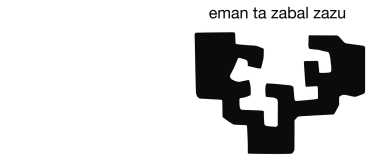


Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Computación

Proyecto de Fin de Grado

Iluminación basada en la física para gráficos por computador

Autor

Eritz Yerga Gutierrez

Directora

Carmen Hernández Gómez

Septiembre 2018

Agradecimientos

Para comenzar, quisiera agradecer a mi directora de proyecto Carmen Hernández por todo el apoyo y orientación que me ha ofrecido durante este proyecto.

También me gustaría agradecer a mi familia, mi pareja y mis amigos por todo el apoyo que me han dado durante todos estos años de la carrera tanto en los momentos buenos como en los momentos difíciles.

Finalmente, quiero agradecer a algunos profesores o compañeros de clase que considero que han influido de manera positiva en mí tanto a nivel profesional como personal.

Muchas gracias a todos.

Resumen

El objetivo principal de este proyecto es el estudio de los modelos de iluminación basados en física para los gráficos por computador e implementación de algunos de ellos en una aplicación de gráficos en tiempo real. Para ello, se ha profundizado en el estudio sobre el renderizado de gráficos por computador y se han estudiado las propiedades físicas de la luz, sus interacciones con las superficies y las distintas técnicas de modelado físico de estas interacciones a la hora de renderizar escenas a partir de las ecuaciones de *shading*.

Índice general

Índice de figuras

1. CAPÍTULO

Introducción

El renderizado por computador en tiempo real consiste en generar, de forma fluida, imágenes de una escena tridimensional con el objetivo de mostrarlas por pantalla e, incluso, permitir al usuario interactuar con la aplicación de manera inmersiva, obteniendo un *feedback* inmediato sobre sus acciones.

La velocidad de renderizado se mide en fotogramas por segundo (FPS); es decir, el número de imágenes generadas por nuestra aplicación por segundo. Dependiendo de la complejidad de la escena y de cómo se realice el renderizado, la tasa de fotogramas variará. Cuanto mayor sea la tasa de fotogramas, el movimiento se verá más fluido.

No obstante, el número de fotogramas por segundo no es la única métrica que se puede utilizar para analizar el comportamiento del renderizado. Por ejemplo, el tiempo por fotograma (*frame-time*) medido en milisegundos (ms) también es otra métrica importante a tener en cuenta. Una forma de analizar esta medida es mediante un gráfico temporal que nos muestre el *frame-time* en cada instante de tiempo. En los instantes en los que hay saltos muy notorios (por ejemplo, de 16 ms a 30 ms), aunque sólo sean durante un pequeño instante de tiempo, el ojo humano es capaz de percibir esa discontinuidad en los fotogramas la cual se manifiesta visualmente como si la imagen fuera a tirones. A este fenómeno se le denomina *stuttering* o *micro-stuttering*, dependiendo de la magnitud de los saltos en el tiempo.

Por otra parte, la percepción de la fluidez de una aplicación de gráficos por computador en tiempo real varía para cada usuario, ya que no todos los ojos humanos funcionan ni reaccionan de la misma manera. Además, el tipo de pantalla que utiliza el usuario también

afecta al resultado. Las pantallas tienen una tasa de refresco máxima medida en hercios (Hz) que nos indica el número de veces por segundo que dicha pantalla puede refrescar la imagen que muestra. Esto limita el número máximo de fotogramas que se pueden mostrar al usuario, aunque la aplicación esté generando más fotogramas que la tasa de refresco.

Los gráficos por computador comenzaron a utilizarse por primera vez en la década de los 60 y 70 para generar imágenes principalmente. Animación y gráficos 3D por computador aparecieron por primera vez en la película de ciencia-ficción *Futureworld* (1976) [11], escrita y dirigida por el novelista John Michael Crichton. Los gráficos 3D por computador en tiempo real, sin embargo, no se popularizaron hasta la década de los 90, con la llegada de los videojuegos en 3D. Una de las primeras apariciones de gran relevancia de gráficos 3D por computador en tiempo real para consolas domésticas fue en *Star Fox* (1993) [10] para la consola *Super Nintendo Entertainment System* (SNES).

Sin embargo, los grandes avances en gráficos por computador en tiempo real no comenzaron hasta la introducción de uno de los grandes pilares de hoy en día del renderizado de gráficos en tiempo real: el hardware de aceleración gráfica, las unidades de procesamiento gráfico (también conocidas como *Graphics Processing Unit*, GPU). Muchos consideran que 1996 es el comienzo de esta era con la introducción de la GPU *3Dfx Voodoo 1* [2].

