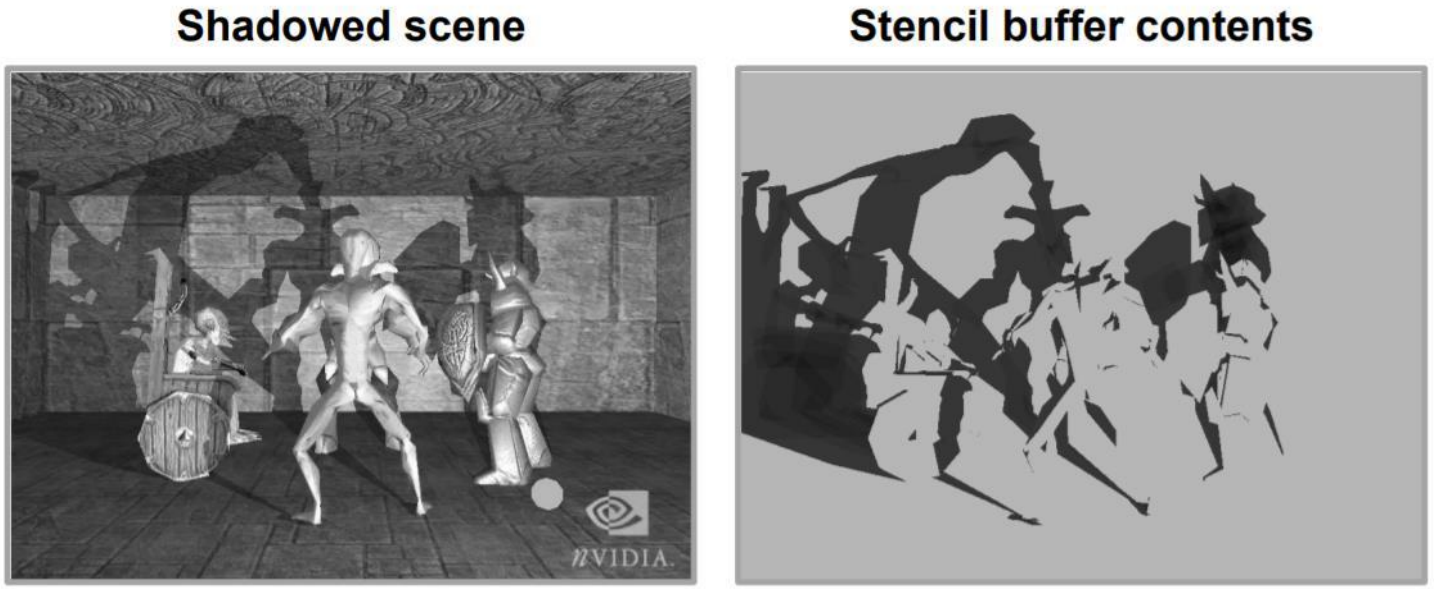
- Introducción
- Framebuffers y renderizado en texturas
- Fuentes de luz direccionales
  - Renderizado del shadow map
  - Renderizado del mundo

Sombras. Introducción

- Anteriormente, hemos visto cómo implementar un sistema de iluminación.
- En cambio, el efecto de iluminación no está completo si, además de poder definir fuentes de luz que iluminen los objetos, no tenemos un sistema para proyectar sombras basadas en la geometría de los mismos.
- OpenGL proporciona un buffer especial, el stencil buffer, que permite dibujar este tipo de sombras proyectadas.

Sombras. Introducción

- A este tipo de sombras se le denomina hard shadows , ya que el volumen de la sombra es uniforme (no se produce difuminado en los puntos más alejados de la fuente de luz).



