

Visualización: Materiales

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2019/2020

1. Introducción

Tras modelar una escena en cuanto a la geometría de los elementos que intervienen en ella, es necesario generar una imagen y mostrarla. Recordemos que es el objetivo de la informática gráfica, la síntesis de imágenes, generar una imagen a partir de la información gráfica que se ha adquirido o se ha producido. Un sistema gráfico, por tanto, no se queda en la creación o adquisición de información gráfica y su representación, sino que un módulo importante del sistema es la generación de la imagen que muestra al usuario la información gráfica que gestiona.

Situémosnos en el mundo real, figura 1, ¿qué ocurre cuando alguien capta una imagen, digital o analógica, de una escena?

Todo comienza en la luz y cómo reacciona ésta con los objetos según el material con el que estén hechos. Cuando un rayo de luz incide en un punto de un objeto, parte de esa luz incidente se refleja en todas las direcciones, es lo que se denomina **reflexión difusa**.

En el caso de la fotografía de la figura 1, de todos esos rayos reflejados desde un determinado punto del edificio, algunos de ellos han llegado a la lente de la cámara y han sido conducidos a impresionar un pixel (en el caso de una imagen digital) del CCD¹ de la cámara con un determinado color. Y así con todos los puntos del edificio y todos los píxeles de la

¹Charged-Coupled Device. Dispositivo de carga acoplada. El soporte que es impresionado por la luz en una cámara digital.



Figura 1: Para crear una imagen por ordenador se toma como referencia el mundo real

imagen. De manera que finalmente, esa matriz de píxeles de colores que es la imagen captada, hacen percibir una aproximación bastante buena de la realidad. Se muestra un fragmento en la figura 2.



Figura 2: Cada píxel en una imagen digital ha captado una porción del mundo (**proyección**) y su color representa parte de los procesos que han ocurrido entre la luz y los materiales en la escena (**fotometría**).

Se tiene, por tanto, que una parte del proceso de captación de una imagen de una escena es un trabajo de geometría. Una proyección de un escenario 3D a una imagen 2D, y en ese proceso hay que contemplar la proyección en sí misma de cada objeto, la distorsión con la que se perciben los objetos debido a la perspectiva (las zonas más lejanas se proyectan más pequeñas en la imagen que las más cercanas) y el hecho de que unos objetos ocultan a otros.

Cualquier algoritmo que genere una imagen 2D a partir de una escena 3D debe ocuparse de la proyección, la perspectiva y la ocultación.

Pero otra parte del proceso es qué ha ocurrido con la luz y los materiales antes de que esos rayos salieran de ese punto del edificio hacia la lente de la cámara.

Visualización: Materiales

Decíamos que cuando un rayo de luz incide en un punto de un objeto, parte de ese rayo se refleja en todas las direcciones, por tanto cada punto de la escena se convierte en una fuente de **luz indirecta**. La fuente de **luz directa** en el ejemplo sería el Sol. De manera que en cada punto del edificio están incidiendo múltiples rayos de luz, en algunos puntos (donde no se aprecian sombras) incide la luz directa del Sol, y en todos los puntos inciden la luz indirecta procedente de los otros puntos del escenario, incluido el cielo, que también recoge y transmite parte de la luz que recibe del Sol.

¿Por qué razón no se ven absolutamente negras las zonas en sombra, si no le da la luz directa del Sol? Porque también están iluminadas, aunque sea con una menor intensidad, por el resto del escenario que actúa como fuente de luz indirecta.

Pero según sea el material de un objeto, se producen otros fenómenos con la luz. Si el material está pulido, una gran parte de la luz que incide en un punto será reflejada en la dirección simétrica, **reflexión especular**. En la figura 3 se percibe la alta reflexión especular en el suelo y en los brillos del juguete (los brillos no son más que un reflejo de la fuente de luz).

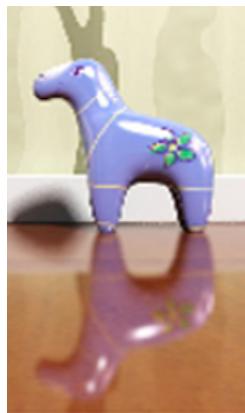


Figura 3: La componente especular de un material puede hacer que los objetos se comporten en parte como espejos.

Por otro lado, en los objetos que son de un material parcialmente transparente (se hablaría de **opacidad o transparencia**, un concepto es el opuesto al otro), parte de la luz que incide en un punto será transmitida a través del objeto posiblemente desviando su trayectoria (**refracción**), o generando rayos transmitidos a través del objeto en todas las direcciones (**translucencia**).

Como puede deducirse, un rayo de luz que procede de una fuente de luz como el Sol o una bombilla, recorre un largo camino por diversos objetos antes de llegar a un píxel. En ese camino, en cada impacto con un objeto se va desdoblando en otros rayos que recursivamente siguen su propio camino, con más impactos y desdoblamientos. Cada uno de esos innumerables rayos van *recogiendo* información de los objetos y materiales por donde van pasando, de manera que el color de un píxel no es más que un reflejo de toda la información recogida por

todos los rayos que llegan a ese píxel.

Si decíamos que una parte del proceso de generación de una imagen eran cálculos de geometría, otra parte importante del proceso son cálculos de fotometría.

1.1. Modelo de iluminación

Reproducir todo ese proceso de geometría y fotometría en un ordenador es imposible. Solo baste decir que cuando decimos que la reflexión difusa se produce en todas las direcciones, nos referimos a infinitas direcciones.

Por tanto, para reproducir en un ordenador dicho proceso de generación de una imagen a partir de una escena virtual (**rendering**), se tienen que elegir un subconjunto finito de entre todas las propiedades físicas de la luz y de los materiales que intervienen en el proceso. A ese conjunto de propiedades elegidas, junto con el modo de procesarlas, es a lo que se denomina **modelo de iluminación**.

En función del modelo de iluminación elegido se obtendrán imágenes más cercanas o alejadas a lo que captaría una fotografía. Y como os podéis imaginar, a mayor cercanía en cuanto a calidad a una fotografía, más tiempo de cálculo será necesario para generar una imagen de ese tipo.

Si el modelo elegido, por ejemplo, no contempla la representación o procesamiento de reflejos especulares, no se podrán mostrar reflejos o brillos en una imagen, aunque sea rápida de generar. La calidad de una imagen y el tiempo empleado en generarla depende de lo completo que sea el modelo de iluminación elegido.

1.2. Materiales

La imagen de la figura 4 muestra diversos modos de reflexión o transmitancia de la luz en un objeto según su material

Si bien antes se ha comentado el camino que recorre un rayo de luz desde que sale de la fuente de luz hasta que llega al ojo del observador, que llamaremos *rayo luminoso*, ese camino también puede recorrerse al revés: se parte del ojo del observador hacia la escena y se

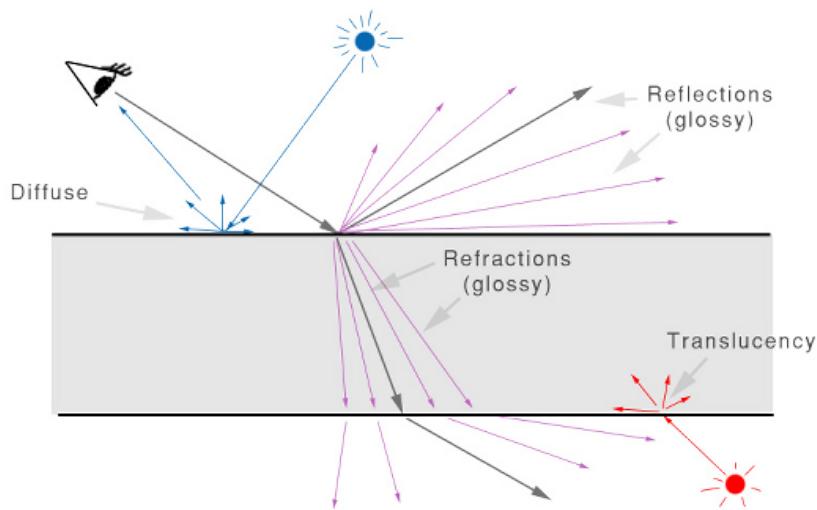


Figura 4: La luz puede reaccionar de muy diferentes formas con un objeto según sea su material.

va recorriendo ésta hasta que en última instancia se llegue a la fuente de luz. A los rayos que se recorren de esta forma los llamaremos *rayos visuales*.

En la implementación de los algoritmos de cálculo en los modelos de iluminación se contemplan ambos tipos de recorrido aprovechando la información que cada tipo de rayo proporciona en los cálculos a realizar.

En la ilustración de la figura 4 se muestran gráficamente algunas de las posibles reacciones de los rayos con la superficie de un objeto. Se observa cómo un rayo luminoso (en azul) procedente de la fuente de luz al impactar en el objeto se refleja desdoblándose en múltiples rayos luminosos en todas las direcciones (reflexión difusa), uno de esos rayos llega al ojo del observador.

También se muestra como otro rayo luminoso (en rojo) que impacta en el objeto por la parte de atrás según está mirando el observador, atraviesa el objeto pero desdoblándose en múltiples rayos luminosos en todas las direcciones (traslucencia). Se considera que el objeto de la ilustración es parcialmente transparente.

También se muestra como un rayo visual (en negro) cuando impacta en el cuerpo se desdobra en dos rayos visuales (también en negro), reflexión especular y refracción, permitiendo a ese observador ver reflejado como un espejo lo que ocurre a un lado de ese cristal, al mismo tiempo que ve refractado lo que se encuentra al otro lado del cristal, como en el ejemplo de la figura 5.