En los últimos años se han dado grandes avances tanto en hardware de aceleración gráfica como en técnicas para mejorar la velocidad de renderizado y calidad de imagen, entre ellas, el modelado físico de materiales utilizados en modelos 3D y la utilización de ecuaciones basadas en propiedades físicas para calcular la iluminación y el sombreado de los polígonos en dichos modelos; esto es, el método de renderizado basado en la física (*Physically-Based Rendering* o *PBR*).

Las técnicas PBR son una evolución de las técnicas clásicas utilizadas en la iluminación en gráficos por computador hacia un modelo más basado en física. Las funciones matemáticas usadas para calcular el sombreado ahora representan las reacciones físicas correctas que la luz tiene con las superficies, entre otras cosas. Este enfoque es importante ya que nos permite representar objetos de una manera más realista y convincente en entornos virtuales, incluso en gráficos renderizados en tiempo real. Lo que nos permite acercarnos más al fotorealismo (ver Figura 1.1) o capturar el aspecto escenas de la vida real o películas. Sin embargo, esto causa que se pierda control artístico y además son modelos de iluminación más costosos computacionalmente que los modelos de iluminación más antiguos.

Estas técnicas pueden ser muy útiles en películas CGI, videojuegos y realidad virtual

Figura 1.1: Detroit: Become Human (2018).
Captura de un fotograma renderizado en tiempo real en el videjouego. Utiliza un shader BRDF con término difuso Lambert y término especular de micro-facetas GGX, materiales diseñados para PBR, unidades fotométricas para luces y materiales con emisión y colocación de luces y cámara en la escena como si se estuviera filmando en la vida real. [5]

donde la inmersión del usuario es una parte muy importante de una experiencia positiva, pero también podrían ser muy útiles en aplicaciones de diseño de interiores o exteriores para previsualizar una aproximación de lo que podría ser el resultado final (si se consigue capturar la iluminación y materiales de la vida real correctamente). No obstante, estas técnicas también tienen sus limitaciones, simulan las interacciones físicas de la luz con la materia a nivel macroscópico y por lo tanto si se necesita una precisión mayor que eso no son adecuadas. Además, en algunos casos no se puede conseguir PBR completo en tiempo real, por lo que en aplicaciones en tiempo real algunos modelos de iluminación PBR no se pueden utilizar o se deben realizar ciertos sacrificios en la exactitud de la simulación.

2. CAPÍTULO

Documento de los objetivos del proyecto

2.1. Objetivos del proyecto (alcance y exclusiones)

El alcance del proyecto se recoge en dos apartados: objetivos de aprendizaje y objetivos de diseño.

Los objetivos de aprendizaje son los siguientes:

1. Aprender a utilizar la librería JavaScript de WebGL conocida como ThreeJS.

2. Estudiar los conceptos básicos necesarios para comprender las técnicas utilizadas en la iluminación basada en la física de escenas 3D.

3. Estudiar en profundidad las técnicas utilizadas en la iluminación basada en la física de escenas 3D.

Los objetivos de diseño son los siguientes:

1. Implementación de un modelo de iluminación basada en la física sencillo.

2. Implementación de un modelo de iluminación basada en la física más complejo y completo.

2.2. Herramientas utilizadas

En este proyecto se han utilizado las siguientes herramientas:

- *Three.js*: Una librería JavaScript para crear escenas 3D y mostrarlas mediante WebGL.
- *GLSL*: El lenguaje de los *shaders* para OpenGL o WebGL.
- *Overleaf*: Un editor online para el lenguaje *LaTeX*. Se ha utilizado para redactar la memoria.
- *Visual Studio Code*: Un editor de código multiplataforma con soporte para múltiples lenguajes y extensiones. Se ha utilizado para escribir el código JavaScript y GLSL.
- *Chrome*: Un navegador web basado en *Chromium* con soporte para WebGL y que dispone de una herramienta de depuración JavaScript muy potente. Se ha utilizado para ejecutar y depurar la aplicación.
- Librerías JavaScript: Se han utilizado librerías JavaScript adicionales para el desarrollo de la aplicación. Su objetivo y funcionamiento se encuentran detallados en la sección *Librerías utilizadas* del capítulo 4 *Desarrollo del proyecto*.

A continuación detallamos cada una de las herramientas mencionadas.

