

Tema 2 - Arquitectura y modelos para entornos virtuales.

2.3 Sistemas básicos de iluminación y cámaras.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

Tema 2: Arquitectura y modelos para entornos virtuales.

- 2.1 Grafos de escena y modelos jerárquicos.
- 2.2 Métodos básicos de representación.
- 2.3 Sistemas básicos de iluminación y cámaras.
- 2.4 Modelos de generación procesal.

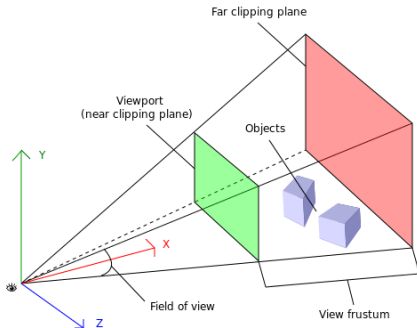
2.1 Sistemas básicos de iluminación y cámaras



Figura 1: Una escena sin iluminación o cámara.

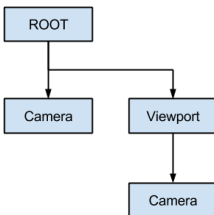
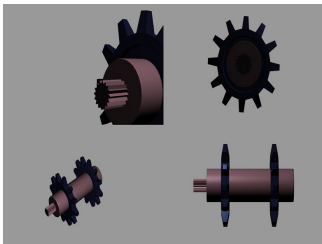
Cámaras (I)

- Cámara y pantalla no son lo mismo.
- **Vista:** (viewport) la vista es el area visible de la cámara activa. Suele estar ligado a un *buffer*, por lo que tiene una resolución fija en píxeles.



Cámaras (II)

- Una cámara está asociada a una vista (normalmente su vista padre).
- Solamente puede haber una cámara activa.
- Las cámaras pueden tener sub-vistas asociadas (sub-viewport), que suelen renderizar a texturas.



Ejercicio Teórico-Práctico:

Creación de un cámara FPS.

Estudia el siguiente tutorial:

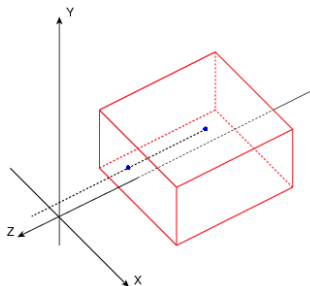
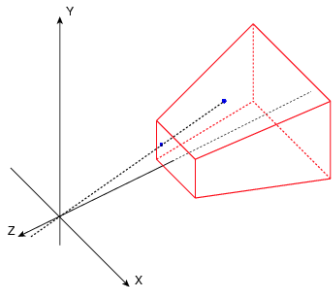
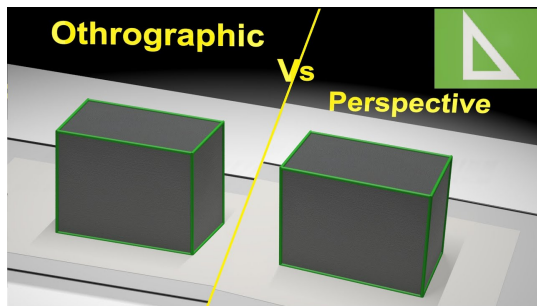
https:

[//docs.godotengine.org/en/stable/tutorials/3d/fps_tutorial/part_one.html](https://docs.godotengine.org/en/stable/tutorials/3d/fps_tutorial/part_one.html)

Plantea el movimiento de cámara con raton en Godot. Implementa un botón de salto sin utilizar física de Godot, para ello estudia el movimiento de tiro parabólico.

https://en.wikipedia.org/wiki/Projectile_motion

Perspectiva vs. ortogonal



La física de la luz (I)

