Tema 7: Realismo Visual

José Ribelles

Departamento de Lenguajes y Sistemas Informáticos, Universitat Jaume I

VJ1221 - Informática Gráfica

Contenido

Hoy veremos...

Introducción

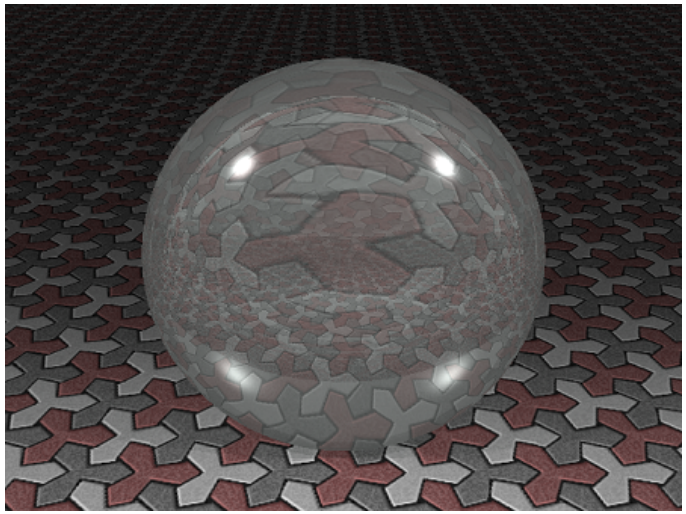
¿Qué podemos hacer para mejorar aún más la calidad visual?

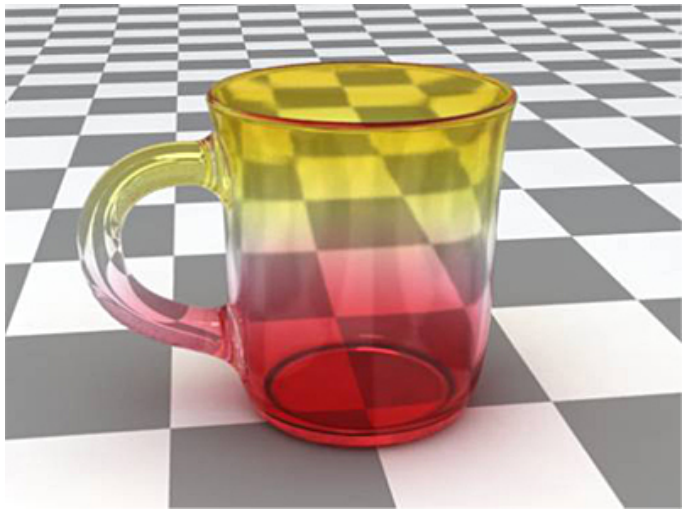
Hoy veremos...

Transparencia

Descripción

- Los objetos transparentes son muy habituales.
- Es un problema complejo.
- Suelen ir asociados con un efecto de refracción de la luz (agua, vidrio, etc.).
- Pueden cambiar el color de la luz al atravesarlos.





Transparencia

¿Qué podemos hacer nosotros?

- Nos vamos a centrar en el caso de objetos transparentes:
  - delgados, con superficies muy finas,
  - y en cómo afectan a los objetos que se ven a través de ellos.
- ¿Es simplificar mucho el problema?
  - Sin duda, pero aún así la ganancia visual que vamos a obtener va a ser muy alta.

Aleksandar Rodic, `http://arodic.github.io/p/jellyfish/`

Transparencia

¿Cómo procedemos?

- Si hay algún objeto transparente, el color de los píxeles cubiertos por dicho objeto depende de sus propiedades y las de los objetos que hayan detrás de él.
- Hay que mezclar el color del objeto transparente con lo que haya detrás.
- Alpha Blending: el grado de transparencia se suministra a la GPU como cuarta componente en las propiedades de material, conocida como componente *alpha*.
- Si *alpha* es 1 el objeto es totalmente opaco, y 0 significa que es totalmente transparente:

$$C_{final} = alpha \cdot C_{fragment} + (1 - alpha) \cdot C_{framebuffer} \quad (1)$$

- Para que esto funcione, hay que dibujar en primer lugar los objetos que sean opacos para después dibujar los transparentes.

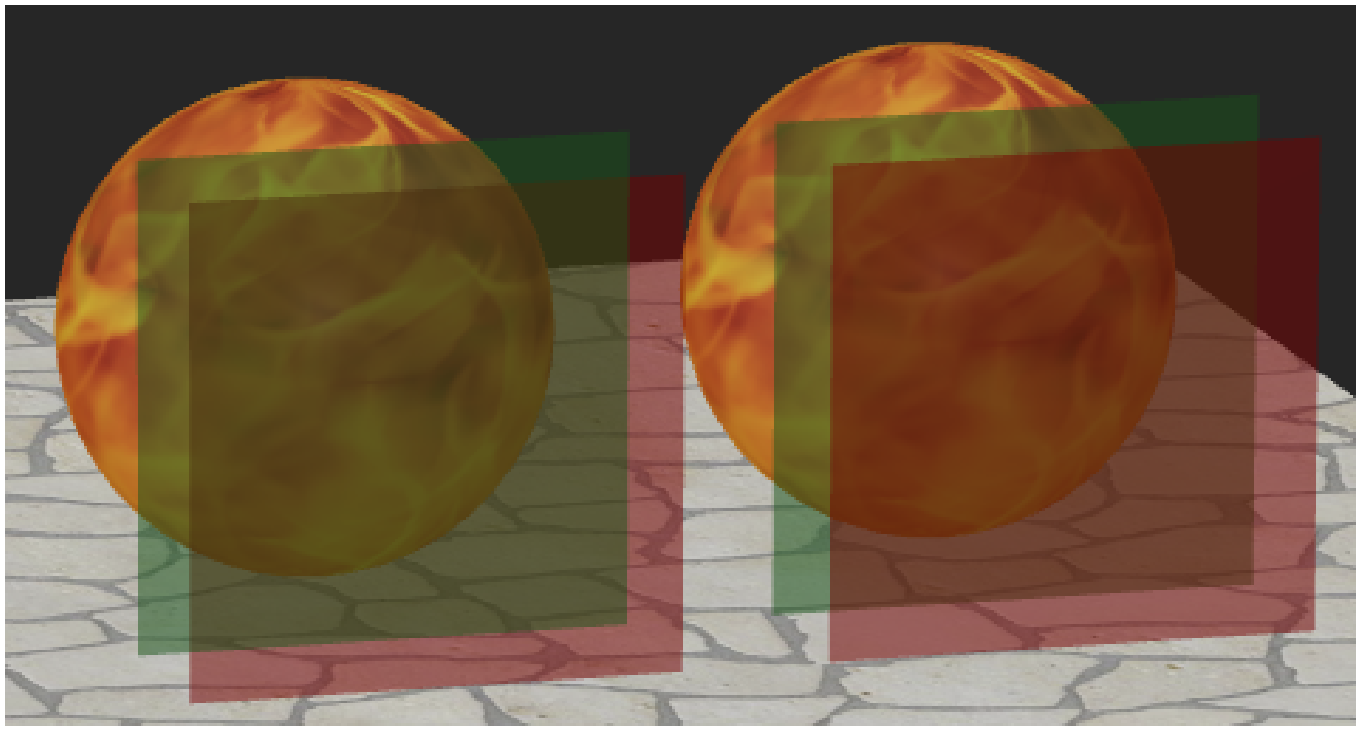
Ejemplo de transparencia

Alpha: 0.3, 0.5, 0.7

Transparencia

¿Y si se ve un objeto transparente a través de otro transparente?

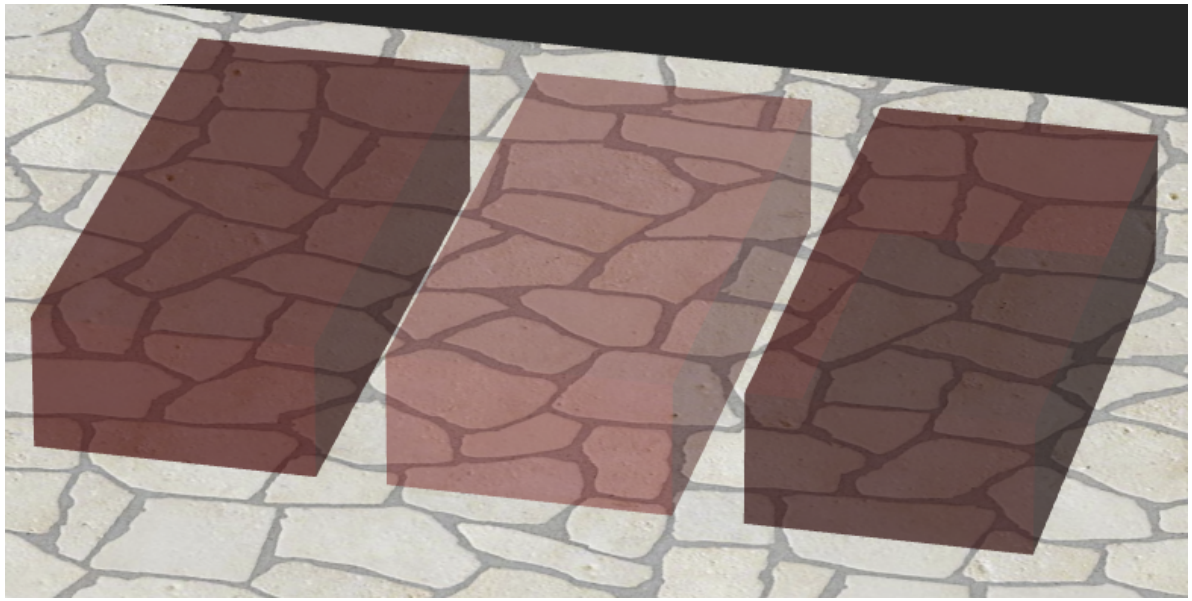
- El resultado depende del orden.
- Hay que pintar de atrás hacia delante los objetos transparentes.
- Y desactivar la actualización del buffer de profundidad.