En un objeto con un material con una reflexión especular perfecta, el impacto de un rayo visual solo produciría un rayo visual simétrico reflejado (en negro en la ilustración), pero

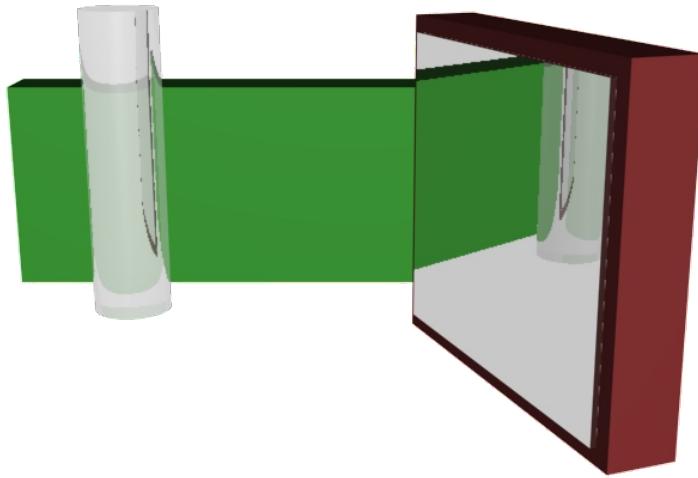


Figura 5: En el cilindro de la imagen se observa, cómo al mismo tiempo, se ve refractado el muro verde de detrás y reflejado el espejo de la derecha.

si la superficie no está perfectamente pulida lo normal es que el impacto de ese rayo visual produzca más de un rayo reflejado, reflexiones glossy, abiertos en abanico (rayos magenta en la ilustración), haciendo que ese observador vea reflejado en ese objeto lo que hay en ese lado, pero lo verá borroso, como en el ejemplo de la figura 6.



Figura 6: Las 3 copas metálicas reflejan el suelo en la parte de la copa que está justo encima del pie. La de la izquierda de una manera nítida, la del centro con menos nitidez y la de la derecha de una forma bastante borrosa.

El que una reflexión especular se vea más o menos nítida depende de cuán cerrado sea el ángulo de rayos glossy reflejados. Viéndose más borroso cuanto más abierto sea el ángulo.

Lo mismo ocurre con los rayos visuales de refracción. En un cristal de ventana normal, se ve muy nítidamente lo que hay al otro lado mientras que en un cristal como los que se ponen en la ventana de un cuarto de baño, se puede apreciar que hay una persona al otro lado pero

se ve de una forma muy borrosa. Esa nitidez depende de cómo de cerrado sea el ángulo de refracciones glossy que se generen.

Lo que se muestra son solo algunos ejemplos de características que pueden considerarse en un material que aplicado a un objeto y procesadas adecuadamente permiten generar imágenes con calidad cercana a una fotografía.

1.3. Componentes especular y difusa

En general, en un material se consideran dos componentes: La **componente especular**, figura 7.a, referente a los rayos reflejados en la dirección simétrica a la dirección desde donde impactó el rayo de entrada. Salen en un abanico más o menos cerrado y tienen una intensidad media-alta.

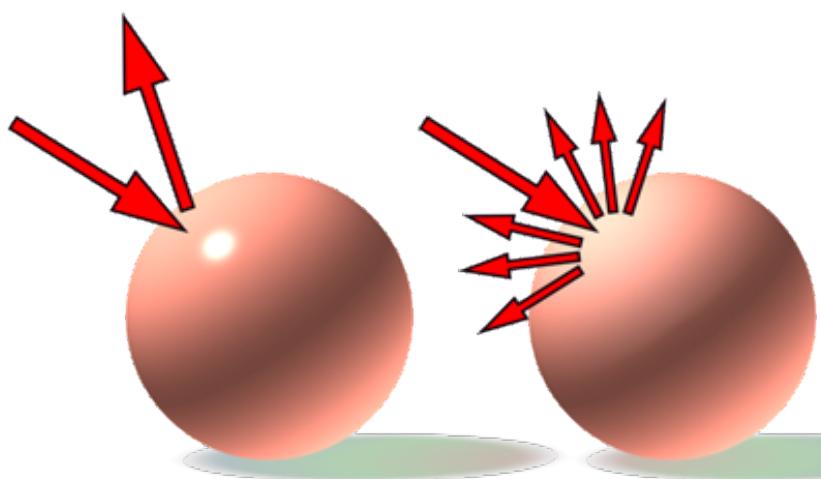


Figura 7: a) Reflexión especular

b) Reflexión difusa

La componente difusa, figura 7.b, se refiere a los rayos que son reflejados en todas las direcciones. Tienen una intensidad baja.

Todos los materiales tienen componente difusa y aquellos materiales que representan acabados pulidos, además tienen componente especular.

Un modelo básico de iluminación que considere ambas componentes, ¿cómo calcularía el color en un punto? (véase la figura 8)

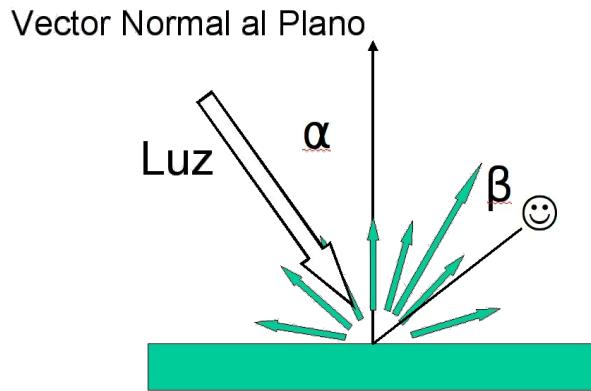


Figura 8: El cálculo de iluminación tiene en cuenta el vector normal del objeto en el punto donde se realizan los cálculos así como el ángulo de entrada del rayo luminoso y el ángulo de salida del rayo visual.

Lo primero es indicar que no sería correcto hablar de cálculo de color, si no de cálculo de energía luminosa, aunque finalmente esa energía luminosa se transforme en un color en el píxel.

Una ecuación básica de ese cálculo, que considera la energía añadida por componente difusa y la componente especular, sería

$$E_{out} = E_{in} \cdot k_d \cdot \cos(\alpha) + E_{in} \cdot k_e \cdot \cos^n(\beta)$$

donde:

- E_{in} es la energía del rayo luminoso de entrada
- E_{out} es la energía luminosa calculada en dicho punto y que llega al observador
- k_d, k_e, n son constantes que dependen del material
- α es el ángulo que forma el rayo luminoso de entrada con el vector normal del objeto en dicho punto
- β es el ángulo que forma el rayo visual del observador con el rayo luminoso reflejado en la dirección simétrica a la dirección de entrada

Aunque no se entre en los detalles de la ecuación, sí es importante que se comprendan los siguientes puntos:

- El vector normal del objeto en el punto de cálculo es fundamental. Nos indica cómo está orientada la superficie del objeto con respecto a la luz y al observador. Esas orientaciones, representadas en esta ecuación con los ángulos α y β afectan directamente a cómo reacciona la luz al impactar en dicho punto y a cómo lo percibe el observador.
- El material define parámetros, representados en esta ecuación con k_d , k_e , que normalmente representan porcentajes de reflexión. En la ecuación, indican que del 100 % de la luz incidente, un porcentaje k_d es reflejado de manera difusa y un porcentaje k_e es reflejado de manera especular.
- Derivado del punto anterior, la suma de todos los porcentajes de reflexión que intervengan en una ecuación no debe superar el 100 %, un objeto no puede reflejar más luz de la que recibe.
- La energía saliente en un punto es el resultado de sumar la energía saliente parcial según cada componente considerada.
- En el cálculo de cada componente se tiene en cuenta, de manera general, la energía entrante, los atributos del material que influyen en esa componente y los ángulos que afectan a dicha componente.
- El modelo de iluminación es la conjunción de estas 3 cosas: qué propiedades de la luz se consideran, qué propiedades de los materiales se consideran, cómo realizar los cálculos con todos esos elementos.

A partir de dichas generalidades, diferentes modelos de iluminación usarán distinta formulación y tendrán en cuenta diferentes atributos del material y diferentes mediciones en la escena, ángulos, distancias, etc.

Un modelo de iluminación tradicional, como es el modelo de Phong, considera una componente más, la **componente ambiental**. Este modelo no tiene en cuenta la iluminación indirecta que hay en la escena, o mejor dicho, no calcula la iluminación indirecta que recibe cada punto en la escena procedente del resto del escenario. La aproxima considerando que hay una *cantidad* de luz en el ambiente, constante en toda la escena, que es la que procedería de la iluminación indirecta. A esa cantidad de luz, constante en toda la escena, se le denomina **luz ambiental**.

Un modelo básico de material muy usado tradicionalmente en computación es el basado en el modelo de iluminación de Phong, donde un material quedaría definido por 3 colores y un escalar. Véase la figura 9.

La componente difusa se define mediante un color, es el color del objeto, también denominado **color difuso**.

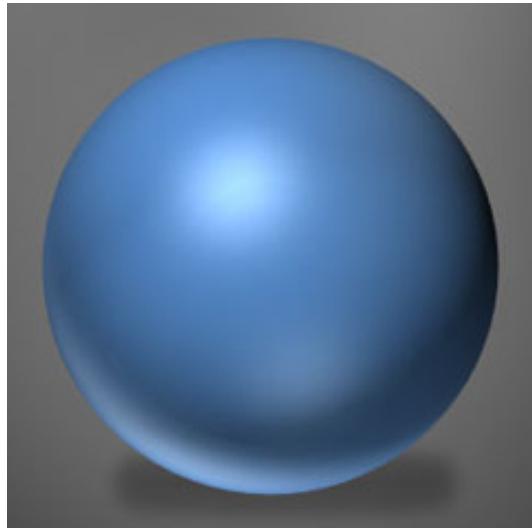


Figura 9: En esta esfera iluminada se aprecian las diferentes componentes del modelo de iluminación de Phong.

Las zonas de los objetos que no están iluminadas de manera directa no se ven negras, porque están iluminadas por la iluminación indirecta, que en el modelo de Phong es la luz ambiental. Lo normal es que esas zonas sean del mismo color del objeto pero más oscuras. Por tanto, la componente ambiental de un material con este modelo se define tomando como base el color difuso y oscureciéndolo. Sería el *color ambiental*.

Si el material está pulido pueden verse brillos en los objetos. Los brillos se corresponden con la componente especular, el observador ve el brillo porque está situado en la dirección simétrica a la dirección de incidencia de la luz, el brillo no es más que se está viendo reflejada la fuente de luz. Lo normal es que el color en esas zonas más brillantes también tengan como base el color del objeto, pero aclarado, muy cercano al blanco, el *color especular*.