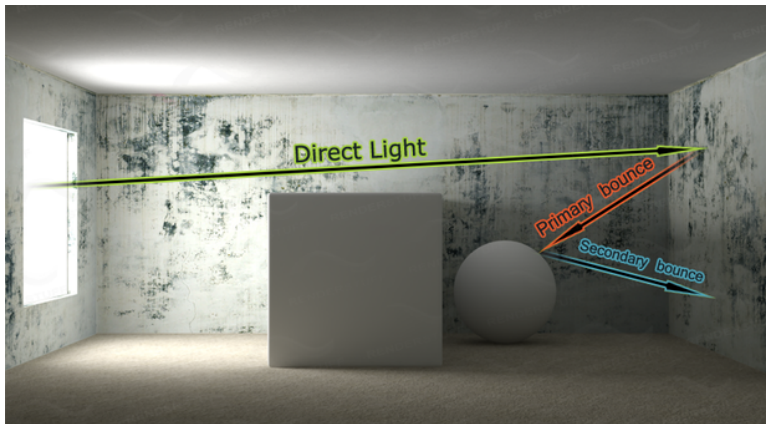


Figura 3: Ejemplo de rebotes de luz directos e indirectos.

La física de la luz (II)



Figura 4: Ejemplo de difracción de la luz.

Modelo sencillo de iluminación: fuentes de luz

Las fuentes de luz son la forma más sencilla de iluminación:

- Fuente puntual: $\vec{p} = (p_x, p_y, p_z)$. Emiten luz en todas direcciones.
- Fuente direccional: $\hat{d} = (d_x, d_y, d_z)$. Emite luz en una única dirección (normalizada). Está situada en el infinito.
- Fuente focal: Viene dada por posición y dirección, y tiene un ángulo de apertura. $F \in \mathbb{E}^3 \times \mathbb{R}^3 \times \mathbb{R}$

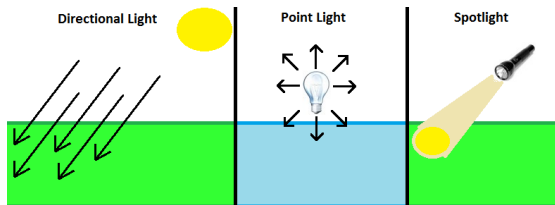


Figura 5: Tipos de luces básicas.

Funcionamiento de la iluminación básica

Una *normal de una cara* es el vector unitario en \mathbb{R}^3 que sale perpendicular a esta.

- La orientación de la cara normalmente indica el sentido del vector de dirección.
- Un vértice tiene varias normales, correspondiente a cada una de sus caras.
 - ▶ En los entornos de RV las normales se suelen definir a nivel de vértices (**normal por vértice**).

```
normal_array[0] = Vector3(0, 0, 1)
```

Lambert: el modelo de iluminación más sencillo

Para cada punto de la superficie:

$$c = \hat{L} \cdot \hat{N} = |\vec{N}| |\vec{L}| \cos \alpha = \cos \alpha$$

Nota como los vectores de dirección de luz y normales están **normalizados**.

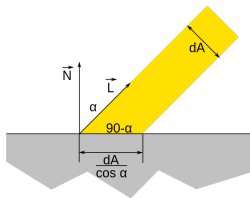


Figura 6: Modelo de lambert: luz difusa.

Materiales sencillos e iluminación

- Pero la superficie puede tener un material. El material más sencillo es un color.
- Un **color** c se define como un vector en \mathbb{R}^3 , con tres componentes: rojo, verde y azul. Aunque el dominio visible está en $[0, 1]$, a veces se permite exceder este valor (ej. materiales de emisión).
- Los colores se mezclan lumínicamente, dos colores se pueden mezclar entre sí multiplicando componente a componente ambos colores.

Entonces:

$$c = c_{material} \cdot \hat{L} \cdot \hat{N} = c_{material} \cdot |\vec{N}| |\vec{L}| \cos \alpha = c_{material} \cdot \cos \alpha$$

- Si hay varias luces, entonces se suman los valores.