[Paper de Nvidia en la GDC sobre shadow volumes]

Sombras. Introducción

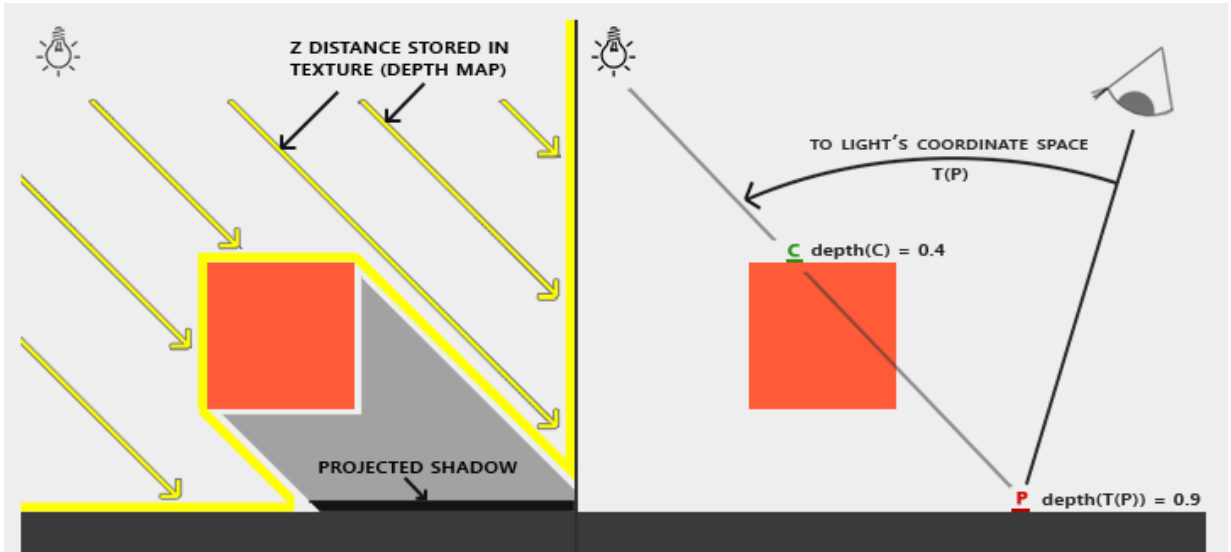
- Este sistema ha caído en desuso, aunque fue utilizado en algunos juegos muy populares como Doom 3 de id Software.
- Hoy en día, utilizando shaders , es posible emplear otras técnicas de sombreado que nos permiten obtener soft shadows , o sombras con suavizado.

Sombras.

- Implementar proyección de sombras es una tarea muy sencilla.
- Implementar proyección de sombras correctamente es, en cambio, un proceso complicado, debido a la gran cantidad de factores a considerar para que no se produzcan defectos (artifacts) en el renderizado de las sombras.
- En este tema, aprenderemos a implementar un sistema de proyección de sombras basado en shaders .
- Utilizaremos dos pasadas del renderer para implementar el sombreado.
    1. En la primera, renderizamos la escena desde la posición de la fuente de luz, y guardaremos la profundidad de cada fragmento renderizado en una textura.
    2. En la segunda, renderizamos la escena de la manera habitual, pero compararemos la profundidad de cada objeto con la profundidad obtenida, y oscureceremos los fragmentos que están ocluidos.

Framebuffers

- (Pass 1) Renderizamos la escena desde la posición de la fuente de luz. El depthBuffer guardará la distancia de la luz al objeto más cercano.
- (Pass 2) Renderizamos la escena de la manera habitual, pero compararemos la profundidad de cada objeto con la profundidad obtenida, y oscureceremos los fragmentos que están ocluidos..



Framebuffers

- Para implementar las sombras, vamos a necesitar en primer lugar ser capaces de renderizar la escena en una textura.
- Al comienzo del curso hablamos del back buffer, ese buffer de la memoria donde realizamos el pintado de elementos antes de volcarlo en la pantalla.
- También hemos hablado ya del depth buffer, un buffer donde se almacena la distancia al observador de cada fragmento dibujado.
- En OpenGL, el back buffer se denomina color buffer , y al conjunto de todos los buffers donde se escribe información (color, depth , stencil …) se le denomina el **framebuffer** . A cada uno de los buffers que contiene se le denomina un render buffer.
- Por defecto, el framebuffer activo será nuestra ventana de OpenGL, y a la hora de crearla especificaremos qué render buffers queremos que tenga (color, depth , stencil).

Framebuffers

- **void glGenFramebuffers GLsizei n, GLuint * ids)**

Genera n framebuffers y coloca sus identificadores en el array apuntado por ids .

- **void glDeleteFramebuffers GLsizei n, const GLuint * framebuffers**

Elimina n framebuffers cuyos identificadores se encuentras en el array apuntado por framebuffers .