Y en cuanto al escalar que forma parte de la definición del material, denominado *exponente de reflexión especular*, se trata de un valor para indicar como de intensos y extensos son esos brillos. Un valor más alto indica brillos pequeños en cuanto a la superficie que ocupan pero de una intensidad alta, figura 10.a, y un valor bajo produce brillos que en los objetos ocupan más superficie, pero de una intensidad menor, figura 10.b.

Otros materiales más complejos se definirán en base a definir más componentes, también nombrados en la literatura como canales. Se diría en ese caso canal difuso, canal especular, etc.

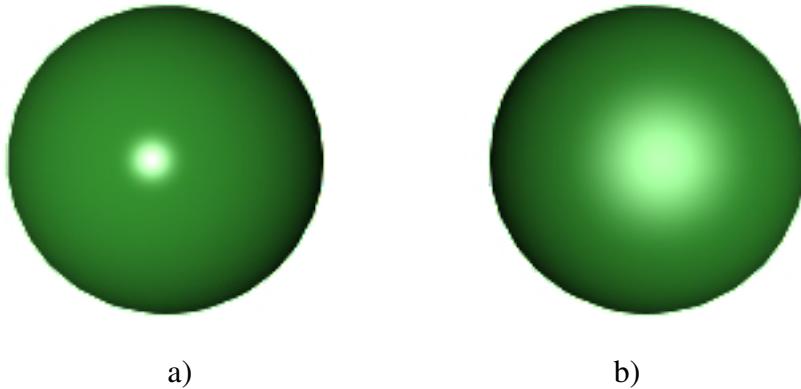


Figura 10: Un material basado en Phong incluye un valor para definir cómo son los brillos de ese material en cuanto a extensión e intensidad.

2. Materiales Three basados en un color

Dado que Three es una capa sobre WebGL, todo el proceso que implica la visualización se realiza en la GPU, la cual ejecuta programas shaders (o simplemente, shaders). El uso de shaders, por tanto, es obligatorio.

Un programa shader recibe toda la información necesaria involucrada en un modelo de iluminación, información sobre las luces, la geometría de los objetos, los materiales e implementa los algoritmos del dicho modelo de iluminación.

Three incorpora, de una manera transparente al programador, los programas shaders correspondientes a los modelos de iluminación que implementa. El desarrollador solo tiene que asignar materiales a los Mesh de su escena. Cada material tiene asociado su propio shader, de modo que cuando el renderer recorre el grafo de escena para realizar la visualización, cada nodo Mesh del grafo es procesado por la GPU con el shader que le corresponde según el material asignado a la geometría en ese Mesh.

No obstante, se pueden programar shaders propios y así realizar visualizaciones personalizadas.

2.1. La clase `Material`

Se trata de una clase abstracta, es la clase base para todos los materiales y define un conjunto de atributos que son comunes a todos ellos.

name: Permite identificar el material con un nombre

transparent: Boolean

- Si `false`, la figura será 100 % opaca
- Si `true`, la figura será transparente según el atributo de opacidad

opacity: Indica el nivel de transparencia con un valor entre 0.0 (invisible) y 1.0 (opaco)

visible: Boolean. Si se pone a `false` se le está indicando al visualizador que no procese los Mesh que tengan este material. Por lo tanto tiene el efecto de hacer esas figuras invisibles.

side: Indica en qué lados de una cara se aplica el material, sus valores posibles son las constantes `THREE.FrontSide`, `THREE.BackSide`, `THREE.DoubleSide`, con el significado que indica cada constante.

wireframe: Boolean. Se pone a `true` para ver una figura en modo alambre. Sea cual sea el material que tenga. Figura 11.

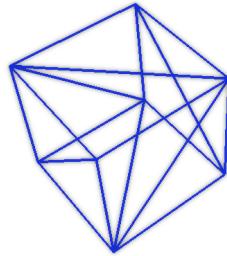


Figura 11: En este modo de visualización solo se muestran las aristas de la malla de polígonos.

shading: Permite configurar el modo en el que se colorean los polígonos. Con el valor `THREE.FlatShading` se hace un cálculo de color por polígono y todo el polígono se colorea de dicho color, notándose el facetado del objeto (figura 12.a); con el valor `THREE.SmoothShading` se realiza un cálculo de color por píxel con lo cual el objeto presenta un degradado suave en sus colores, no apreciándose el facetado de la malla de polígonos que la define (figura 12.b).

Hay que indicar que no es lo mismo poner el atributo `visible = false`; que poner los atributos `transparent = true`; `opacity = 0.0`;

Aunque en ambos casos se tiene el efecto de que la figura que tenga ese material no se ve en la visualización, hay una gran diferencia. En el primer caso, con el atributo `visible = false`; la figura no se ve porque no se ha procesado, es como si no estuviera en la escena.

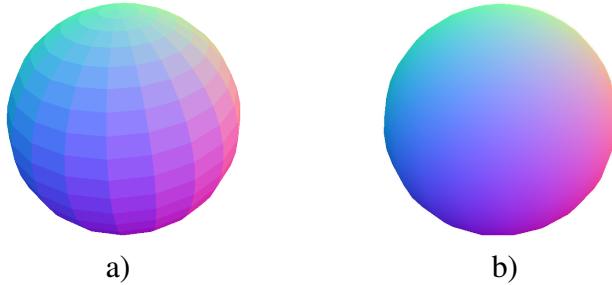


Figura 12: Aunque los objetos en Three están representados mediante B-rep y todos los objetos son poliedros de caras planas, con la técnica de sombreado suave se consigue la apariencia de estar formados por superficies curvas. Se muestra usando el material `MeshNormalMaterial`.

En el segundo caso la figura sí está aunque está con total transparencia y por tanto es invisible. Como se comentó en los apuntes sobre los motores de física, estas figuras que sí están pero no se las ve son útiles para poner ciertos límites en los movimientos del resto de figuras en una escena gestionada con un motor de física, ya que sí son detectadas en las colisiones.

En cuanto al modo alambre, además de que simplemente sea el material que se desea para una determinada figura, se puede usar para diferenciar temporalmente una figura del resto, por ejemplo para dar un *feedback* al usuario en un *picking*.

Modos de darle valores a los atributos

Existen dos modos de darle valores a los atributos de un material. Una forma es en el momento de instanciar la clase. En cuyo caso hay que pasarle al constructor un único parámetro de tipo diccionario con los atributos y valores que se quieran asignar. Y otra forma es, después de instanciar la clase, darle valores a los atributos del material que se ha instanciado.

```
var parametros = { color: 0xf763de, side: THREE.DoubleSide };
var material1 = new THREE.MeshPhongMaterial (parametros);

var material2 = new THREE.MeshLambertMaterial ({color: 0x123456, wireframe: true});

var material3 = new THREE.MeshPongMaterial ({color: 0xabcd});
```

2.2. La clase `MeshNormalMaterial`

Se trata de un material básico, una forma rápida de ponerle un material a una figura cuando todavía no se sabe qué material ponerle. Se crea con la clase `MeshNormalMaterial()`, no necesita parámetros ya que con los valores por defecto ya es suficiente en la mayoría de las ocasiones.

Mediante este material, a cada vértice de la geometría se le asigna un color que no es más que interpretar el vector normal en dicho vértice como si fuese un color.

Como se sabe, el vector normal es una coordenada 3D (nx, ny, nz) que indica la dirección del vector y además está normalizado, el módulo de dicho vector es 1. Por lo que se cumple que cada coordenada $nx, ny, nz \in [0, 1]$.

Así, si el vector normal en un vértice es (nx, ny, nz) , se establece que el color en ese vértice sea (r, g, b) donde $r = nx, g = ny, b = nz$.

El aspecto de este material es como se muestra en la figura 12.

2.3. La clase `MeshLambertMaterial`

Implementa el modelo de Lambert. Este modelo contempla las componentes ambiental y difusa. Al no tener la componente especular los **materiales** resultantes son **mate**, no muestran brillos. Pueden mostrar tonalidades claras del color del objeto según le incida la luz y según la orientación del observador pero sin mostrar los brillos que son propios de la componente especular. Se ven ejemplos en la figura 13.

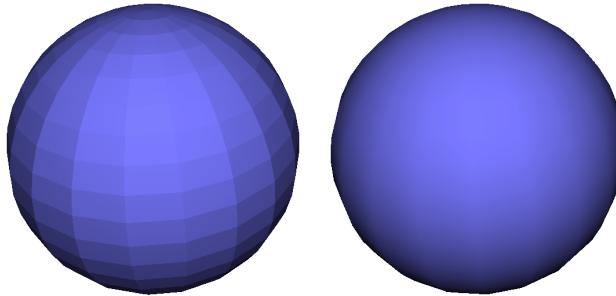


Figura 13: El material basado en el modelo de Lambert se caracteriza por ser mate.

No obstante, este material permite añadir opcionalmente una **componente emisiva** que simula la emisión de luz.

Los materiales con componente emisiva se usan para ponérselo a farolas, bombillas etc. Para transmitirle al observador la percepción de que dicha luminaria está encendida.

No confundirse, un objeto con un material con componente emisiva **no emite luz**, no ilumina nada, solo da la sensación de que sí lo hace.

En la figura 14 se muestra una farola en un escenario con unas columnas. La luminaria de la farola consta de 2 elementos, por un lado la geometría de la luminaria, una esfera, y por

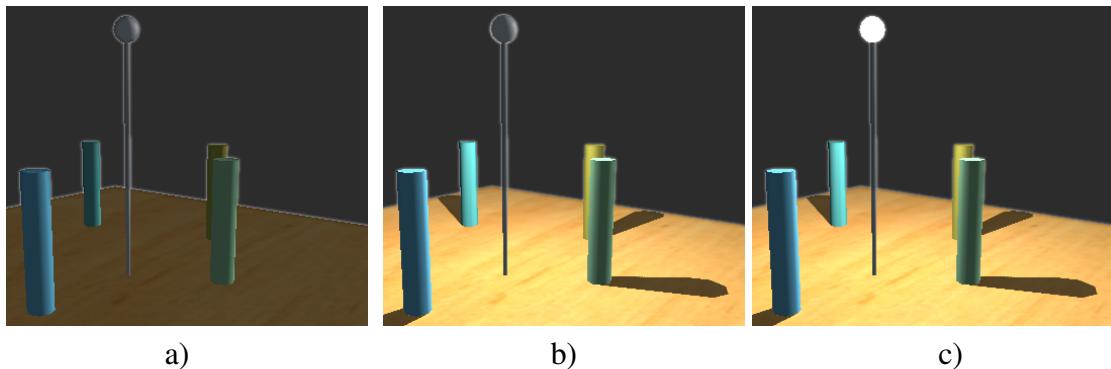


Figura 14: a) Luz apagada y luminaria sin componente emisiva. b) La luz se ha encendido pero la luminaria sigue sin componente emisiva. c) La luz continúa encendida y se activa la componente emisiva de la luminaria.

otro lado un elemento luz, en la misma posición que la esfera.