Propiedades de las luces

Las luces suelen extenderse añadiendo propiedades numerosas:

- **Color.**
- **Energía:** un multiplicador a la intensidad (el color ya no es normalizado en ese caso).
- **Energía indirecta:** segundo multiplicador usado para la luz indirecta y ambiental.
- **Luz negativa:** la luz se vuelve sustractiva en lugar de aditiva.
- **Especular:** afecta a la intensidad del brillo que se produce en determinados materiales.
- **Cocinado (bake):** las luces se pueden precalcular para aumentar el rendimiento.
- etc.

Sombras arrojadas

En los mundos virtuales las luces son las que arrojan sombras (opcional).

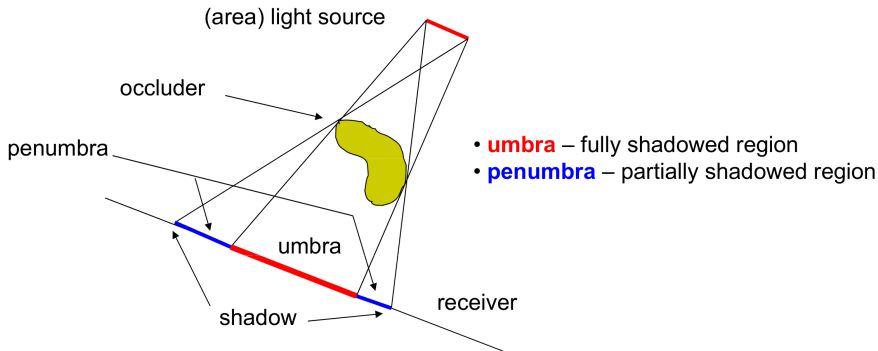


Figura 7: Esquema de sombra arrojada por una luz.

Sombras arrojadas terminología

Definiciones:

- **Receptor:** (*receiver*) objeto que está bajo la sombra.
- **Ocluser:** (*caster/occluder*) objeto que bloquea la luz del receptor.
- **Umbra:** región que está completamente bajo una sombra.
- **Penumbra:** región que está en transición entre la umbra y un área iluminada.

Tipos de sombra según la luz (I)

Fuentes puntuales, direccionales y algunas focales:

- Las sombras son duras si no hay iluminación global de escena (no hay penumbra).
- Son rápidas de calcular.

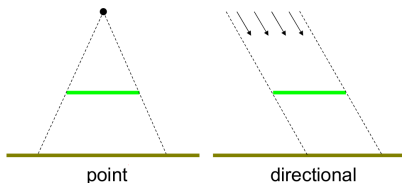


Figura 8: Sombras generadas por luces puntuales y direccionales.

Tipos de sombra según la luz (II)

Sombras en luces de área (se usa un rectángulo y múltiples vectores en bordes) y algunas focales (se pueden usar múltiples vectores):

- Las sombras son suaves incluso si no hay iluminación global de escena.
- Son algo más lentas de calcular.

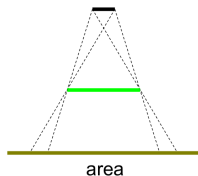


Figura 9: Sombras generadas por luces de área.

Técnicas para el cálculo de sombras

Técnicas más comunes para el cálculo de sombras en tiempo real:

- *Planar Shadows.*
- *Shadow Volume.*
- *Shadow Maps.*
- *Ray casting.*

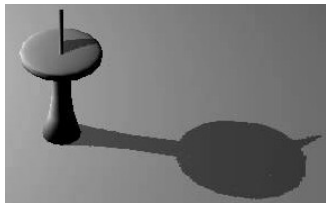
Planar Shadows (I)