- **void glBindFramebuffer GLenum target, GLuint framebuffer)**

Activa el uso del framebuffer especificado como destino para las operaciones de renderizado. El valor de target debe ser GL_FRAMEBUFFER

Framebuffers

- **Void glFramebufferTexture(GLenum target, GLenum attachment , GLuint texture , GLint level)**

Vincula una textura a una de las salidas del framebuffer .

El valor de target debe ser **GL_FRAMEBUFFER**.

El valor de level (que representa el número de mipmap) debe ser 0.

El valor de attachment puede ser:

- **GL_COLOR_ATTACHMENTi:** Dibuja en la textura especificada una de las salidas de color del fragment shader .
- **GL_DEPTH_ATTACHMENT:** Escribe en la textura especificada el depth buffer.
- **GL_STENCIL_ATTACHMENT:** Escribe en la textura especifica el stencil buffer (no lo utilizamos en esta asignatura)
- **GL_DEPTH_STENCIL_ATTACHMENT:** Escribe en la textura especificada el depth buffer y el stencil buffer simultáneamente.

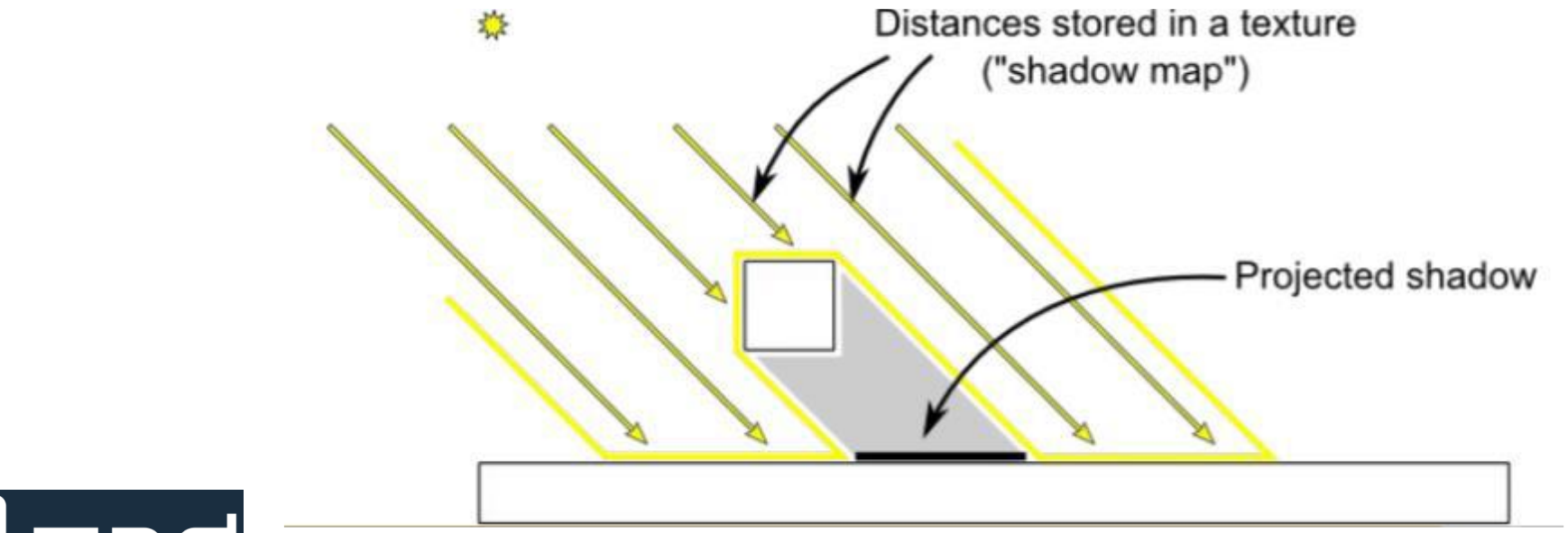
Renderizado de texturas

La textura en la que se dibujará cada buffer se genera de la misma manera que ya vimos anteriormente, pero hay que tener en cuenta algunas consideraciones:

- Su modo de filtrado (tanto de ampliación como de reducción debe ser **GL_NEAREST**).
- **No** se generarán **mipmaps**.
- Si vamos a utilizar la textura para escribir el depth buffer, en la función glTexImage2D debemos utilizar estos parámetros:
  - **internalFormat (3er parámetro):** Utilizaremos GL_DEPTH_COMPONENT16, GL_DEPTH_COMPONENT24 o GL_DEPTH_COMPONENT32.
  - **format (7º parámetro):** Utilizaremos GL_DEPTH_COMPONENT.