2.2.1. Three.js

ThreeJS es una librería JavaScript para crear escenas 3D y visualizarlas en un navegador con soporte para *WebGL*. Provee clases de objetos JavaScript para crear cámaras, escenas, modelos 3D (creando la geometría en JavaScript o cargando modelos desde ficheros), materiales (a los que se les puede añadir shaders GLSL) y luces de manera rápida y funciones con las que manipular dichas clases, realizar cálculos o conversiones de tipos, entre otras funcionalidades. Su página oficial es: https://threejs.org/. Y la librería está disponible a través del repositorio oficial en GitHub: https://github.com/mrdoob/three.js/.

2.2.2. GLSL

GLSL es el lenguaje de shaders de OpenGL. Se trata de un lenguaje declarativo y fuertemente tipado, su sintaxis es similar a C.

Los *shaders* GLSL que hemos utilizado son: *Vertex shader* y *Fragment shader*. Los vertex-shader son programas que se ejecutan en la GPU por cada vértice de los modelos 3D, mientras que los fragment-shader son programas que se ejecutan en la GPU por cada pixel de los modelos visible en la cámara.

GLSL provee una interfaz de programación de aplicaciones (abreviado *API*) que permite el uso sencillo y directo de funciones matemáticas (como por ejemplo *dot()* para realizar el producto escalar entre dos vectores o funciones trigonométricas como *cos()*, *sin()* o *tan()*), funciones de conversiones de tipos (por ejemplo, *vec4()* para convertir vectores de dimensiones inferiores a un vector 4D) y tipos de datos frecuentemente usados en shaders (por ejemplo, mat4 para matrices 4D, vec3 para vectores 3D, ...) entre otras. Se incluye una lista de tipos de variables utilizadas en este proyecto en el anexo A *Tipos de variables GLSL*, así como una lista de funciones utilizadas en este proyecto en el anexo B *Funciones prefefinidas en GLSL*. Además, se incluye una referencia en cada uno para consultar más información.

2.2.3. Overleaf

Overleaf es un editor de documentos *LaTeX* online que tiene todas las funcionalidades de cualquier otro editor convencional dispone y además permite compartir el documento con la posibilidad de realizar ediciones simultáneamente. Su página oficial es: `https://www.overleaf.com/`.

Se escogió esta herramienta para escribir la memoria debido a su sencillez de uso y funcionalidades para compartir la memoria con posibilidad de edición.

2.2.4. Visual Studio Code

Visual Studio Code es un editor de código multiplataforma, de código abierto y con soporte para muchas herramientas y lenguajes. Gracias a las extensiones disponibles para el editor se puede expandir la su funcionalidad de manera sencilla.

Se escogió esta herramienta para crear la aplicación por familiaridad con el uso de la herramienta y por el soporte nativo para HTML y JavaScript que dispone, así como sus extensiones para el resaltado y depuración del lenguaje GLSL.

2.2.5. Chrome

Chrome es un navegador web creado por Google basado en el proyecto *Chromium*. Está disponible en varias plataformas, tiene soporte para WebGL (lo cual es un requisito para el uso de la aplicación) y dispone de un depurador con diversas funcionalidades.

Se escogió esta herramienta para las pruebas de la aplicación ya que este navegador es muy popular y proporciona el soporte necesario para el desarrollo de la aplicación.

2.3. Planificación

El proyecto se ha llevado a cabo desde Diciembre de 2017 hasta Agosto de 2018 y se ha dividido en dos fases:

- Documentación: En esta fase se ha consultado y estudiado sobre fenómenos visuales, iluminación, propiedades de la luz y la teoría BRDF. Una vez establecida la base, consulté publicaciones sobre los shader BRDF establecidos como objetivos de diseño, decidimos implementar el BRDF Cook-Torrance como modelo de ilumina-ción inicial debido a su sencillez y el BRDF Principled como objetivo final debido a ser un modelo muy completo y más complejo. Para el shader BRDF Cook-Torrance consulté *A Reflectance Model for Computer Graphics* [6] y *An Inexpensive BRDF Model for Physically-based Rendering* [13] y para el shader BRDF Principled consulté *Physically-Based Shading at Disney* [4] y *Extending the Disney BRDF to a BSDF with Integrated Subsurface Scattering* [3]. También consulté *Microfacet Models for Refraction through Rough Surfaces* [16] y *Background: Physics and Math of Shading* [9] para información adicional.

- Implementación de la aplicación y redacción de la memoria: Durante esta fase implementé la aplicación en JavaScript y preparé una escena de prueba con el fin de probar los shaders GLSL a medida que los implementara. Primero implementé el shader BRDF Cook-Torrance y una vez completada la implementación de dicho

shader, implementé el shader BRDF Principled. Para esta fase consulté la documentación de ThreeJS [1] y documentación sobre GLSL [7] [15] [14] cuando fue necesario.