En la imagen a) la luz está apagada, el escenario de columnas está oscuro, no está iluminado. Si se quiere iluminar ese escenario, la única opción posible es encender la luz, pasando a la imagen b), se ilumina el escenario y las columnas hacen sombra, pero la escena no se percibe natural, ¿cómo es posible que la luminaria esté iluminando al mismo tiempo que ella misma se ve tan oscura como en la imagen a)? La solución está en la imagen c), donde además de encender la luz, se ha activado la componente emisiva de la esfera, dando ahora sí la sensación de que la luminaria está encendida.

Dicho esto, ¿qué atributos añade este material a la clase base?

color: Define el color difuso de la figura

emissive: Define el color emisivo de la figura. Por defecto es negro pero cuando se quiere usar se suele poner blanco, o al menos un color muy claro.

emissiveIntensity: Define la intensidad de la componente emisiva. Un valor real entre 0 y 1. En el ejemplo visto en la figura 14 se usa el mismo valor para los atributos `intensity` de la luz y `emissiveIntensity` del material.

2.4. La clase *MeshPhongMaterial*

Esta clase implementa el modelo de Phong, que contempla las componentes ambiental, difusa, especular. Aunque esta clase también contempla la componente opcional emisiva.

A efectos prácticos, es como la clase `MeshLambertMaterial` más la componente especular. Esta clase se usa para representar **materiales** que sí produzcan **brillos**.

Los atributos que añade esta clase con respecto a la clase `MeshLambertMaterial` están relacionados con esta componente, y son:

specular: Define el color de los brillos de la figura. Como se ha comentado previamente, el color especular suele definirse tomando como base el color difuso de la figura y aclarándolo hasta casi el blanco.

shininess: Define la intensidad de los brillos, por defecto tiene un valor de 30. Recordar la explicación que se ha dado del exponente de reflexión especular previamente, en la página 10.

3. Materiales con texturas

Como se ha visto, el color de cada punto de una figura se obtiene a partir del color que haya definido en el material modulándolo según el ángulo con el que incide la luz en ese punto y el ángulo desde donde mira el observador. Repasar la sección 1.3. En definitiva, si se indica que una figura es azul, su color difuso es azul, los puntos de esa figura se verán con diferentes tonalidades de azul según la luz pero toda la figura será azul.

Pero ese color difuso, no tiene por qué ser un único valor fijo almacenado en el material y que se aplique en toda la figura, sino que puede tomarse de una imagen y por tanto poder definir el color difuso de un material con todos los colores disponibles en una imagen concreta y así tener en una figura múltiples colores. Ver figura 15.

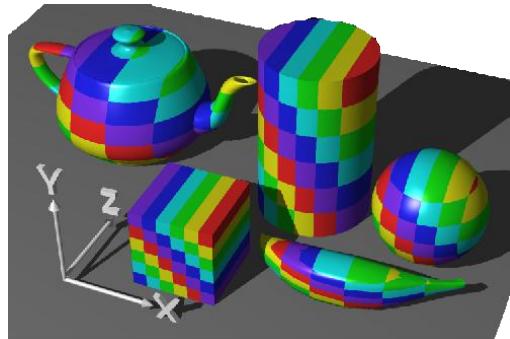


Figura 15: Mediante el uso de imágenes en la definición de un material se obtienen materiales multicolor.

Visualización: Materiales

A las imágenes que se usan para este fin se le denominan **texturas**, para diferenciarlas de la imagen que se genera al visualizar la escena, a la que se le denomina imagen, render o frame. Igualmente, a los píxeles de una textura se le denominan **téxeles**.

Pero, ¿cómo se realiza esa correspondencia entre los puntos 3D de un sólido y los téxeles 2D de una textura?

Se sabe que la piel de una figura tridimensional es en realidad un elemento bidimensional, por lo tanto, se puede hacer corresponder fácilmente con otro elemento bidimensional. Ver figura 16.

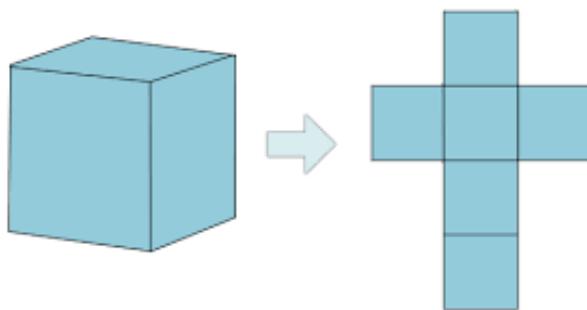


Figura 16: La piel de una figura tridimensional se puede *desdoblar* en una superficie, ya que es un elemento bidimensional.

Por lo tanto, es fácil establecer una relación de correspondencia entre los puntos de la piel de una figura tridimensional y los puntos de una textura para, por ejemplo, hacer que los puntos de la piel de la figura tomen el color de los puntos de la textura. Ver figura 17.

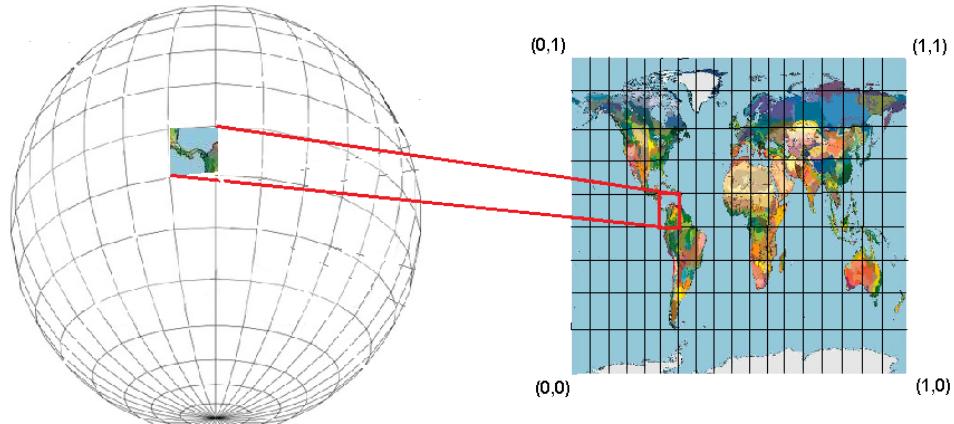


Figura 17: La piel de una figura tridimensional es directamente mapeable en una textura. Para ello se usan las coordenadas de textura.

Para ello, cada vértice de la malla de polígonos que define la piel de la figura, además de tener su coordenada (x, y, z) que establece su posición en el espacio, tiene una coordenada

de textura (u, v) , bidimensional, que establece su mapeo en la textura. Las coordenadas u y v son valores normalizados en el intervalo $[0, 1]$ de manera que la esquina inferior izquierda de la textura es la coordenada $(0, 0)$ y la esquina superior derecha de la textura es la coordenada $(1, 1)$, como se muestra en la figura 17.

Así, los algoritmos que calculan el color en cada punto de la figura, además de obtener el vector normal de ese punto y los diversos ángulos que necesite, usa las coordenadas de textura de ese punto para acceder a la textura difusa de ese material y obtener el color base que debe usar en los cálculos.

En Three, las geometrías que se construyen con clases de su biblioteca son construidas con las coordenadas de textura ya calculadas para cada vértice. En las nuevas clases que desarrolle el programador construyendo primitivas nuevas, ya sea de manera manual o procedimental, no solo hay que calcular las coordenadas de los vértices y sus conexiones para formar polígonos, también hay que calcular las coordenadas de textura si se quiere poder aplicar materiales con textura a estas primitivas.

Aunque el uso de las texturas más habitual sea para dar color a las figuras, como si se envolvieran en un papel de regalo, no es ni mucho menos el único. Porque al igual que el color que hay en un téxel de la textura se interpreta como un color para colorear un punto en la piel de la figura, el color de ese téxel se puede interpretar de otra forma para definir o modificar otros parámetros del material y conseguir otros efectos en la visualización que se haga de las figuras que tengan ese material.

Por ejemplo, en la figura 18, se ve cómo los tonos de gris de la textura a) se han usado para modificar los vectores normales de una esfera y conseguir un aspecto rugoso, imagen b), o cómo los tonos blancos y negros puros de la textura c) se usan para definir el valor de opacidad y conseguir una esfera calada, imagen d).

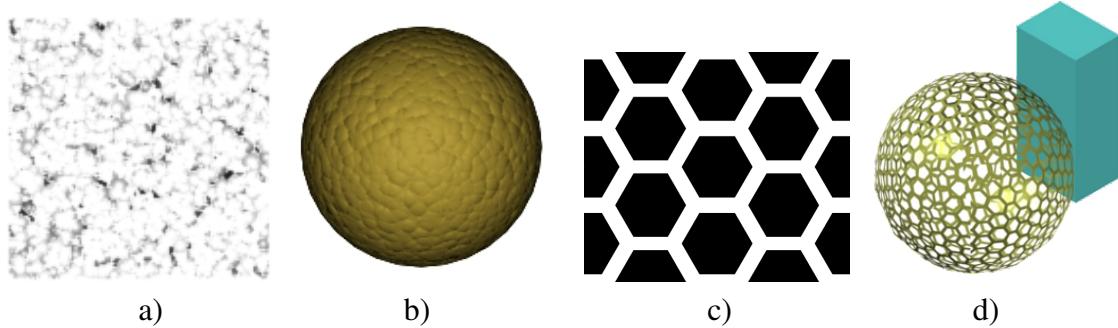


Figura 18: Los colores de los téxeles de la textura se pueden interpretar de diferente forma para modificar elementos de la figura y parámetros del material.