- Se utiliza una matriz de proyección.
- Dado un punto \vec{p} del plano, y su normal \hat{n} , calculamos la ecuación del plano: $d = -(\hat{n} \cdot \vec{p})$, $\hat{n} \cdot \vec{p}_i + d = 0$.
- A partir del plano, y un punto de proyección p , creamos una matriz que proyecte un punto arbitrario en ese plano:

$$M = \begin{pmatrix} \hat{n} \cdot \vec{p} + d - p_x n_x & -p_x n_y & -p_x n_z & -p_x d \\ -p_y n_x & \hat{n} \cdot \vec{p} + d - p_y n_y & -p_y n_z & -p_y d \\ -p_z n_x & -p_z n_y & \hat{n} \cdot \vec{p} + d - p_z n_z & -p_z d \\ -n_x & -n_y & -n_z & \hat{n} \cdot \vec{p} \end{pmatrix}$$

Planar Shadows (II)

- Esta matriz M es una transformación homogénea, y se inserta después de calcular el espacio objeto-mundo, y antes de la transformación mundo-ojo.
- Algoritmo:
 - 1 Deshabilitar el *z-buffer* y dibujar la sombra en el *alpha-buffer*.
 - 2 Habilitar el *z-buffer* y dibujar el objeto.
 - 3 Dibujar el suelo y modular el *canal alpha* de destino.
- Problema: sombras sin penumbra.



Shadow volume (I)

- Un volumen es un espacio formado por un objeto ocluidor y la luz.
- Limitado por las aristas del objeto ocluidor.
- Todos los objetos dentro del volumen están en sombra.

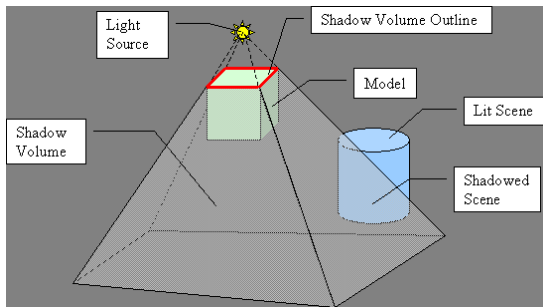


Figura 10: Esquema que define el volumen de sombra.

Shadow volume (II)

Algoritmo:

- ➊ Crear un contador.
- ➋ Lanzar un rayo en la escena.
- ➌ Incrementar el contador cuando el rayo perfora una cara frontal.
- ➍ Decrementar el contador cuando el rayo perfora una de las caras traseras del volumen de sombra.
- ➎ Cuando golpee un objeto:
 - ▶ Si contador $> 0 \rightarrow$ en sombra.
 - ▶ En otro caso \rightarrow no está en sombra.

Shadow volume (III)

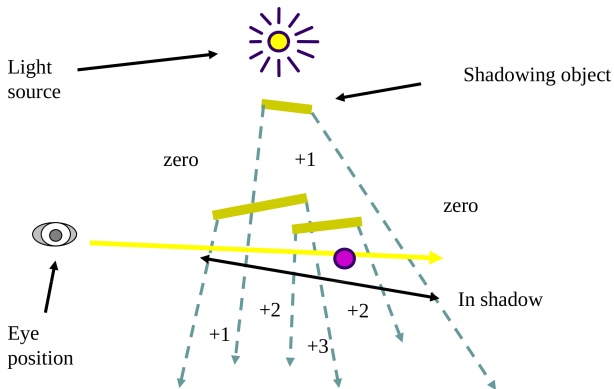


Figura 11: Ejemplo de algoritmo para Shadow volumes.

Shadow volume (IV)

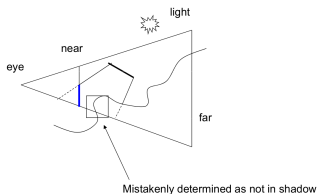
Implementación: solución *z-pass*

- ➊ Renderiza la escena solamente con luz ambiente y emisión, se actualiza el *z-buffer*.
- ➋ Desactivamos el *z-buffer* y la escritura en buffers de color. Activamos el *stencil buffer* (manteniendo el test de profundidad activo).
- ➌ Inicializamos el *stencil buffer* a 0.
- ➍ Dibujamos el volumen de sombras dos veces (usando *face culling*).
 - ▶ 1ª pasada: renderizamos las caras frontales, +1 *stencil buffer* si pasa el test de profundidad.
 - ▶ 2ª pasada: renderizamos las caras traseras, -1 *stencil_buffer* si pasa el test de profundidad.
- ➎ Renderizamos la escena de nuevo, con la luz de la fuente cuando los píxeles valen 0 en el stencil. (Si en un pixel $\text{stencil} \leq 0 \rightarrow$ en sombra, en otro caso iluminado.)