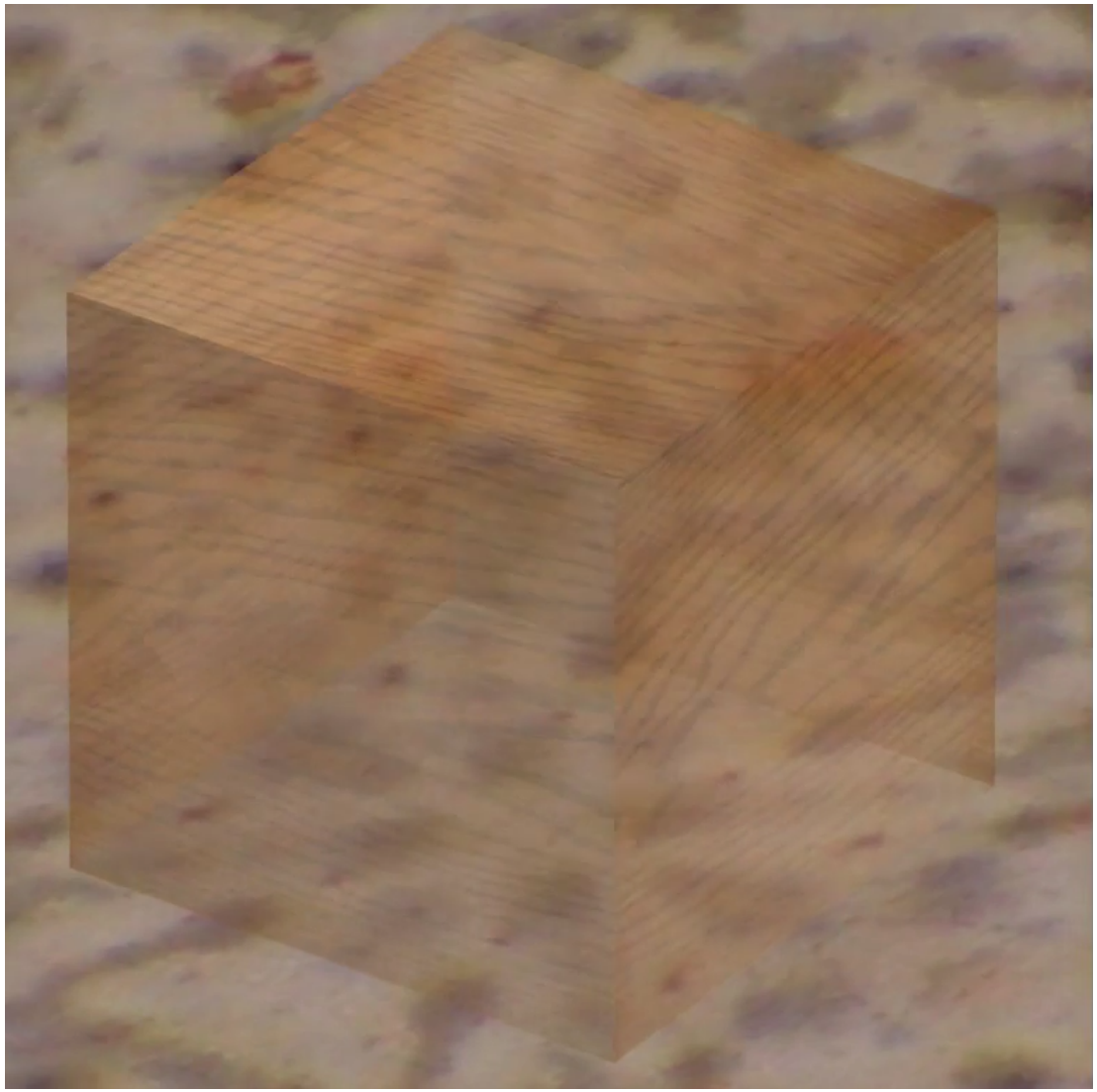
Transparencia

¿Y qué pasa con los objetos cerrados?

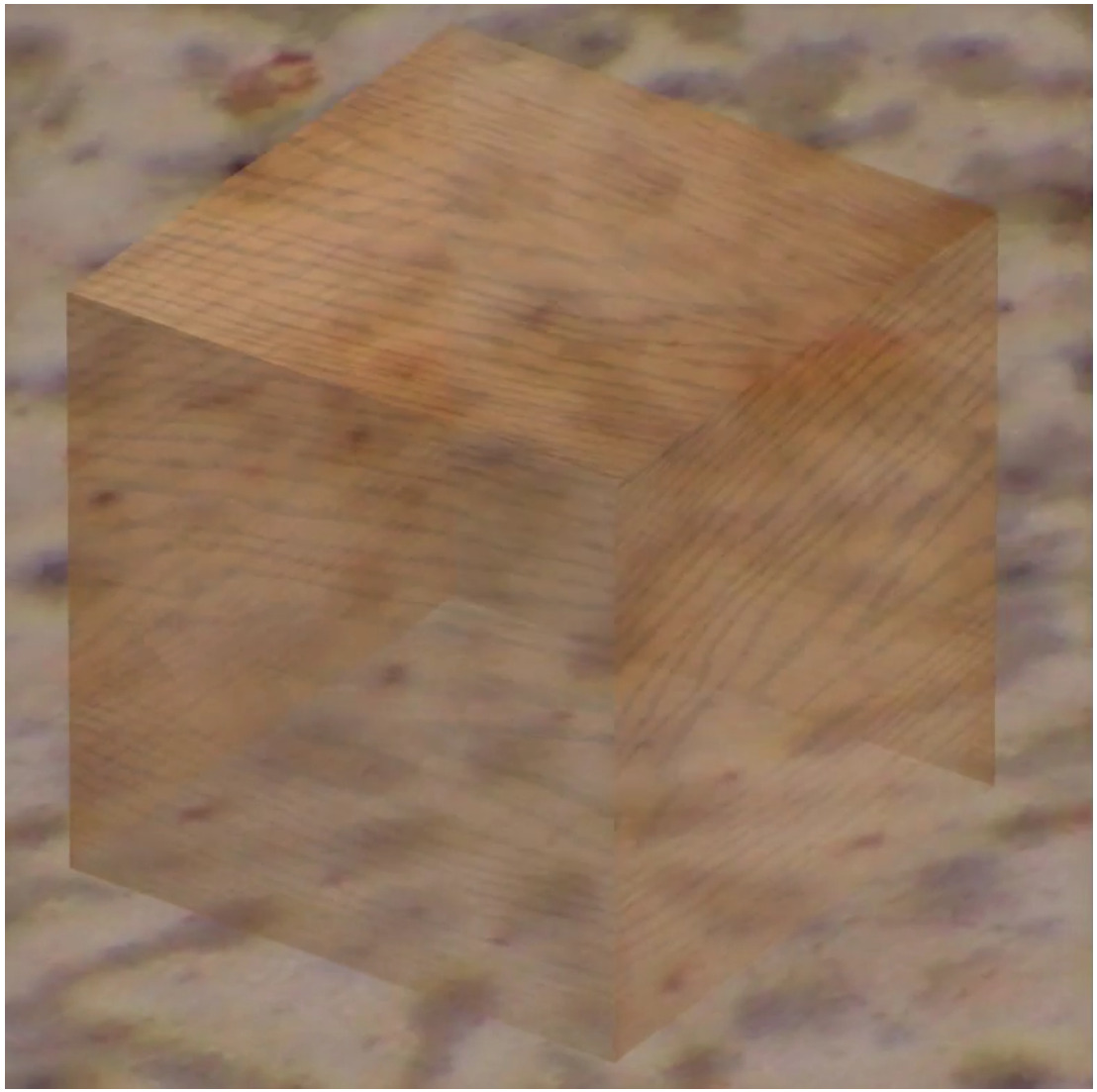
- ¿Deberían verse las caras traseras? Tenemos tres posibilidades
  - No hacer nada (izqda).
  - Que se eliminen las caras traseras (centro).
  - Que se eliminen las caras delanteras, pintar, que se eliminen las caras traseras y pintar de nuevo (derecha).