Se ha visto que un material está definido por varios componentes o canales (ambiental, difuso, etc.). Pues la forma de definir esos canales puede ser mediante un valor dado en el canal

y que se usaría de manera constante para todos los puntos de la figura o mediante una textura de manera que cada punto de la figura pueda tener un valor distinto en ese canal, tomado de la textura.

Un material más o menos complejo está definido en base unos valores fijos en unos canales y unas texturas en otros canales.

3.1. Uso de las texturas en Three

Usar texturas para definir un material en Three requiere diferentes acciones. Para cada una de ellas hay que:

1. Cargar la textura.
2. Configurar cómo se aplica: posición, orientación, repeticiones.
3. Asignarla a un canal de un material.

3.1.1. Carga de la textura

Hace falta un cargador de texturas. Se obtiene instanciando la clase `TextureLoader`.

Se pueden usar imágenes `.png`, `.gif`, o `.jpg`. Aunque no hay un tamaño obligatorio, los mejores resultados se obtienen con imágenes cuadradas y resolución potencia de 2 (256x256, 512x512, etc.). Siempre es conveniente tratar previamente las imágenes que se vayan a usar como texturas en algún editor de imágenes como Gimp o Photoshop.

La textura cargada se referencia con una variable que se usará para configurarla.

```
var loader = new THREE.TextureLoader();
var textura = loader.load('ruta/imagen.png');
```

3.1.2. Configuración de la textura

Aunque la textura se aplica en la figura de acuerdo con las coordenadas de textura que poseen los vértices del objeto, es posible realizar modificaciones en el espacio de la textura

como giros o desplazamientos de manera que facilita el posicionamiento y orientación de la textura en la figura.

Hay que entender estas operaciones como si se partiera de la textura original, se cargara en un editor de imágenes como Gimp o Photoshop y tras dichas operaciones se obtuviera una textura modificada que es la que se aplica; es sobre la que se mapean las coordenadas de textura de los polígonos.

Para las operaciones que se hagan en el espacio de la textura se referenciará al sistema de coordenadas 2D de la textura, denominando S al eje horizontal y T al eje vertical.

Estas operaciones se realizan actuando sobre la variable que referencia a la textura que ha devuelto el cargador.

Repeticiones

Se puede replicar la textura varias veces y de manera independiente en los ejes S y T . De manera que una repetición de 2×2 generaría una textura como la mostrada en la imagen 19.

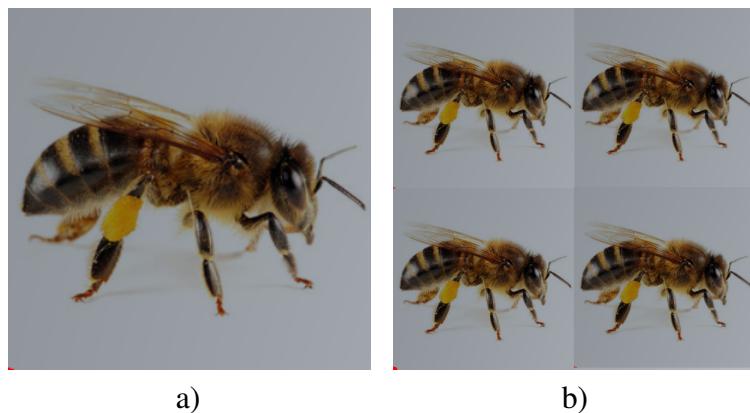


Figura 19: A partir de la textura cargada (a) se pueden configurar repeticiones para que se aplique la textura generada (b).

Las repeticiones en cada eje se indican en el atributo de la textura `repeat` que se puede definir con el método `set`. Por ejemplo: `texturaCargada.repeat.set(2, 2);`

Modificar el número de repeticiones tiene el efecto secundario de modificar el tamaño de la textura. Puede observarse en la figura 19 que poner 2×2 repeticiones de la abeja ha provocado que cada abeja haya reducido su tamaño a la mitad en ambas dimensiones. De igual modo, si se hubieran configurado 0.5×0.5 repeticiones, la abeja habría aumentado su tamaño al doble en cada dimensión y por consiguiente no se vería completa en la textura generada.

El atributo `repeat` se usa para ambas cosas, repetir el contenido de la textura y modificar su tamaño.

No obstante, esta operación tiene 3 modos de actuación, y en cada eje se puede establecer un modo distinto. Los modo de actuación en cada eje se establece asignándole la constante correspondiente a los atributos de la textura `wrapS` y `wrapT` para los ejes *S* y *T* respectivamente. Las constantes que definen los modos son las siguientes:

- `THREE.RepeatWrapping`

La textura se repite en esa dirección (*S* o *T*) tal cual. Como se muestra en la figura 19.b). Es el modo por defecto.

- `THREE.MirroredRepeatWrapping`

La textura se repite en esa dirección (*S* o *T*) reflejada. Como se muestra en el eje *S* de la figura 20.a)

- `THREE.ClampToEdgeWrapping`

La textura no se repite, pero los téxeles de la frontera de la textura sí. Como se muestra en la figura 20.b), en la que a partir de una textura cargada de tablero de ajedrez, se genera la textura mostrada replicando la última fila y columna de téxeles.

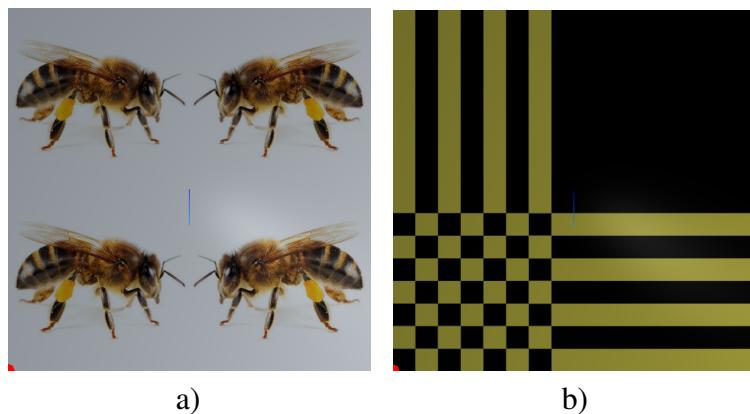


Figura 20: Al repetir la textura se puede replicar la imagen completa (a), o solo la última fila y/o columna de téxeles (b). En el caso de la imagen (a) se ha puesto un modo `MirroredRepeatWrapping` en el atributo `wrapS` mientras que en el atributo `wrapT` el modo es `RepeatWrapping`.

Si en la textura aplicada a un canal del material se cambia el modo de repetición, hay que indicar que es necesaria una actualización en dicho canal. Por ejemplo, si se ha cambiado el modo de repetición en la textura asignada al canal `map`, la necesidad de actualización se indicaría así: `material.map.needsUpdate = true;`

Aunque si lo único cambia es el número de repeticiones no hay que solicitar ninguna actualización.

Desplazamiento

El desplazamiento es bastante útil para recolocar la textura en horizontal y en vertical. Si tras aplicar una textura a un objeto se observa que “los dibujos” de la textura caen más bajos de lo que se deseaba, basta con aplicarle un desplazamiento hacia arriba a la textura.

Otro uso habitual es combinándolo con una textura a la que se le ha modificado el tamaño actuando sobre el atributo `repeat` estableciendo el modo `ClampToEdgeWrapping`. En el ejemplo de la figura 21, a la textura cargada de la abeja se le ha puesto una repetición de 2×2 para hacerla más pequeña y se le ha puesto el modo `ClampToEdgeWrapping` para tener solo un ejemplar de abeja en vez de 4. El resultado es que la abeja, más pequeña, se queda en la esquina inferior izquierda (a). Con el desplazamiento se recoloca la abeja en el centro (b).

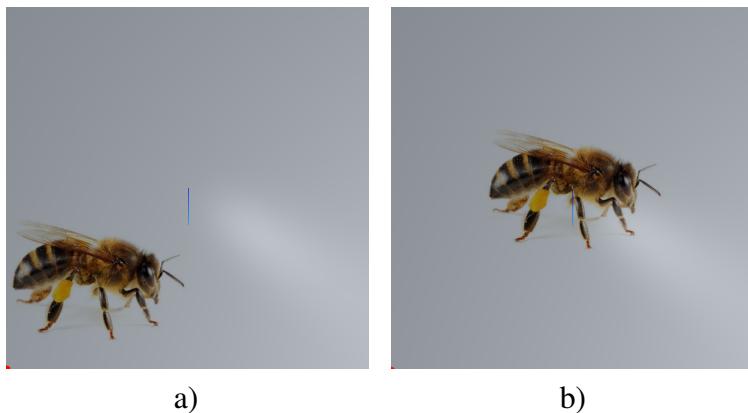


Figura 21: La operación de desplazamiento permite “recolocar” la textura para que encaje en el objeto como se deseé.

El atributo de la textura que hay que definir es el atributo `offset` que se modifica con el método `set`, con los dos parámetros que indican desplazamiento en los ejes *S* y *T*.

Tener en cuenta que el desplazamiento se produce en la dirección opuesta al parámetro.

La textura de la figura 21.b ha salido con: `textura.offset.set (-0.5, -0.5)`

Giros

Con esta operación la textura generada está girada con respecto a la cargada. Un giro que se hace siempre por un eje perpendicular al plano que contiene la textura y que pasa por un centro de giro.

El atributo para aplicar un giro es el atributo **rotation** de la textura, dándole un ángulo en radianes.

El centro de giro, que por defecto está en la esquina inferior izquierda, también se puede cambiar modificando el atributo **center** con el método `set`.

Para obtener una textura generada como se muestra en la figura 22 se ha modificado la textura así:

```
// El cargador de texturas
var loader = new THREE.TextureLoader();

// Se carga la textura de la abeja
var abeja = loader.load('../imgs/abeja.png');

// Se reduce su tamaño y se evitan las réplicas. Solo un ejemplar.
abeja.repeat.set(2,2);
abeja.wrapS = THREE.ClampToEdgeWrapping;
abeja.wrapT = THREE.ClampToEdgeWrapping;

// Se modifica el centro de giro y se realiza el giro
abeja.center.set(0.5, 0.5);
abeja.rotation = Math.PI/4;
```



Figura 22: Los giros es otra operación útil al manejar las texturas. Permiten orientar la textura de la manera deseada.