La fase de documentación comenzó en Diciembre y continuó durante toda la duración del proyecto. La fase de implementación comenzó en Abril y la redacción de la memoria comenzó en Mayo.

3. CAPÍTULO

Estado del arte

Las técnicas de iluminación y sombreado en los gráficos por computador han ido evolucionando, aumentando en complejidad y realismo, a medida que la tecnología avanzaba gracias al desarrollo de hardware con mayor potencia de cómputo y de software. Las técnicas de iluminación basada en física (*Physically Based Rendering* o PBR) fueron implementadas en un principio para *pre-renderizado* o *renderizado offline* (es decir, no se utilizaban en tiempo real) debido al coste computacional y a que el hardware todavía no era lo suficientemente potente.

A principios de los años 2010 comenzaron a aparecer implementaciones en tiempo real de los métodos PBR, además también empezaron a aparecer artículos que tomarían grande importancia en la industria, como por ejemplo el de Disney [4] en 2012. En aquel tiempo los fabricantes de consolas también estaban en desarrollo de la nueva generación de consolas, que se lanzaron en 2013. La popularización de los métodos PBR para renderizado en tiempo real pasó en el mismo instante que dicho cambio generacional. Con el cambio de generación de consolas los estudios de videojuegos tuvieron que realizar actualizaciones significativas a sus motores gráficos, esto junto con la potencia de cómputo superior de la nueva generación de consolas les dio la oportunidad de integrar las técnicas PBR en su metodología de trabajo.

Ahora mismo las técnicas PBR se han convertido en el éstandar de la industria de gráficos en tiempo real. Son de gran importancia y gran parte de su popularización es gracias a los videojuegos en los que actualmente, debido a que el objetivo es conseguir un mayor realismo en los gráficos y una mayor inmersión, es una práctica común utilizar shaders

BRDF y materiales diseñados para estos mismos.

4. CAPÍTULO

Definiciones básicas

Nuestro objetivo, enmarcado en el área de gráficos por computador, es generar una imagen 2D lo más realista posible de una escena tridimensional compuesta por modelos geométricos 3D, materiales y texturas que les aportan una apariencia más fiel a la realidad, una o más fuentes de luz y una cámara virtual colocada en dicha escena. Por todo ello, comprender los fenómenos físicos relacionados con la luz es de vital importancia cuando se pretende realizar un renderizado realista de una escena.

Fundamentalmente, el efecto de iluminar una escena se describe mediante un *modelo de iluminación* que tiene en cuenta la interacción de la energía electromagnética con las superficies de los objetos de la escena. Los modelos físicos de iluminación tienen en cuenta una serie de factores, como las posiciones de los objetos en relación con las fuentes de luz y con otros objetos, las propiedades de los materiales y las características de las fuentes luminosas. Por ejemplo, los objetos de la escena pueden ser más o menos transparentes, tener materiales opacos, brillantes, mates y/o texturas. Además, pueden utilizarse fuentes de luz de formas, colores y posiciones variables para iluminar una escena.

Esto es, los modelos de iluminación tienen en cuenta los parámetros de las propiedades ópticas de las superficies, las posiciones de los objetos en la escena, el color y las posiciones de las fuentes de luz, junto a sus características propias, y la posición y la orientación de la cámara a la hora de calcular la intensidad de la luz proyectada desde una posición concreta de la superficie en una dirección de vista específica.

A continuación, describiremos detalladamente algunos conceptos básicos para entender los modelos de iluminación que hemos utilizado en este trabajo.

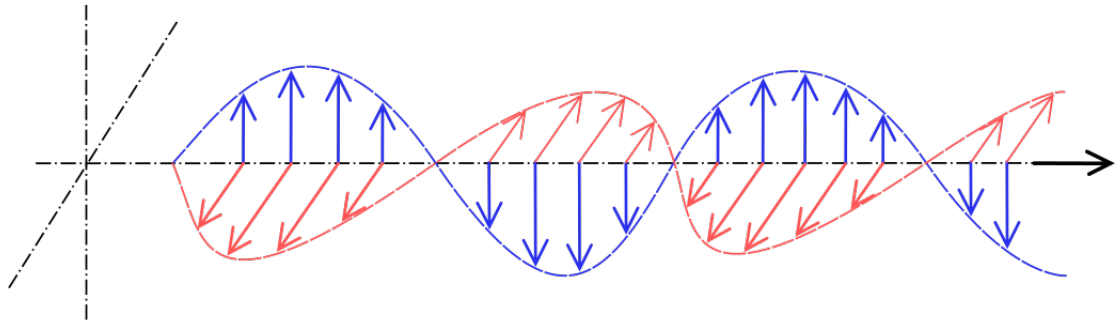


Figura 4.1: Representación de una onda electromagnética. El campo eléctrico (en azul) y el campo magnético (en rojo) son perpendiculares entre sí.
(*Imagen de Wikimedia*)

4.1. Luz

La luz es la radiación electromagnética que puede ser percibida por el ojo humano y que consiste en un flujo de fotones que pueden actuar como partículas u ondas (dualidad onda-partícula) dependiendo de cómo lo observemos y que explica las características de su comportamiento físico. La velocidad de la luz en el vacío es 299 792 458 m/s.