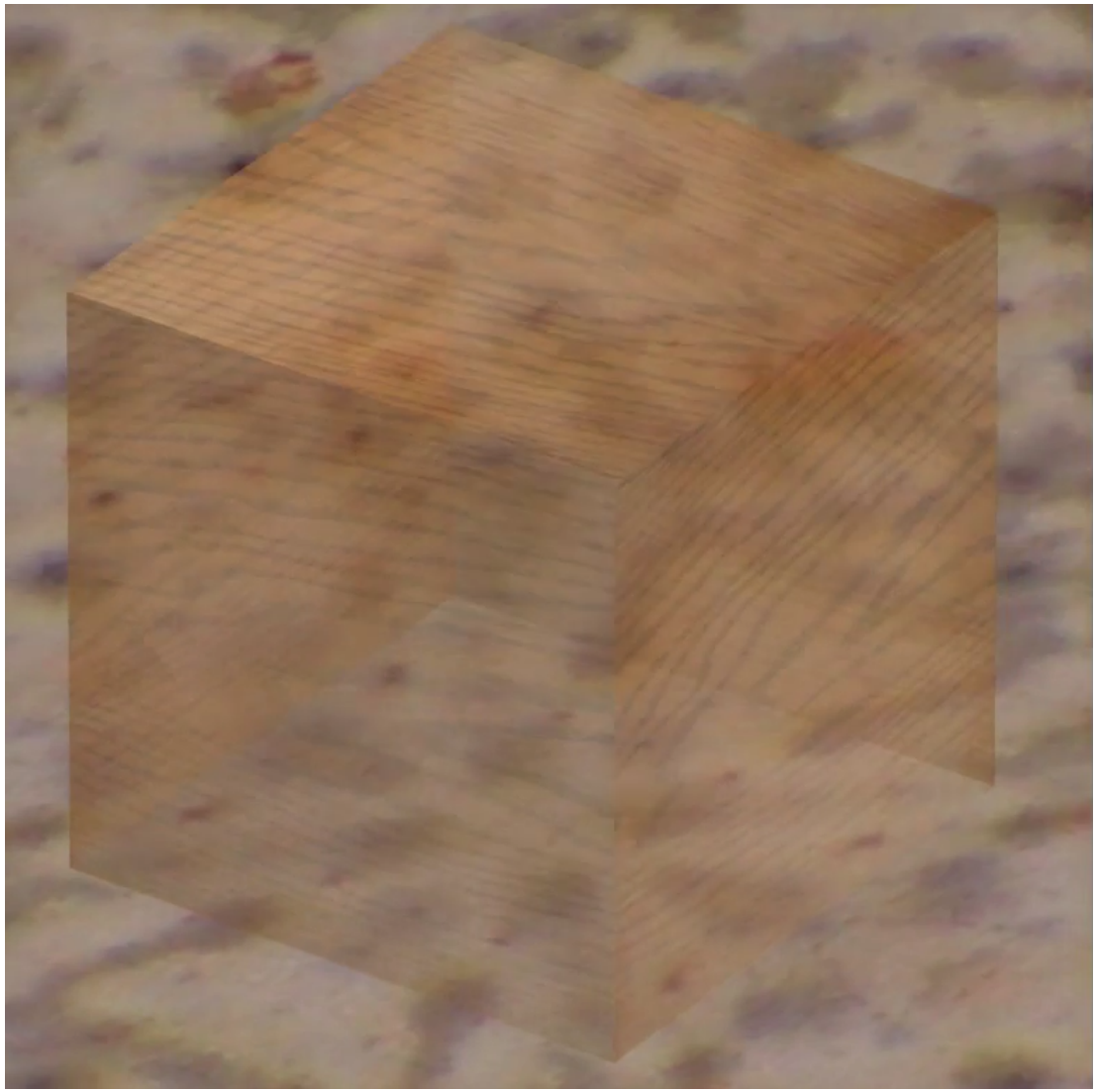
Transparencia, no hacer nada...



Transparencia, eliminar caras traseras...



Transparencia, ordenar el dibujado del objeto 2 veces...



Transparencias con WebGL

Listado 1: Objetos transparentes

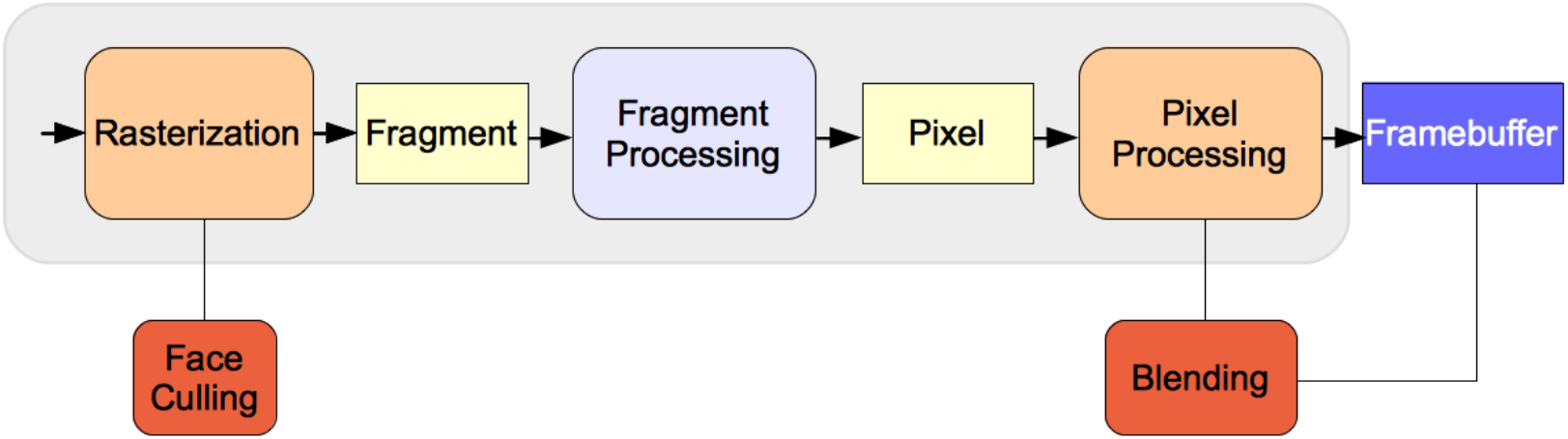
```
// Primero pintar los objetos opacos
....
// Despues los transparentes
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);  // operador over
gl.enable (gl.BLEND);                                // habilita la transparencia
gl.enable (gl.CULL_FACE);                            // habilita el face culling
gl.depthMask(false);

gl.cullFace(gl.FRONT);  // se eliminaran los triangulos cara al observador
drawSolid(exampleCube); // se han omitido sus transformaciones por claridad

gl.cullFace(gl.BACK);   // se eliminaran los triangulos de la parte trasera del objeto
drawSolid(exampleCube);

gl.disable(gl.CULL_FACE);
gl.disable(gl.BLEND);
gl.depthMask(true);
```

WebGL, ¿dónde ocurren las cosas?



Transparencia

Algunas pegas...

- La ordenación no siempre es trivial, por ejemplo, si un objeto transparente cruza a otro transparente ...
- Hay soluciones más complejas, pero también mucho más caras y no libres de problemas.
- No es un tema resuelto completamente en la GPU.

Cuestiones

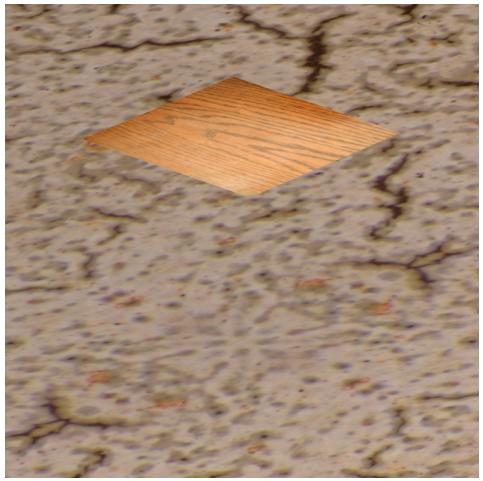
- En el cálculo de la transparencia mediante el operador over se utilizan dos valores de color, ¿cuáles son?
- En WebGL 2.0, ¿cómo se establece la transparencia a nivel de fragmento? (solo hay una respuesta correcta)
  1. Utilizando la variable especial gl_BlendFactor.
  2. Utilizando la cuarta componente en fragmentColor con valores en el rango [0..1].
  3. Utilizando la coordenada homogénea de cada vértice.
  4. En el fragment shader, calculando el resultado del operador over y asignándolo a fragmentColor.

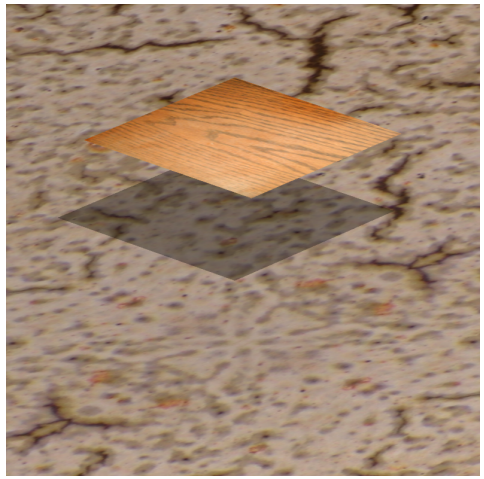
Hoy veremos...

Sombras

¿Por qué son importantes?

La ausencia de sombras en la escena es algo que, además de incidir negativamente en el realismo visual de la imagen sintética, nos dificulta de manera importante su comprensión.