3.1.3. Canales del material donde usar texturas

Se considera que se usan materiales `MeshLambertMaterial` o `MeshPhongMaterial`.

Canal difuso

Es el canal donde se indica el color base de la figura.

Como se ha comentado, es el canal más usado para definir un material mediante una textura ya que permite tener en un material diferentes colores y con pocas figuras sencillas se pueden simular escenarios más complejos.

Por ejemplo, si solo se tuvieran materiales cuyo canal difuso solo se pudiera definir con un color, para modelar un bloque de ladrillos unidos por mortero como el de la figura 23, habría que modelar, posicionar y orientar todos los ladrillos y todas las uniones de mortero. Gracias a disponer de materiales que admiten definir el canal difuso mediante una textura se obtiene el mismo resultado modelando solo una caja.

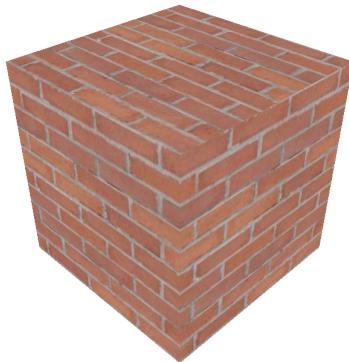


Figura 23: La técnica de las texturas ahorra trabajo de modelado.

No obstante, se tienen dos canales difusos.

color: Se le asigna un color². Es el color base del objeto.

map: Se le asigna una textura. Los téxeles son interpretados como color base en el cálculo de iluminación.

Se puede definir un solo canal difuso o definir ambos. El resultado es el que se espera. Si solo se usa un canal difuso, la información de ese canal es la que determina el color base de la figura en cada punto. Si se usan ambos, lo que se obtiene es el resultado de mezclar los colores que proceden de ambas fuentes. Como se muestra en la figura 24

Canales para simular relieve

La simulación de relieve donde no se ha modelado se consigue actuando sobre los vectores normales del objeto. Estos canales no están disponibles en el material MeshLambertMaterial.

²Aunque en los ejemplos se ha definido siempre los colores mediante un valor hexadecimal (la opción recomendada), hay muchas formas en Three de definir un color. Consultar la documentación de la biblioteca para conocerlas.

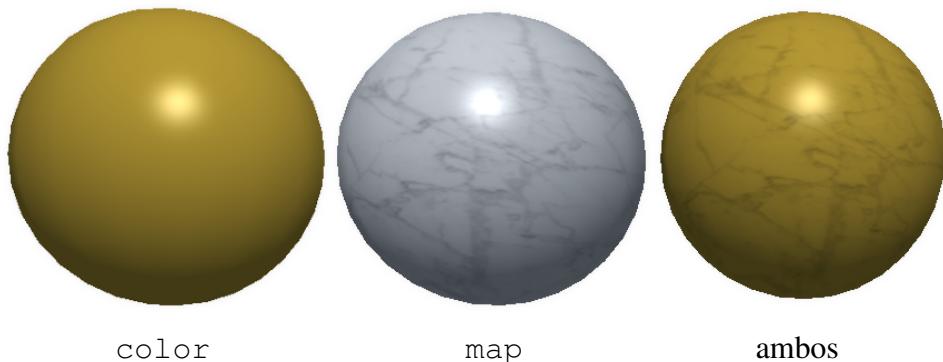


Figura 24: Hay dos canales para colorear una figura, mediante color base constante y mediante textura. Ambos canales se pueden usar conjuntamente y obtener un color combinado.

Una forma muy intuitiva de entenderlo es pensar que los vectores normales son como unos “palitos” que salen de la superficie de un objeto y que siempre, el trozo de superficie de donde sale el vector va a estar perpendicular a dicho vector. De modo que si tomamos un vector y lo inclinamos, para que el trozo de superficie que hay en su base se mantenga perpendicular, solo es posible si la superficie cede y se “arruga”.

Imaginar un experimento en donde en un papel plano se pegan unos palitos perpendiculares a la misma, figura 25. Si inclinamos diferentes filas de palitos según se muestra en la figura 26.a el resultado es que al papel le salen unos surcos como se ve en la figura 26.b.

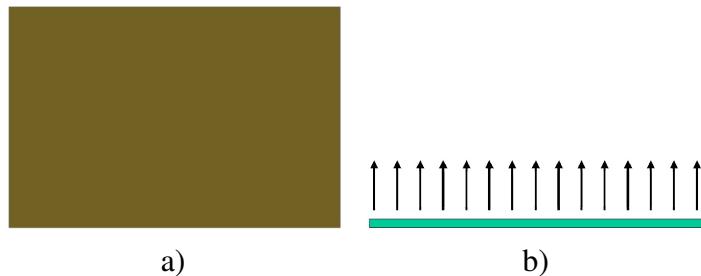


Figura 25: a) Una vista superior de un papel plano. b) Una vista frontal del papel, en el que se han dibujado los “palitos” perpendiculares pegados al papel. Vectores normales.

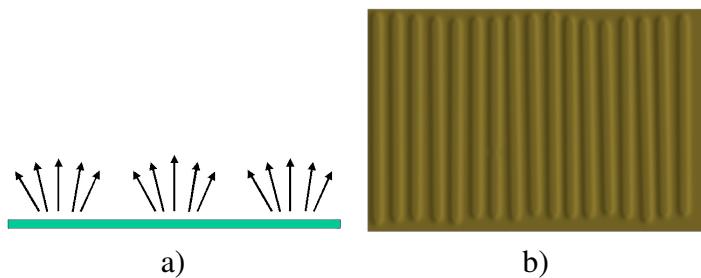


Figura 26: Inclinando los palitos, el papel cede y se arruga.

Desde un punto de vista más formal, la explicación radica en que en el cálculo de la iluminación se tienen en cuenta los ángulos que forma el vector normal con la fuente de luz y el observador. Si la normal se inclina, los ángulos cambian y el color que el modelo de iluminación calcula en ese punto es el que le corresponde a esos ángulos, es decir, es el que le correspondería a una superficie perpendicular a dicha normal inclinada.

La percepción del objeto es como si se hubiera modelado para que realmente su superficie en cada punto se corresponda con las normales utilizadas en el modelo de iluminación.

Existen dos modos habituales para actuar sobre las normales y conseguir simular relieve. Three dispone de un canal para cada modo. Se puede usar uno u otro, pero no los dos simultáneamente.

Modificando las normales existentes

Se usa una textura en tonos de gris y la normal en un punto del objeto se modifica teniendo en cuenta el gradiente de tono de gris de la textura en el téxel donde se mapea el punto del objeto considerado. A estas texturas se les llama **texturas bump map** o de perturbación de las normales.

Como sabéis, el gradiente es como la pendiente en una gráfica. Indica cómo varían los valores en la gráfica en torno al punto estudiado. En el contexto que nos ocupa, si en la textura se tiene una zona clara rodeada de zonas más oscuras, las normales se modificarán proporcionando la percepción de que la zona clara está más resaltada que las zonas oscuras colindantes.

Y al revés, si en la textura se tiene una zona oscura rodeada de zonas más claras, las normales se modificarán proporcionando la percepción de que la zona oscura está más hundida que las zonas claras colindantes.

Hay que resaltar que lo que produce percepción de relieve es el cambio de tonalidad (existencia de gradiente distinto de cero). Las zonas que sean de un mismo tono se mostrarán planas, da igual que ese tono sea claro u oscuro, lo que produce relieve son los cambios de tonalidad.

Las imágenes de la figura 27 muestran claramente este aspecto. Donde se aprecia relieve es solamente en donde ha habido cambio de tonalidad. Nótese cómo la zona que se percibe como levantada corresponde con un cambio de tonalidad hacia tonos más claros, y que la zona que se percibe como hundida se corresponde con un cambio de tonalidad hacia tonos más oscuros.

En Three, el canal para obtener relieve modificando las normales existentes es **bumpMap**. El material dispone de otro atributo, **bumpScale**, que es un escalar para controlar cómo de intenso (valores altos) o suave (valores bajos) será el efecto.

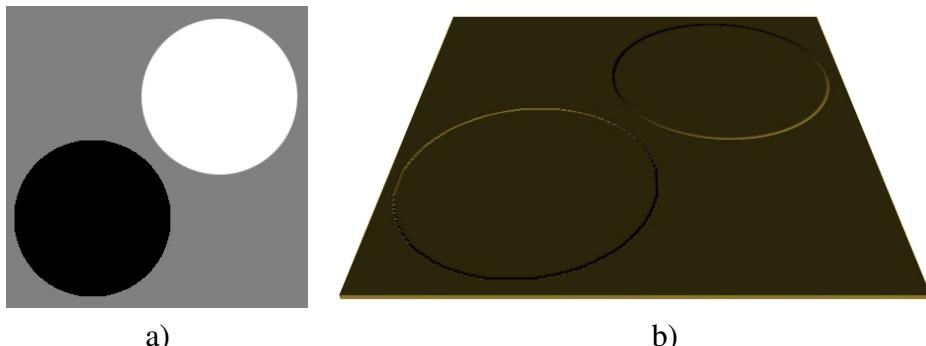


Figura 27: La textura (a) solo produce percepción de relieve (b) donde hay cambio de tonalidad.

En la figura 28 se muestra un efecto bastante intenso. Cualquier matiz en el cambio de tonalidad en la textura es muy realzado.