Shadow volume (V)

Problemas del *z-pass*:

- 1 Cuando el plano cercano intersecta con alguna cara del volumen de sombras (corta la cara).
- 2 Cuando el ojo está en el volumen de sombras
 - ▶ contador = 0 ya no implica sombra → deberíamos tener en cuenta el n° de volúmenes en los que estamos.



Shadow volume (VI)

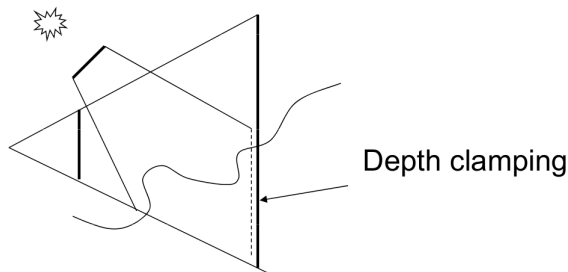
Implementación: solución *z-fail*

- ➊ Renderiza la escena solamente con luz ambiente y emisión, se actualiza el *z-buffer*.
- ➋ Desactivamos el *z-buffer* y la escritura en buffers de color. Activamos el *stencil buffer* (manteniendo el test de profundidad activo).
- ➌ Inicializamos el *stencil buffer* **según el punto de vista**.
- ➍ Dibujamos el volumen de sombras dos veces (usando *face culling*).
 - ▶ 1ª pasada: renderizamos las caras **traseras**, +1 *stencil buffer* si **falla** el test de profundidad.
 - ▶ 2ª pasada: renderizamos las caras **delanteras**, -1 *stencil_buffer* si **falla** el test de profundidad.
- ➎ Renderizamos la escena de nuevo, con la luz de la fuente cuando los píxeles valen 0 en el stencil. (Si en un pixel stencil $\leq 0 \rightarrow$ en sombra, en otro caso iluminado.)

Shadow volume (VII)

Problemas del *z-fail*:

- 1 La sombra puede penetrar el plano trasero → se puede poner un máximo a la distancia en el *z-buffer*, cercano al volumen de sombras.



Shadow maps (I)

- Determinar si el objeto es visible:
 - Usar el algoritmo de *z-buffer* anterior, pero la cámara hace de luz.

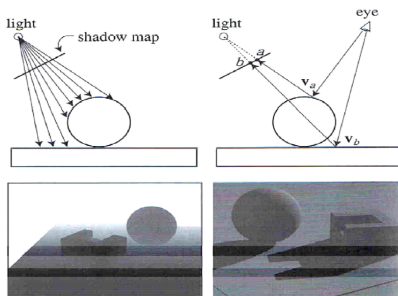
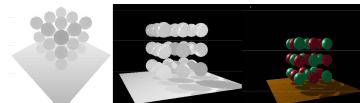


Figura 12: Algoritmo de mapa de sombras.

Shadow maps (II)

Algoritmo:

- ➊ Renderizar la escena usando la luz como cámara y almacenando el *z-buffer*.
- ➋ Generar un buffer de luz (textura 2D) a partir del *z-buffer* (llamado *shadow map*).
- ➌ Renderizar la escena usando la cámara y con el *z-buffer* activo
 - ▶ Para cada fragmento visible en espacio local, transformarlo al espacio de la luz (matrices *modelview*, *projection* y *shadowmatrix*):
 $(x, y, z) \rightarrow (x', y', z')$.
 - ▶ Comparar z' con $z = \text{shadow_map}(x', y')$, si $z' \leq z$ (es más cercano que la luz), entonces el pixel no está en la sombra, en caso contrario sí.