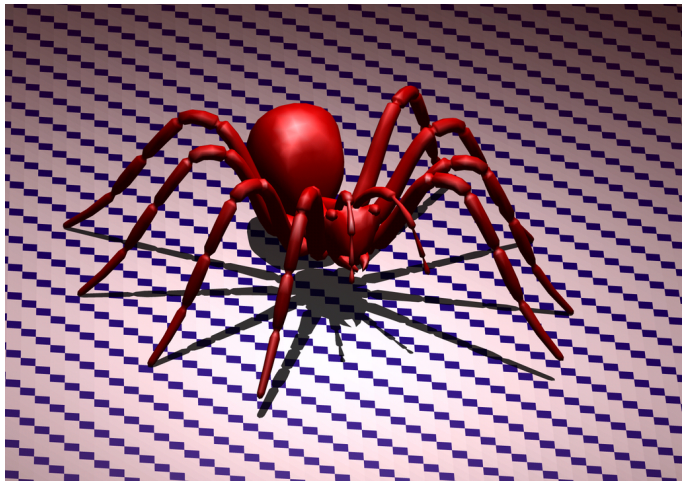


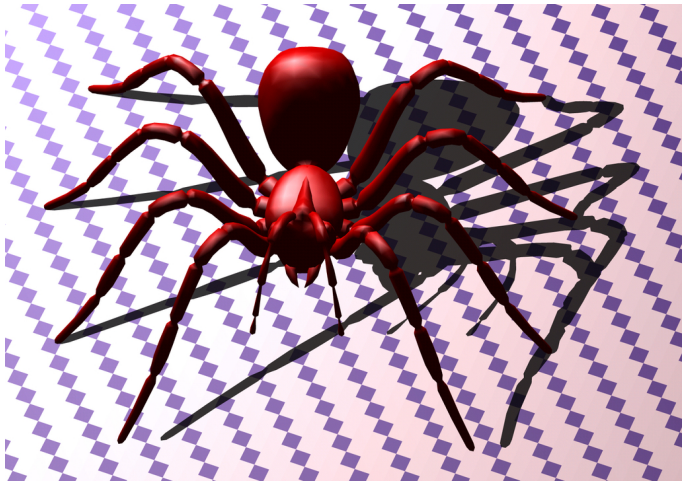
Prácticamente cualquier método que nos permita añadir sombras, por sencillo que sea, puede ser más que suficiente para aumentar el realismo y que el usuario se sienta cómodo al observar el mundo 3D.

Sombras proyectivas

Descripción

- Consiste en obtener la proyección del objeto situando la cámara en el punto de luz y estableciendo como plano de proyección aquel en el que queramos que aparezca su sombra.
- El resultado de la proyección, al que llamamos objeto sombra, se dibuja como un elemento más de la escena pero sin propiedades de material ni iluminación, simplemente de color oscuro.





Sombras proyectivas

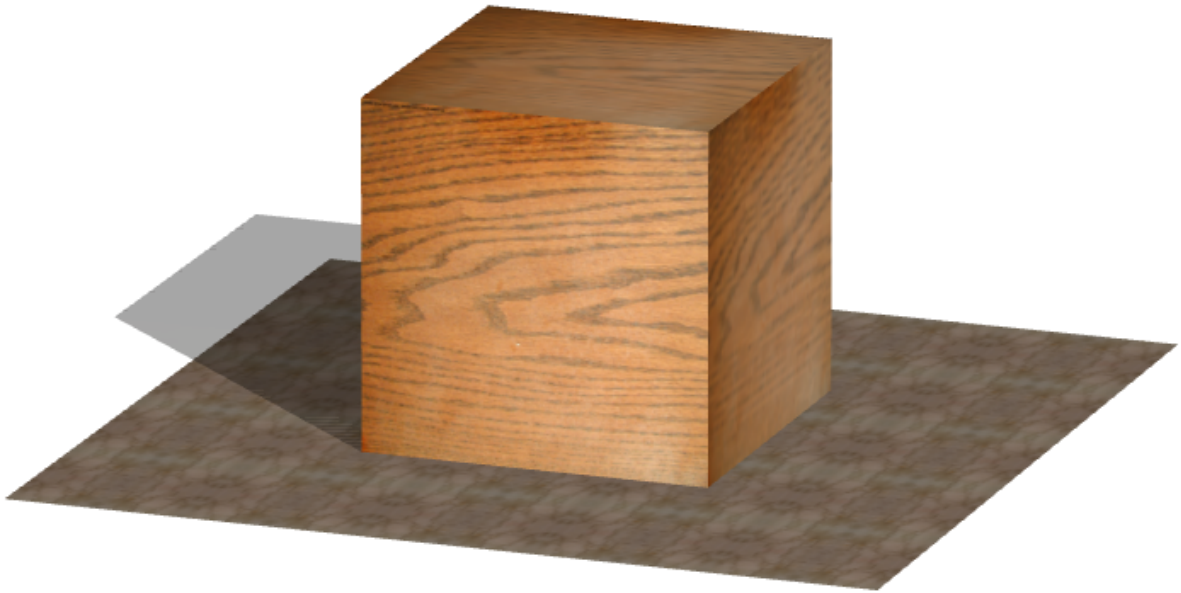
Dada una fuente de luz, L_P, y un plano de proyección, $N \cdot x + d = 0$, la matriz de proyección M es la siguiente.

$$M = \begin{pmatrix} N \cdot L_P + d - L_{P_x} N_x & -L_{P_x} N_y & -L_{P_x} N_z & -L_{P_x} d \\ -L_{P_y} N_x & N \cdot L_P + d - L_{P_y} N_y & -L_{P_y} N_z & -L_{P_y} d \\ -L_{P_z} N_x & -L_{P_z} N_y & N \cdot L_P + d - L_{P_z} N_z & -L_{P_z} d \\ -N_x & -N_y & -N_z & N \cdot L_P \end{pmatrix} \quad (2)$$

Sombras proyectivas

Este método presenta una serie de problemas

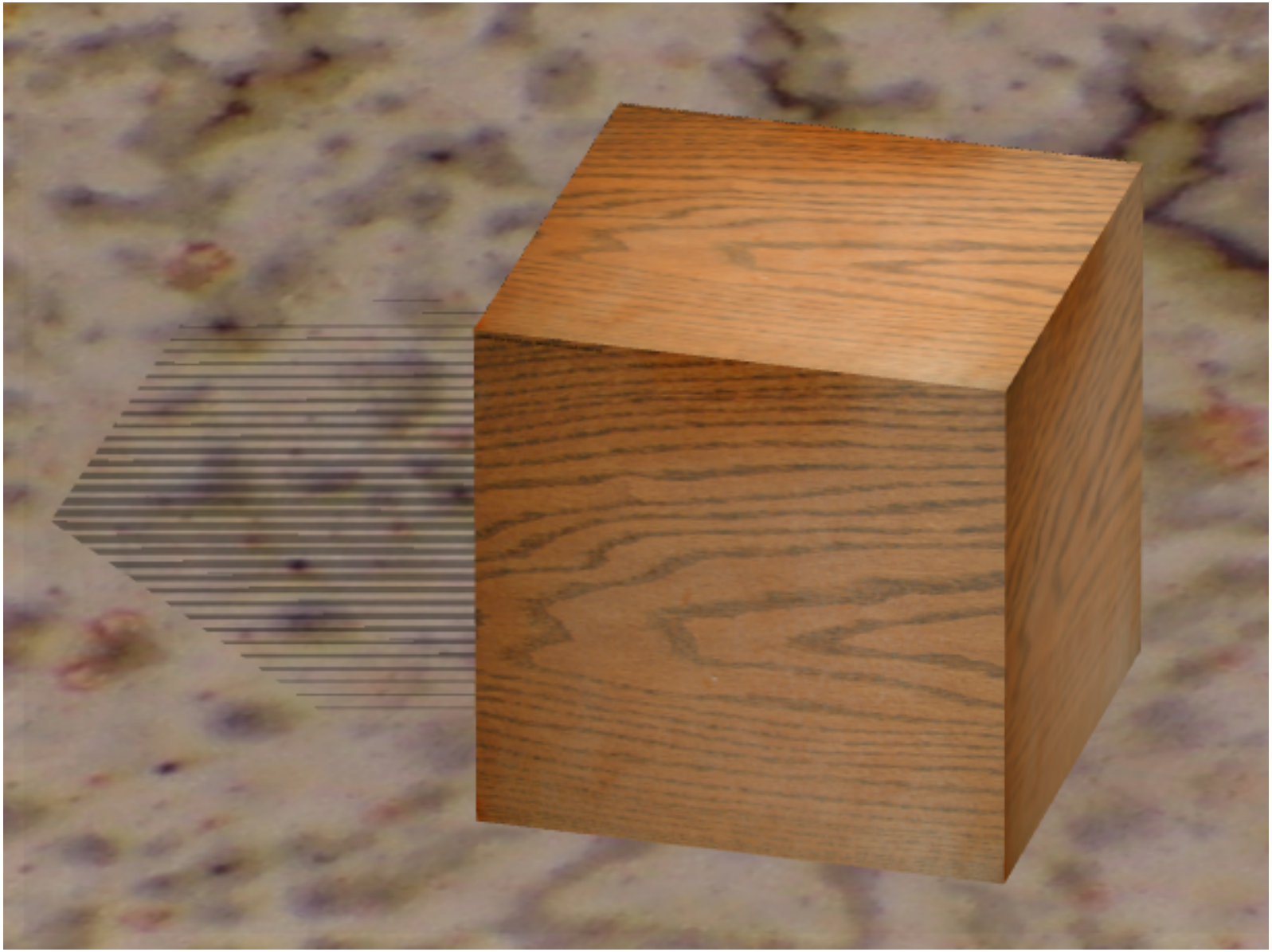
- Hay que controlar que el objeto sombra no vaya más allá de la superficie sobre la que recae.
- Ten en cuenta que la sombra es un objeto más
- Utiliza, por ejemplo, el buffer de plantilla para resolver el problema.