Fuentes de luz direccionales

- Vamos a aprender a proyectar sombras utilizando fuentes de luz direccionales.
- Obtendremos una textura con la información sobre la proyección desde la fuente de luz, como se ve en la siguiente imagen:



Fuentes de luz direccionales

- Los rayos de luz emitidos por una fuente de luz direccional son todos paralelos.
- Para obtener los puntos sombreados de la escena, en primer lugar, renderizaremos la misma desde la posición de la luz, y guardaremos la información de **profundidad** de cada elemento.
- La proyección debe de ser ortográfica (líneas de proyección paralelas), para que los objetos no tengan una disminución aparente de tamaño con la distancia.

Fuentes de luz direccionales

- Como sólo necesitamos información de la profundidad, es mucho más rápido crear un **vertex y un fragment shader específicos** para esta etapa de renderizado.
- Creamos una matriz MVP especial (nos referiremos a ella aquí como DepthMVP) utilizando la **matriz de proyección ortográfica.**
  - En el vertex shader, transformamos cada vértice por esta matriz.
  - En el fragment shader, no hacemos nada (si el test de profundidad está activo, la escritura en el depth buffer es automática).

Fuentes de luz direccionales

- **Después, utilizamos los shaders estándar** para renderizar la escena, pero pasamos al shader un **sampler2D** con la textura de profundidad enlazada.
- También, en el vertex shader , además de calcular las coordenadas de un vértice con la transformación de la escena, debemos calcular sus coordenadas en la textura de profundidad.

➢ Multiplicando el vértice por la matriz MVP, obtenemos su proyección actual.

➢ Multiplicando el vértice por la matriz DepthMVP , obtenemos sus coordenadas en la proyección que hicimos en el primer paso. Estás coordenadas están en el rango [-1,1], siendo (0,0) el centro de la pantalla.

Fuentes de luz direccionales

- Como las coordenadas en la textura están en el rango [0,1], debemos multiplicar por 0.5, lo que nos dará coordenadas en el rango [-0.5, 0.5], y sumarle 0.5, lo que nos dará el rango [0,1] buscado.
- Podemos definir esta última transformación en la siguiente matriz:

$$\begin{vmatrix} 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{vmatrix}$$

- Si multiplicamos esta matriz por DepthMVP, obtenemos una única matriz que realiza toda la transformación, y se la podemos mandar al shader (la podemos llamar DepthBias).

Fuentes de luz direccionales

- Si multiplicamos el **vértice por DepthBias** en el shader, obtenemos sus coordenadas dentro de la textura, que enviaremos al fragment shader.
- También debemos definir en el **fragment shader un sampler2D** con la unidad donde está enlazada la textura de profundidad.
- **En este shader, debemos calcular si el fragmento está ocluido o no**.
    - Si la coordenada z de la textura en el punto transformado por DepthBias es menor que la coordenada z de dicho punto, entonces ese fragmento está ocluido por otro, y debemos multiplicar el color del fragmento por el color de la luz ambiente.

Fuentes de luz direccionales

- El cálculo del sombreado se pude mostrar incorrectamente. A este defecto se le denomina **shadow acne** .
  - Hay varias causas que pueden producirlo. La principal es que la profundidad de un fragmento está muy cerca de la profundidad en ese píxel de la textura.
  - Es importante dar valores apropiados al rango de la cámara para que la textura de profundidad tenga precisión.
  - Además, podemos forzar a que la diferencia de profundidad tenga que ser mayor de una determinada cantidad.

- El cálculo completo quedaría de la forma (el valor de shadowColor se multiplicará al color final del fragmento):

```
Vec4 shadowColor = vec4(1, 1, 1, 1);

If ( texture2D(depthSampler , vec2(depthCoord )).z < depthCoord.z - 0.0009 )

    shadowColor = vec4(ambientLight, 1);
```

Documentación relacionada

- Para más información recomiendo:

https://learnopengl.com/book/learnopengl_book.pdf

¿Dudas?