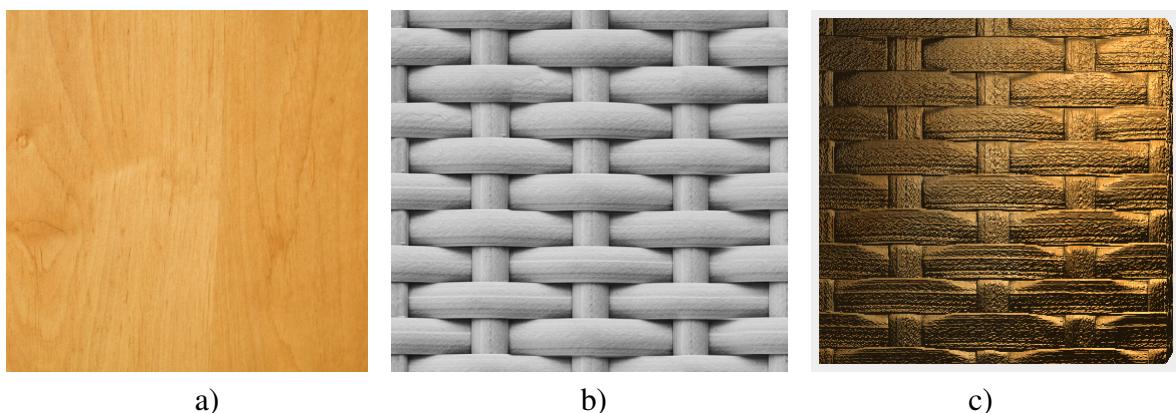


Figura 28: Sobre la tabla (a) se aplica la textura (b) en el canal bumpMap obteniendo un efecto muy intenso (c)

Sustituyendo las normales existentes

La segunda forma de actuar sobre las normales es sustituirlas por otras. En este caso, son las normales lo que está almacenado en la textura. Cuando se está aplicando el modelo de iluminación en un punto del objeto se toma para los cálculos la normal que le corresponde según la textura sin tener en cuenta para nada la normal original. A estas texturas se les llama texturas **normal map**.

En Three, el canal para obtener relieve sustituyendo las normales existentes por la de la textura es **normalMap**, y también tiene un ajuste de la intensidad en otro atributo, **normalScale**, aunque en este caso se trata de un **Vector2** cuyas componentes pueden variar entre 0 y 1. Ese factor de escala sirve para provocar una inclinación mayor o menor en la dirección *x* e *y* de manera independiente. Aunque lo usual es que las dos coordenadas de ese vector 2D de escala sean iguales entre sí de manera que se realce o se atenúe el efecto de relieve sin alterarlo hacia ninguna dirección.

En la figura 29 se muestra un ejemplo.

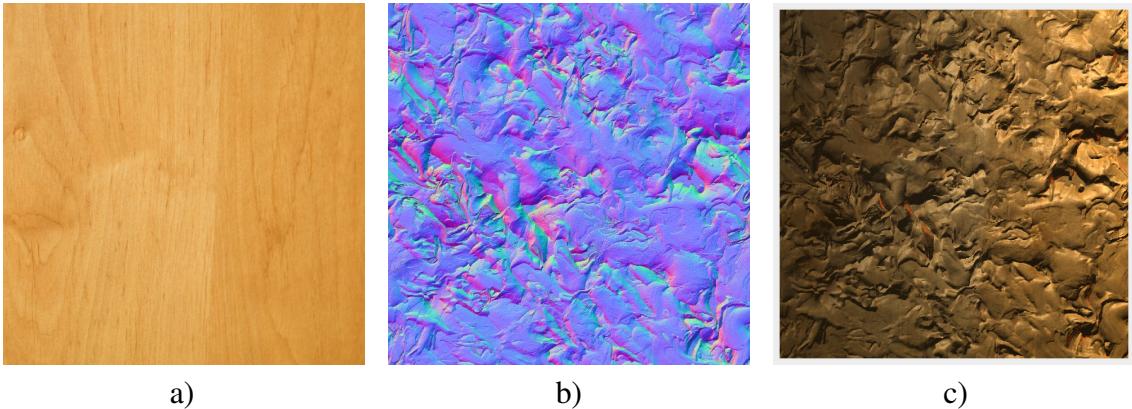


Figura 29: Sobre la tabla (a) se aplica la textura (b) en el canal normalMap percibiendo el aspecto de rugosidad que se muestra en (c)

La tonalidad azul de estas texturas cuando se abren como una imagen RGB es muy característica. Ello es debido a que un vector normal perpendicular a la superficie (como el original) es $(0, 0, 1)$, que interpretado como un color sería el azul puro. Cualquier normal que se aleje de esa perpendicularidad tendrá valores (nx, ny, nz) pero salvo que sea una normal muy *tumbada*, seguirá teniendo la tercera componente bastante más alta que las otras dos, resultando colores con bastante proporción de azul.

Canal especular

Este canal es muy similar a lo que en programas de edición de imágenes se denomina canal alfa asociado a un efecto.

Se trata de una textura en tonos de grises donde la interpretación que se hace de cada téxel es cuánto porcentaje de efecto se aplica en el punto del objeto que se mapea en dicho téxel, teniendo en cuenta que si en el téxel hay un negro puro, se aplica un 0 % de efecto; y si hay un blanco puro, se aplica el 100 % del efecto. Si hay un gris, se aplica el porcentaje de efecto correspondiente a la cantidad de blanco que hay en dicho gris.

En el caso del canal especular en el material `MeshPhongMaterial` se trata de una textura para controlar la cantidad de componente especular que interviene en el cálculo de la iluminación en cada punto del objeto. Se puede controlar en qué zonas del objeto se quieren evitar brillos total o parcialmente. El canal se denomina **specularMap**.

En el ejemplo de la figura 30 se ha usado para conseguir que solo las zonas de agua de la Tierra tengan brillos, mientras que los continentes no.

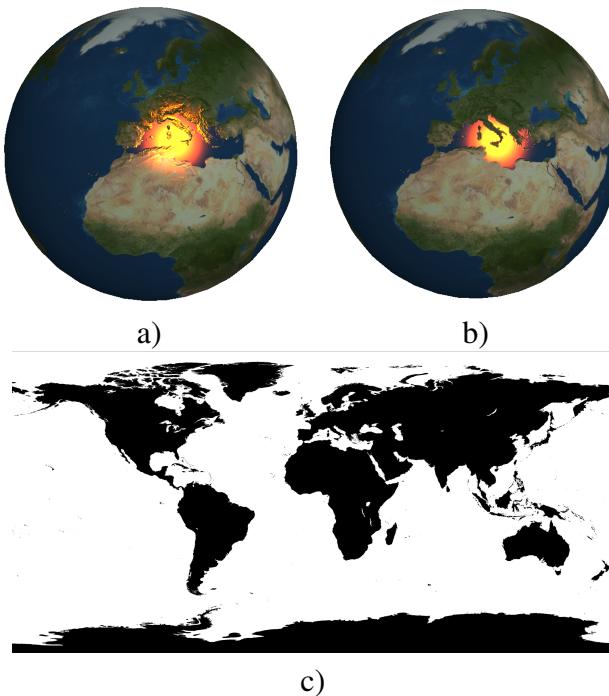


Figura 30: Al aplicar la textura (c) para controlar el canal especular, no aparecen brillos en los continentes de la Tierra (b), justo donde en la textura está el color negro puro. La imagen (a) muestra cómo se renderiza el planeta cuando no se aplica la textura indicada.

Canal de transparencia

En este caso la textura, en tonos de gris, controla la opacidad de un objeto. El téxel negro puro indica 0 % de opacidad (100 % de transparencia) y el blanco puro indica lo opuesto.

El canal se denomina **alphaMap**. Aunque para aplicarlo hay que definir un multimaterial, ya que es necesario aplicar en la misma figura 2 materiales, uno para la parte exterior y otro para la parte interior.

Aunque podría pensarse, con lógica, que debería bastar con poner el atributo `side` del material a `THREE.DoubleSide`, así no se obtiene el efecto deseado.

La forma correcta es como se muestra en el siguiente fragmento de código. Se crea un Mesh mediante un método que permite asignarle a la geometría varios materiales. En este caso se le asigna un material para el interior y otro ligeramente distinto para el exterior.

Leer los comentarios en el código para tener más detalles.

```

// Se define el material que se aplicará en el lado exterior
var matExt = new THREE.MeshPhongMaterial ();

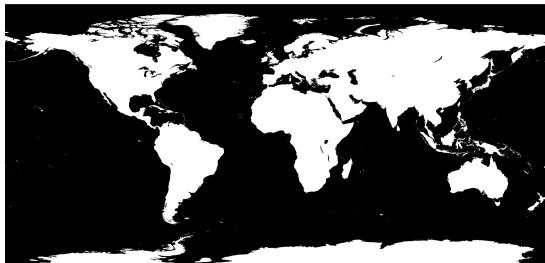
// Se configura de la manera habitual y además ...
matExt.alphaMap = unaTexturaAlfa;
matExt.transparent = true;
matExt.side = THREE.FrontSide;

// Ahora se hace el material para el interior
// Se define clonando el material exterior y cambiando el atributo side
matInt = matExt.clone();
matInt.side = THREE.BackSide;

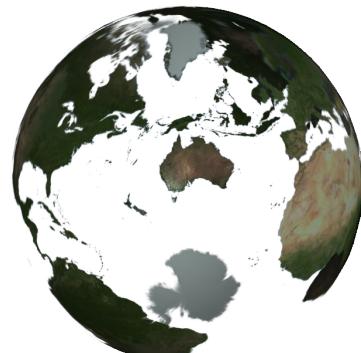
// Se crea una figura multimaterial , el interior primero en la lista de materiales
var figura = THREE.SceneUtils.createMultiMaterialObject (geometria, [matInt, matExt]);

```

El resultado se muestra en la figura 31 junto con la textura empleada.



a)



b)

Figura 31: Las zonas de color negro puro en la textura (a) producen invisibilidad en los océanos de la Tierra (b)

Sincronización de texturas

Como puede verse, es habitual en la definición de un material que se usen diversas texturas en diferentes canales: color difuso, relieve, etc.

Algunas veces, las texturas deben estar sincronizadas perfectamente para que el resultado final no presente efectos extraños. Por ejemplo, si se usa la textura de la figura 32.a en el canal difuso para dar color y la textura bump map de la figura 32.b para dar relieve y simular que el mortero está más hundido que los ladrillos, obviamente la posición en la textura de relieve de los tonos claros/oscuros de los ladrillos/mortero deben coincidir con la posición en la textura de color de los tonos rojizos/grises de los ladrillos/mortero.