Sombras proyectivas

Este método presenta una serie de problemas

- Las sombras son muy oscuras, pero utilizando transparencia se puede conseguir un resultado mucho más agradable al dejar entrever el plano sobre el que se asientan.
- Muy complicado para superficies curvas o sombras de un objeto sobre él mismo.
- Como el objeto sombra es coplanar con el plano que se ha utilizado para el cálculo de la proyección, habría que añadir un pequeño desplazamiento a uno de ellos para evitar el efecto conocido como *stitching*. WebGL proporciona la orden *gl.polygonOffset* para especificar el desplazamiento que se sumará a la profundidad de cada fagmento siempre y cuando se haya habilitado con *gl.enable(gl.POLYGON_OFFSET_FILL)*.



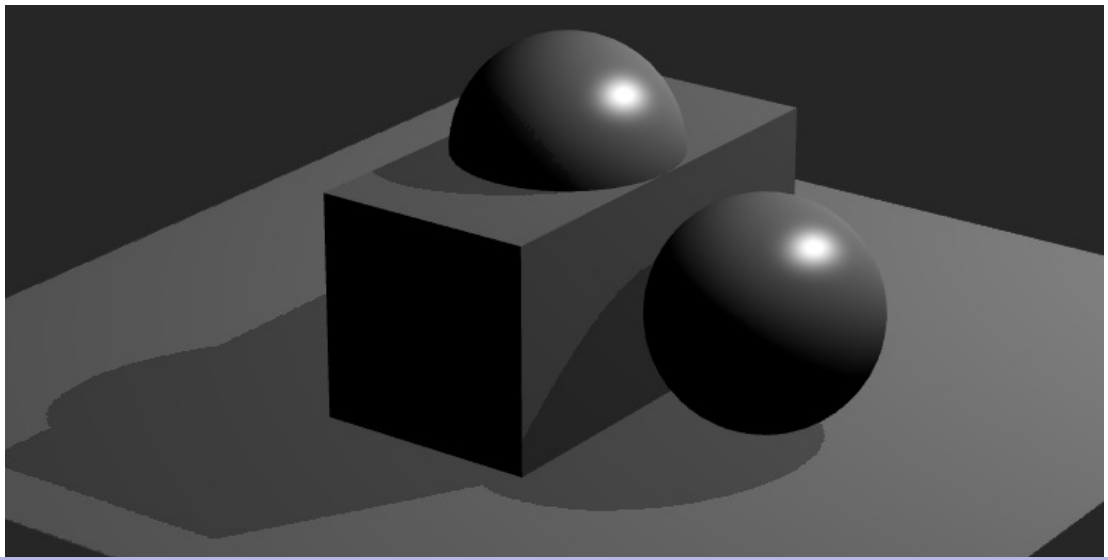
Cuestiones

- Dada una escena que contiene una única fuente de luz y dado un objeto que pertenece a dicha escena, si se hace uso del método de sombras proyectivas ¿cuántas veces se ha de obtener y dibujar el correspondiente objeto sombra?
- Si quiero implementar el método de sombras proyectivas, ¿para qué necesito utilizar un buffer de plantilla?

Shadow Mapping

Descripción

- Dibujar la escena desde el punto de luz.
  - Lo que la luz ve es lo que se ilumina.
  - Si la luz no ve la superficie, es que está en sombra.
- La profundidad se almacena en una textura.
- La escena se vuelve a dibujar desde el punto de vista de la cámara y se consulta la textura para saber si está o no en sombra.



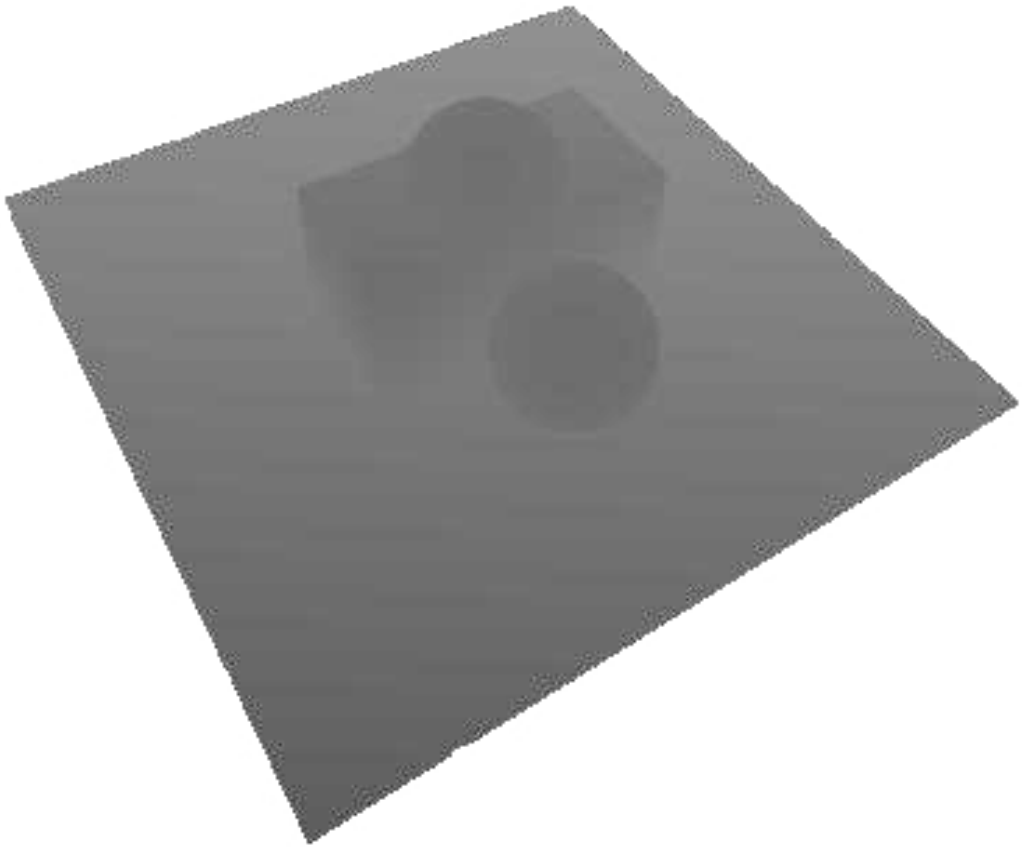
Almacena el buffer de profundidad en una textura

Framebuffer Object

- Un FBO permite crear framebuffers definidos por el usuario.
- La GPU puede dibujar pero su contenido no aparece en pantalla (off-screen rendering).
- Necesitas crear un buffer para almacenar la profundidad al pintar la escena.
- Y crear un objeto textura.
- Al dibujar, en el fragment shader, no necesitas asignar color al fragmento.

Resultado de la textura

- Los valores más oscuros son los más cercanos.