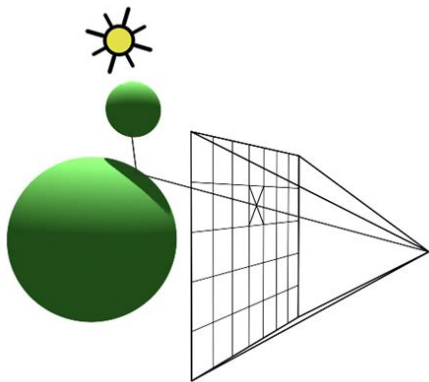
Shadow maps (III)

Problemas:

- Depende de la resolución de la textura (*aliasing*).
- Resolución del *z-buffer*, normalmente 8-24 bits de precisión (depende de la GPU).
- Aliasing en sombras del objeto ocluidor producidas por el mismo objeto ocluidor (*self-shadow*).
- Las sombras aparecen como por bloques (pixelado).
 - ▶ Especialmente problemático cuando un pixel del mapa de sombras cubre varios píxeles de la pantalla.

Ray casting (I)

- Se lanzan rayos: $\vec{r}(t) = \vec{o} + t\vec{d}$, con $0 \leq t \leq \infty$
- Se simulan rayos de luz, que viajan siempre en línea recta, hasta que rebotan (se usa la normal de la superficie).

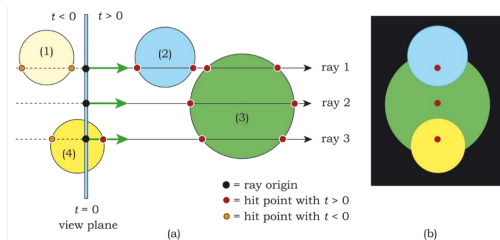


Ray casting (II)

Algoritmo:

- 1 Los rayos se lanzan desde cada píxel de la cámara.
- 2 Si un rayo choca contra un objeto, se lanzan rayos hacia las fuentes de luz.
 - Si alguna de estos rayos choca contra un objeto, está bajo sombra.

Permite hacer sombras translúcidas, incluso de colores.



Iluminación global (I)



Figura 13: Ejemplo de iluminación global.

Iluminación global (II)

Algoritmos en tiempo real (o casi):

- *Ray Tracing* y derivados.

- ▶ Koskela, M., Immonen, K., Mäkitalo, M., Foi, A., Viitanen, T., Jääskeläinen, P., ... & Takala, J. (2019). **Blockwise multi-order feature regression for real-time path-tracing reconstruction**. ACM Transactions on Graphics (TOG), 38(5), 1-14.

https://www.youtube.com/watch?v=julyJEUVDbl&feature=emb_logo

- ▶ Majercik Z., Guertin J.P., Nowrouzezahrai D., MC Guire M.: **Dynamic diffuse global illumination with ray-traced irradiance fields**. Journal of Computer Graphics Techniques Vol 8, 2 (2019)

Iluminación global (III)

Algoritmos en tiempo real (o casi):

- *Radiosity* y derivados.
 - ▶ MC Taggart G.: **Half-life 2/valve source shading**. In Game Developer's Conference (2004), vol. 1. 1
- *Signed Distance Fields*.
 - ▶ Hu, J., Yip, M., Alonso, G. E., Gu, S., Tang, X., & Jin, X. (2020). **Signed Distance Fields Dynamic Diffuse Global Illumination**. arXiv preprint arXiv:2007.14394. <https://godotengine.org/article/godot-40-gets-sdf-based-real-time-global-illumination>
https://www.youtube.com/watch?v=ztkBRFocHww&feature=emb_logo

Iluminación y visualización

- Materiales, algoritmos de visualización e iluminación están íntimamente ligados.