Esta sincronización debe tenerse en cuenta cuando se quieran modificar las texturas en cuanto al número de repeticiones, desplazamientos y giros. Las transformaciones que se

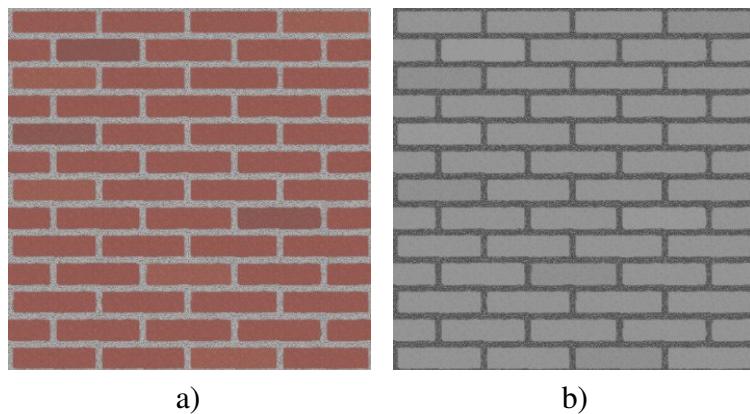


Figura 32: Dos texturas cuyos téxeles aportan diferente información complementaria en la definición de un material. En este caso esas texturas deben estar sincronizadas para que el relieve que aporta la textura (b) a los ladrillos coincida con las zonas en las que hay ladrillos en la textura (a).

hagan, deben hacerse con los mismos parámetros en todo el grupo de texturas que deban estar sincronizadas.

En Three no hay un mecanismo para hacer la sincronización automáticamente. No hay una forma de indicar que determinadas texturas forman parte de un grupo sincronizado. El programador es el responsable de aplicarle a todas las mismas transformaciones.

3.2. Texturas de entorno

Se ha hablado de los diferentes elementos que intervienen en un escenario, objetos con sus materiales, luces, posiblemente un suelo, pero ¿y el fondo?

En los ejemplos de prácticas se ha puesto como fondo de los renders realizados un color. El siguiente paso para dar un mayor realismo sería poner como fondo una textura. Pero aún quedaría poco natural porque se vería siempre la misma textura (el mismo fondo) tanto si se mira al frente, a un lado, al otro lado, arriba, etc.

Las texturas de entorno dan la solución. Se trata de una textura que envuelve a la escena de manera que al igual que en el mundo real: si se mira adelante se pueden ver unos edificios, pero si se mira hacia arriba se verá el cielo o si se mira a un lado se podrán ver unos edificios distintos a los anteriores, o un parque, o lo que sea. Pero todo continuo. En el mundo real no hay cortes ni transiciones cuando miramos lo que hay a nuestro alrededor. Y por supuesto, no se ve siempre lo mismo como fondo.

En el mundo virtual construido con un ordenador esto se consigue con las texturas de entorno. Una textura con la que se consigue tener un fondo que envuelva por completo a la escena, en todas las direcciones y que cuando la cámara se mueva de un lado a otro, el fondo se muestre siempre de manera continua, sin cortes.

Para definir un entorno de este tipo hay que usar **texturas cubemap**. Son un conjunto de 6 texturas cuadradas con las que se puede texturizar el interior de un cubo de manera que, poniendo la escena en el interior del cubo se consigue ese efecto de fondo continuo que se persigue. Se tiene un ejemplo en la figura 33.

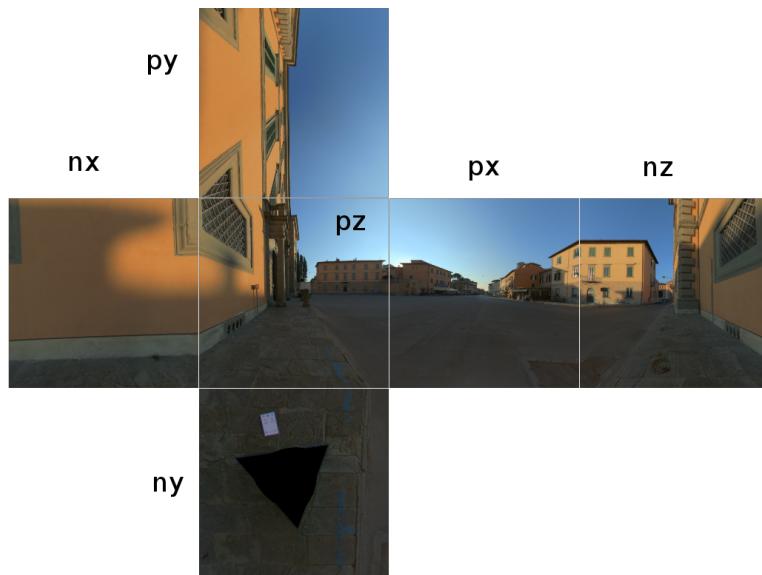


Figura 33: Una textura cubemap puede envolver una escena creando un entorno perfecto. Como se adivina, *plegando* esos 6 cuadrados adecuadamente, se construye un cubo.

La etiqueta que tiene cada imagen cuadrada alude a su posición en un cubo centrado en el origen, y por comodidad suele corresponder con el nombre del correspondiente fichero en el disco. Así, la imagen *py* alude a la cara del cubo que está en el lado positivo del eje *y* y en el disco duro estará en un archivo *py.jpg*. La imagen *nx* alude a la cara del cubo que está en el lado negativo del eje *x* y así sucesivamente.

Sin embargo, en contra de lo que pudiera pensarse en primera instancia, no es necesario crear un cubo lo suficientemente grande para que envuelva al resto de elementos de la escena. Suele haber un lugar específico en el grafo de escena para asignar ahí la textura cubemap previamente cargada y así obtener el efecto de fondo continuo requerido, sin tener que preocuparse por el tamaño de la escena ni el tamaño de ningún cubo que soporte la textura cubemap. En el caso de Three.js, la textura hay que asignarla en el atributo `background` del nodo raíz de la escena.

Para cargar estas texturas hay que usar un cargador específico, denominado `CubeTextureLoader`.

```
var path = "textures/cube/pisa/";
var format = '.png';
var urls = [
    path + 'px' + format, path + 'nx' + format,
    path + 'py' + format, path + 'ny' + format,
    path + 'pz' + format, path + 'nz' + format
];
var textureCube = new THREE.CubeTextureLoader().load( urls );
laEscena.background = textureCube;
```

Reflexión del entorno

Cuando se tiene un entorno, y en la escena hay objetos con una alta reflexión especular, como espejos, materiales cromados, etc. se espera que dichos objetos reflejen ese entorno.

Ello es posible asignando en un canal específico del material de dichos objetos la misma textura cubemap que se haya usado para el entorno de la escena. El canal se denomina **envMap**. Ver el ejemplo de la figura 34.

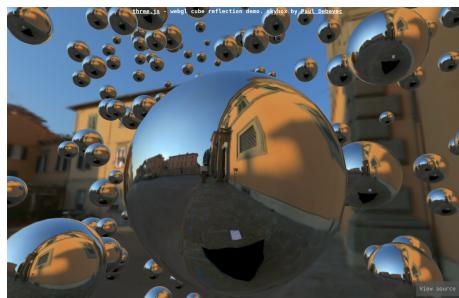


Figura 34: Las esferas cromadas reflejan el entorno. Para ello se ha asignado la misma textura cubemap del entorno en el canal `reflexEnv` del material.

Refracción del entorno

De igual modo, los objetos parcialmente transparentes dejan ver el entorno a través de ellos. Como ocurría en el caso de la reflexión del entorno del apartado anterior, hay que usar la misma textura cubemap en el canal **envMap** del material.

Pero en esta ocasión hay que modificar el modo de mapear la textura de dicho canal y, cómo suele ocurrir cuando la luz atraviesa un objeto parcialmente transparente que además es de un material de diferente densidad a la densidad del aire, la luz se refracta. Si se quiere ese efecto hay que indicar un índice de refracción. El resultado se muestra en la figura 35.

```
// Se crea el material indicandole cuál es la textura cubemap a usar en el canal de entorno
var material = new THREE.MeshBasicMaterial( { color: 0xffffffff , envMap: textureCube } );

// A dicho canal se le cambia el modo de mapeo de la textura. Modo de refracción
material.envMap.mapping = THREE.CubeRefractionMapping;

// Y se le indica al material el índice de refracción
material.refractionRatio = 0.95;
```



Figura 35: Las esferas transparentes muestran lo que tienen detrás de manera refractada.

3.3. Usar un vídeo como textura

En Three también se puede usar un vídeo como textura. Así se podría modelar un cine y proyectar una película.

Un vídeo puede considerarse como una sucesión de imágenes y cada una de estas imágenes se podría usar como textura. Solo habría que ir cambiando de textura a un ritmo de 24 imágenes por segundo.

Para conseguirlo, se debe implementar una parte en el archivo html. Se carga el vídeo pero no se muestra, aunque se le indica que se inicie solo.

```
<video id="video" autoplay loop style="display:none" src="ruta/y/fichero">
</video>
```

En el archivo javascript que corresponda se crea y configura una video-textura a partir de dicho vídeo.

```
// Se referencia al vídeo que se ha incluído en el html
var video = document.getElementById ('video');

// Se carga y configura una video-textura
var texture = new THREE.VideoTexture (video);
texture.generateMipmaps = false; // si el vídeo no es cuadrado
texture.minFilter = new THREE.LinearFilter;

// Ya solo queda asignarlo en el canal map de un material
materialPantallaCine.map = texture;
```

4. Ejemplo: El planeta Tierra

En Prado se ha dejado el código de un ejemplo en el que se emplean texturas en diferentes canales. El ejemplo ha sido tomado del capítulo 2 del libro *Three.js Essential*, de J. Dirksen; disponible como recurso electrónico en biblioteca.ugr.es.

Se trata de varios ejemplos que de manera incremental va configurando un material para mostrar en una esfera una *imagen* de la Tierra bastante conseguida. Observaréis el incremento de calidad que aporta el añadir un mapa de normales para simular relieve. Así como que se incluyan nubes que se desplazan independientemente del giro de la Tierra.

Una captura se puede ver en la imagen de la figura 36.



Figura 36: Combinando adecuadamente los recursos disponibles se consigue modelar una Tierra que incluso cuenta con nubes que se desplazan a su propio ritmo.

Hay que tener en cuenta que los ejemplos de este libro se basan en una versión antigua de la biblioteca Three; por lo que la forma en que el autor del libro hace algunas cosas, están obsoletas en la versión actual de la biblioteca.