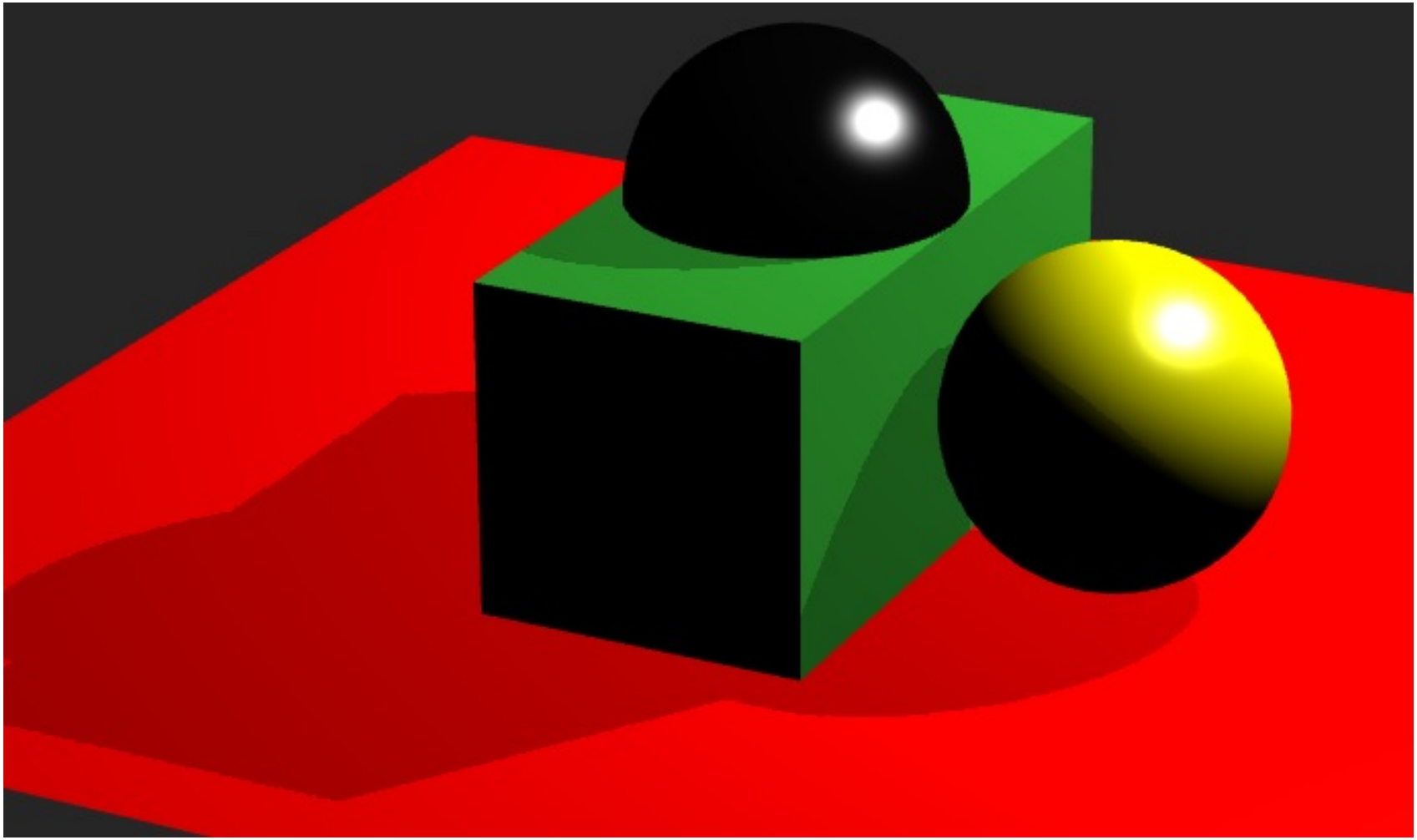


Dibujado

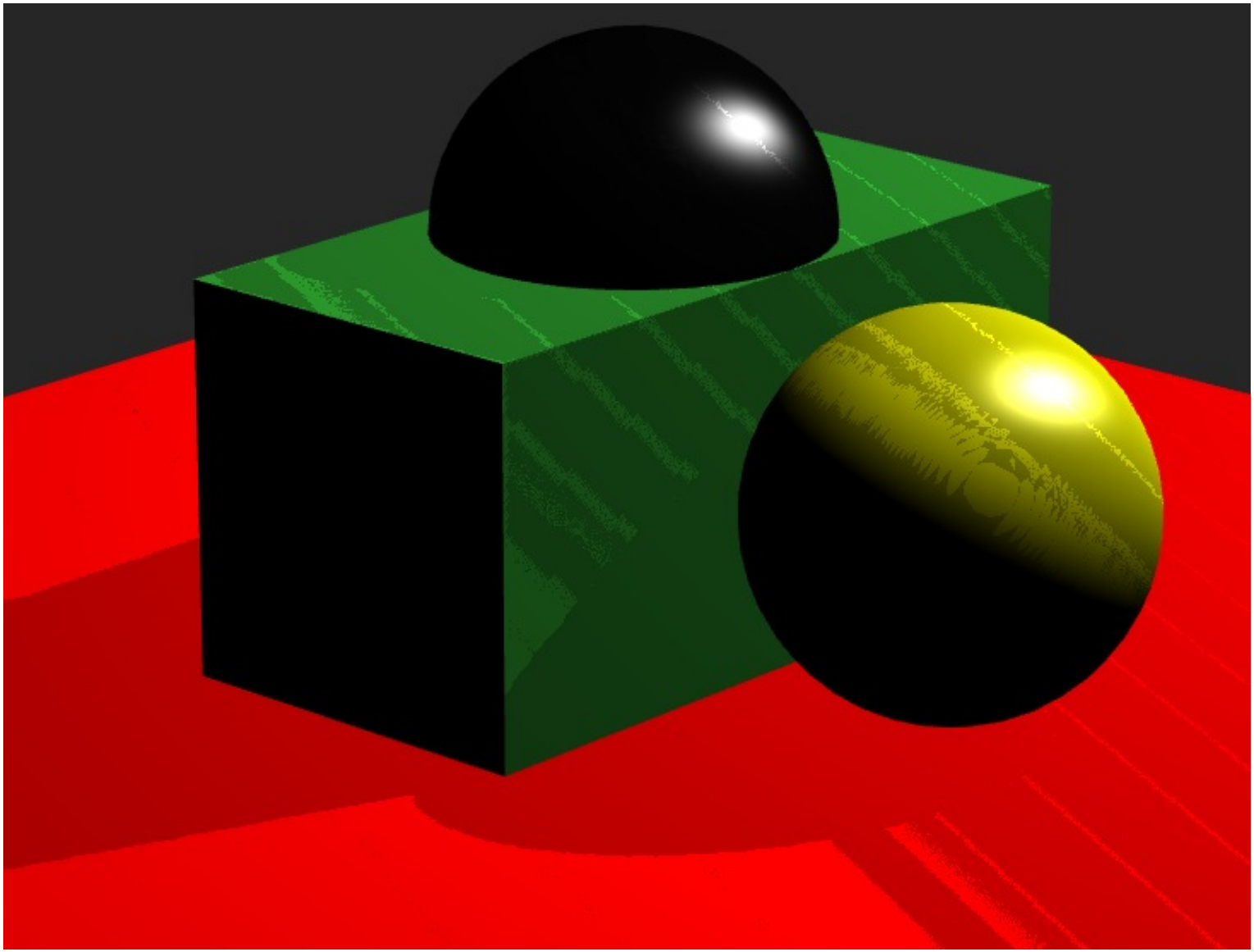
En drawScene()

- Se establece como destino el FBO creado para la ocasión.
- Se dibuja la escena vista desde la posición de la fuente de luz. La info de profundidad se almacena en la textura asociada al FBO.
- Se establece como destino el framebuffer por defecto.
- Se pinta la escena con tu shader preferido, pero además:
- En el vertex shader, necesitas otra variable que almacene el resultado de multiplicar cada vértice por la matriz de transformación correspondiente a la fuente de luz. Y:
  - división perspectiva,
  - y normalizar el resultado entre 0 y 1.
- En el fragment shader:
  - utiliza *textureProj* para acceder a la textura (de tipo *sampler2DShadow*),
  - que te dirá si el fragmento está en sombra o no.

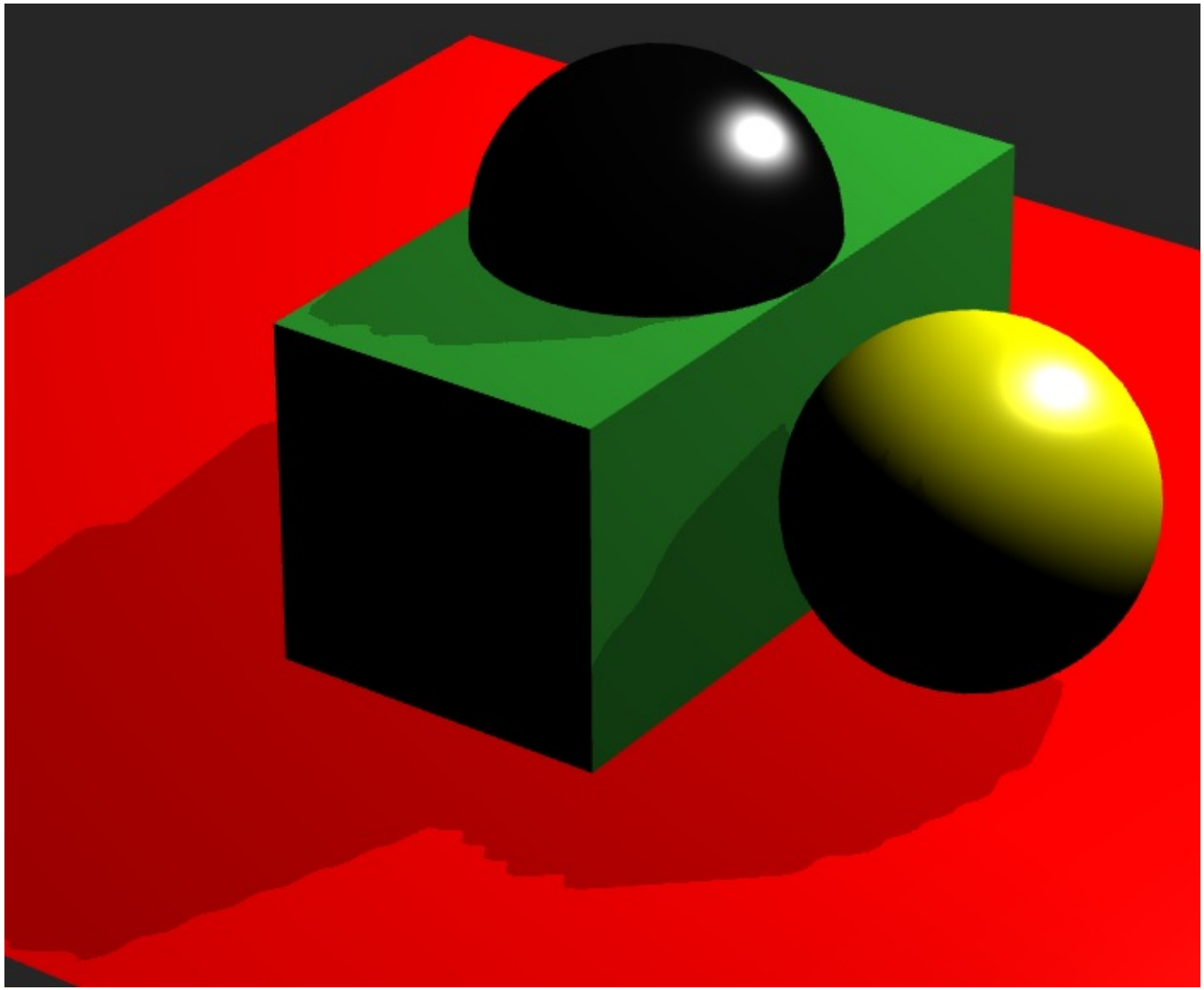
Ejemplo de resultado



Problemas - depth bias

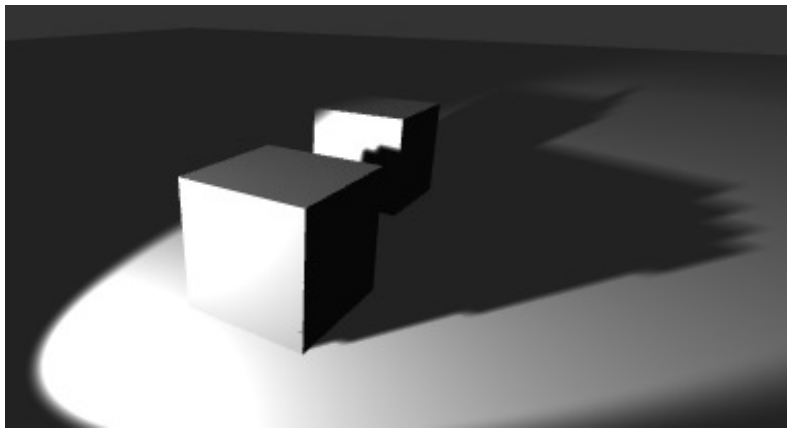


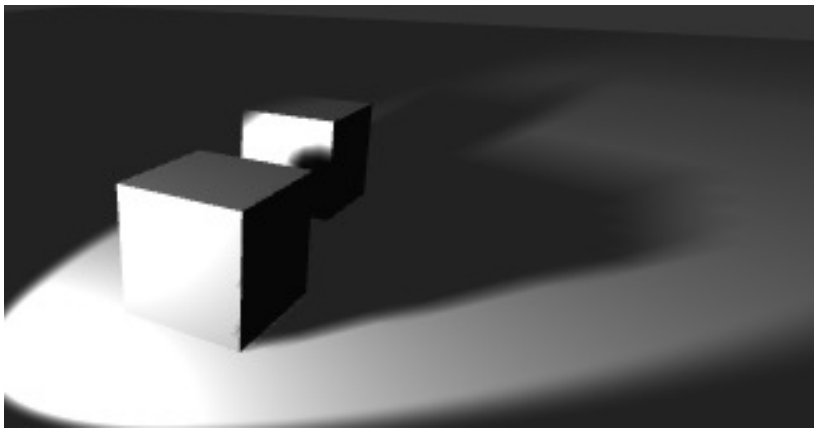
Problemas - texture resolution



Percentage-closer filtering

- Muestrea el área de la textura alrededor del fragmento, ¿cuánto?.
- ¿Qué porcentaje de texeles están en sombra?
- Utilizar el resultado como valor de atenuación de la luz.
- Se emborronan los límites de las sombras.





Cuestiones

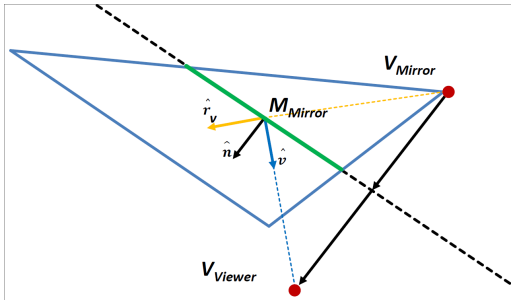
- En el método de Shadow mapping se genera una vista desde la fuente de luz. En concreto, ¿qué información se trata de conseguir? ¿dónde se almacena?
- ¿Qué objetivo tiene el método Percentage-closer filtering, PCE, cuando se utiliza junto con el método shadow mapping?

Hoy veremos...

Espejos

Descripción

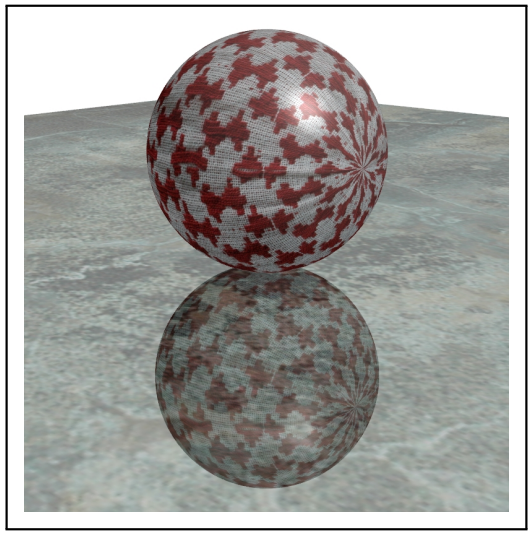
- Ya vimos como simular reflejos sobre superficies curvas.
- Para una superficie plana, lo puedes obtener así:
  - Coloca tu espejo en la escena.
  - Sitúa una cámara detrás de tu espejo y dibuja la escena a una textura como ya viste antes.
  - La dirección de la cámara del espejo es el vector de reflexión de la cámara de la escena respecto a la normal del plano del espejo.
  - Dibuja tu escena desde tu cámara y utiliza la textura obtenida para el objeto espejo.



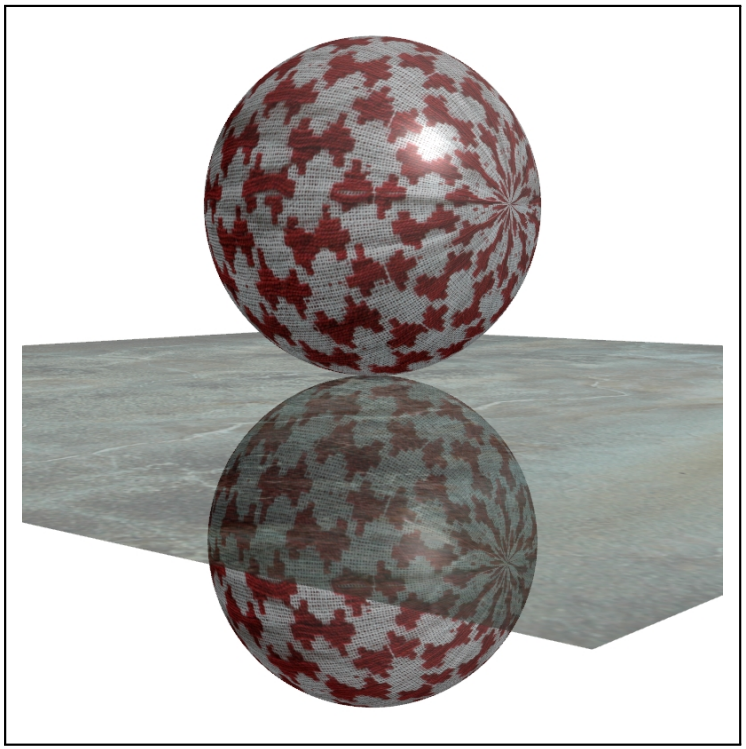
Espejo por simetría

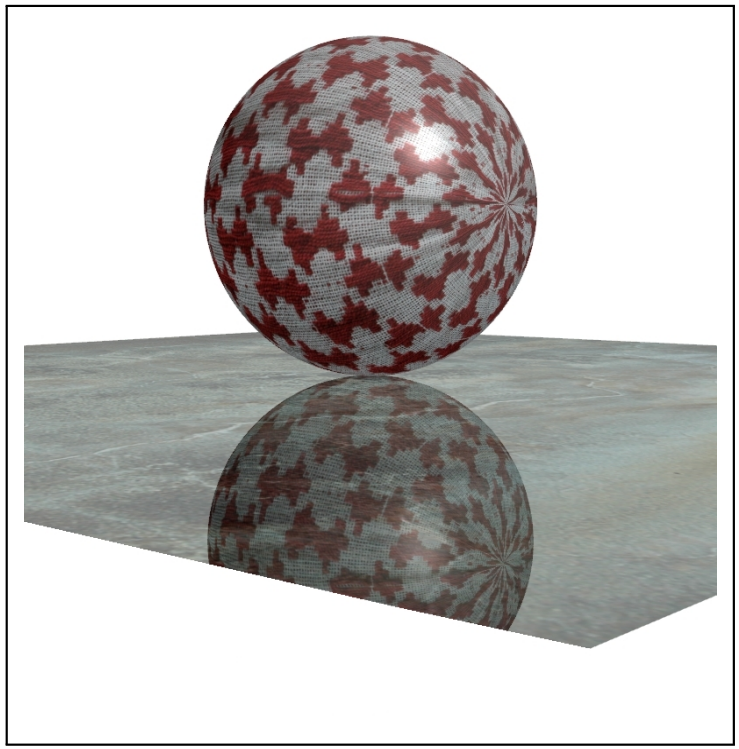
Descripción

- El método consiste en dibujar la escena de forma simétrica respecto al plano que contiene al objeto reflejante.
- Conlleva dos tareas extra:
  - Obtener la transformación de simetría.
  - Evitar que la escena simétrica se observe fuera de los límites del objeto reflejante.



Límites

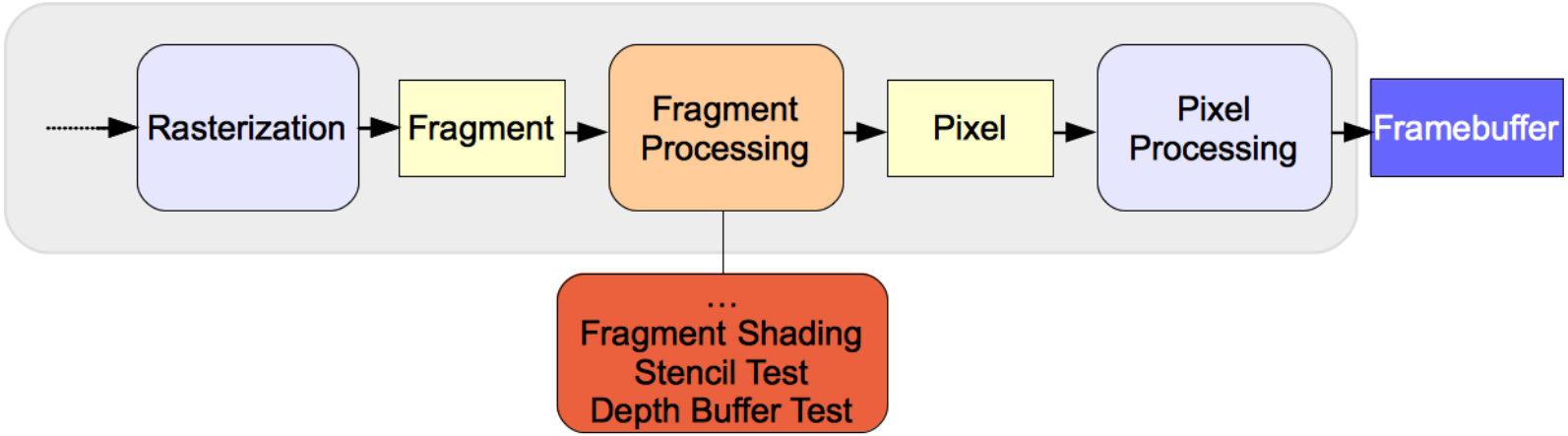




Espejo por simetría

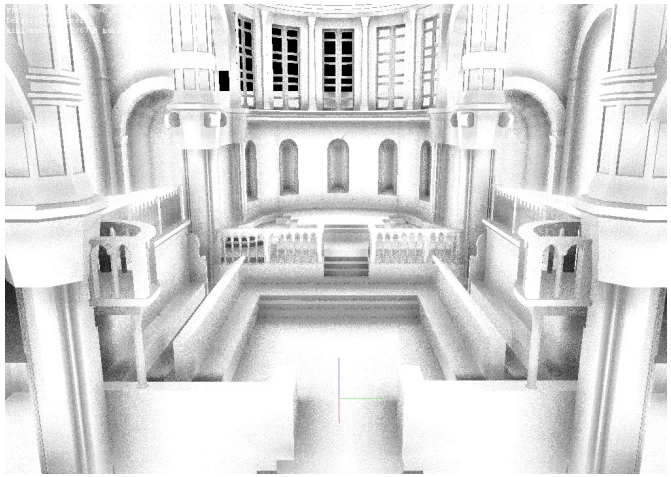
Procedimiento

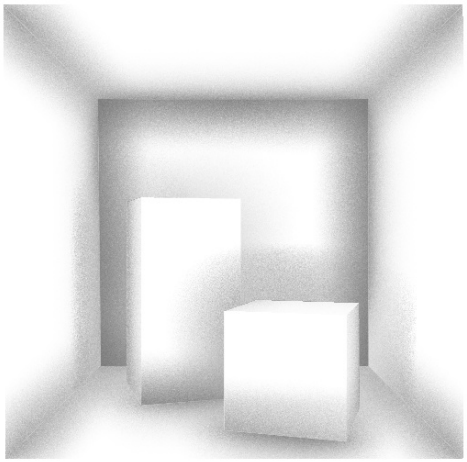
- Utiliza un buffer de plantilla y dibuja sobre él el espejo.
- Con el test de plantilla activado, dibuja la escena reflejada.
- Desactiva el test de plantilla.
- Dibuja la escena normal.
- Dibuja el espejo con transparencia.



Hoy veremos...

Ambient Occlusion





Descripción

- El modelo de Phong utiliza un valor constante para la iluminación ambiente.
- La cantidad de luz ambiente que alcanza un punto de una superficie depende de su entorno.
- Por ejemplo, un punto de una pared cerca de una esquina recibe menos iluminación que uno de enmedio.

Ambient Occlusion

Descripción

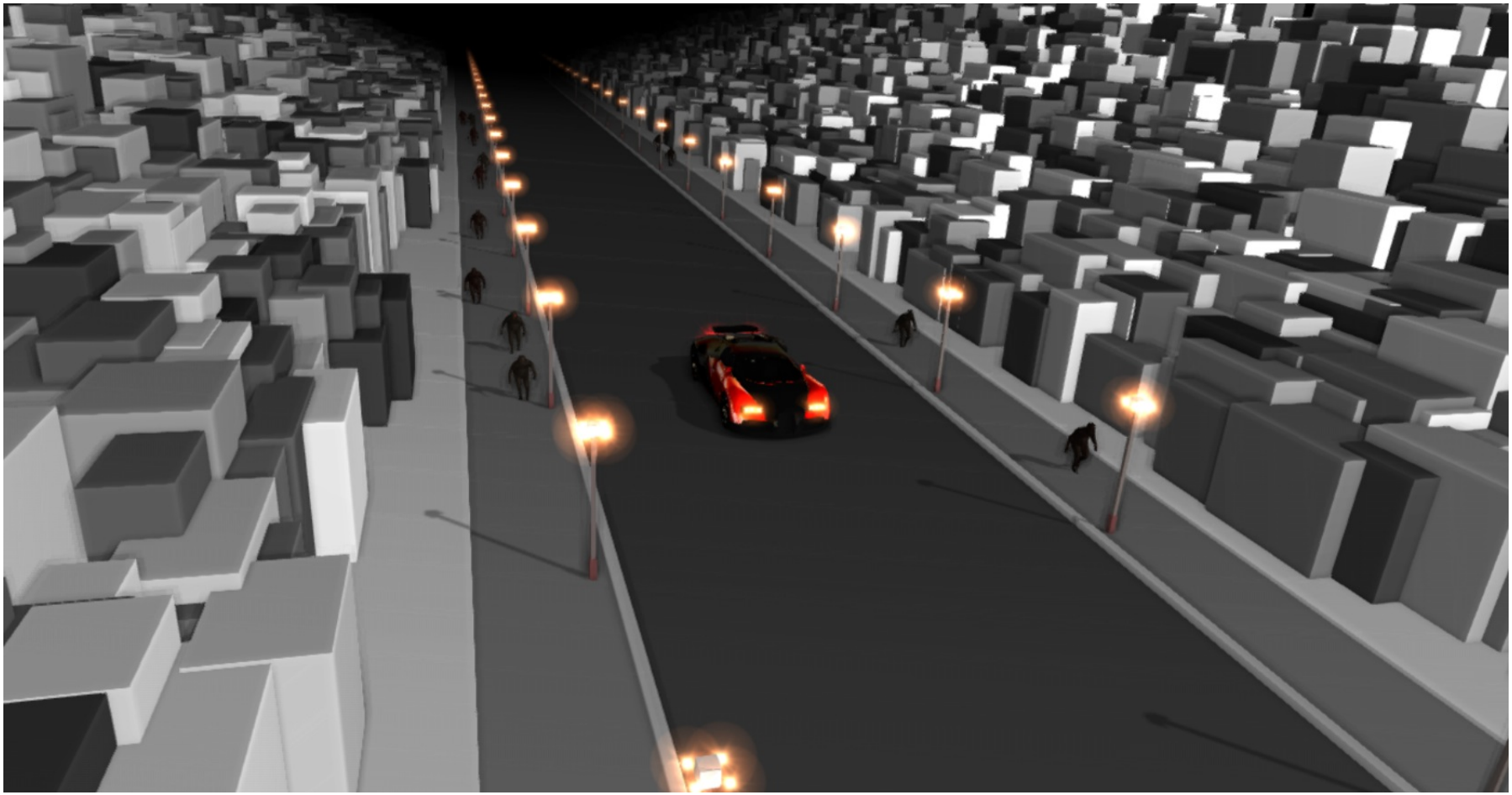
- Ambient Occlusion simula ese efecto.
- La idea es calcular la accesibilidad de cada punto de la superficie y atenuar su iluminación.
- La accesibilidad es una medida de cuánta luz ambiente puede alcanzar un punto sin ser bloqueada por superficies cercanas.
- Se puede calcular lanzando rayos desde el punto y comprobar si intersectan en una distancia próxima.

Ambient Occlusion

Screen Space Ambient Occlusion

- Ideal para ser implementada en tiempo real.
- Para un pixel, la cantidad de oclusión producida por sus vecinos se obtiene a partir de las diferencias de profundidad.
- En primer lugar, se dibuja la escena y en un FBO se almacena normal y profundidad por píxel.
- Entonces se calcula la oclusión por píxel:
  $max(0, dot(n, v)) * (1{,}0/(1{,}0 + d))$
- Los vecinos se muestrean de forma aleatoria.
- El resultado se almacena en un nuevo FBO para ser suavizado.

Ambient Occlusion



http://alteredqualia.com/three/examples/webgl_postprocessing_ssao.html