Las ondas electromagnéticas (ver Figura 4.1) son oscilaciones de los campos eléctricos y magnéticos emitidas por partículas cargadas en aceleración que se propagan por el espacio-tiempo y transportan energía radiante electromagnética. Se realiza una clasificación de estas ondas según la longitud de onda y a esta clasificación se le denomina el *espectro electromagnético* (ver Figura 4.2) el cual abarca todas las longitudes de onda que la luz puede tener. Denominamos *luz visible* a las ondas cuya longitud de onda se enmarcan dentro del *espectro visible* y se encuentran entre la longitud de onda mayor que la luz ultravioleta y menor que la luz infrarroja (entre 380 nm y 760 nm) que es el rango de luz que los ojos humanos pueden ver.

Una propiedad importante de los fotones relacionada con las ondas es que cada fotón tiene una frecuencia o longitud de onda asociada. La energía de cada fotón es proporcional a su frecuencia, lo que afecta las interacciones entre los fotones y la materia. Las frecuencias o longitud de onda distintas se perciben como luz de colores diferentes (o no se perciben si están fuera del espectro visible).

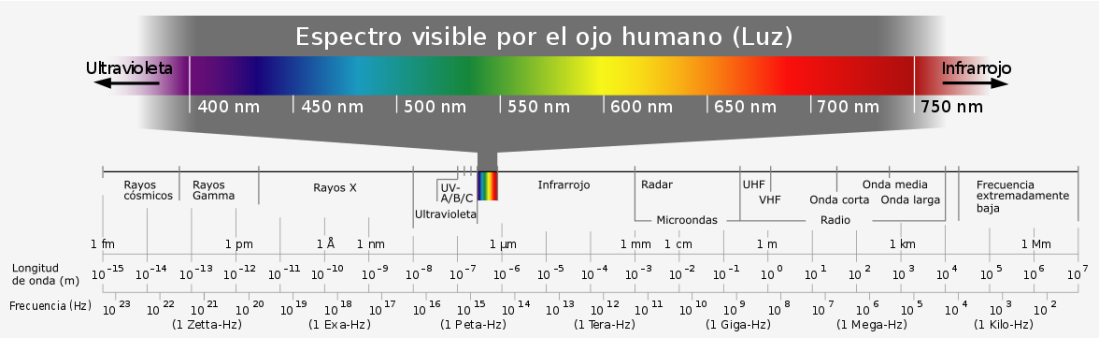


Figura 4.2: Espectro electromagnético. La luz visible se encuentra dentro del espectro visible, en el rango de longitudes de onda de 380 nm-760 nm.
(*Imagen de Wikimedia*)

4.2. Color

El color es el resultado de la percepción visual de la luz por un sensor. El ojo humano puede distinguir aproximadamente 10 millones de colores diferentes del espectro visible. Las células sensibles de la retina reaccionan de modo distinto a la luz según su longitud de onda. Existen dos tipos diferentes de receptores en la retina: bastones y conos. Los bastones son más numerosos y más sensibles que los conos. Sin embargo, los bastones solo perciben las tonalidades de grises. Por el contrario, los conos miden las radiaciones electromagnéticas (son células fotosensibles) y proveen de la sensibilidad al color del ojo. La evidencia experimental sugiere que hay tres tipos diferentes de conos según la recepción del color y que cada uno de esos receptores responden de manera distinta a diferentes longitudes de onda. El cerebro recibe tres señales diferentes de estos receptores y, por ello, cualquier color visible puede ser representado mediante tres números de manera precisa (también conocidos como valores triestímulos). La percepción del color se ha modelado en términos de estos valores triestímulos dando lugar a diferentes representaciones denominadas *espacios de color*.

Existen diversos espacios de color para distintos propósitos: CIE XYZ, RGB, CYMK, CIELAB, HSV, HSL, ... Cada espacio de color tiene sus ventajas e inconvenientes, y es por ello que se utilizan distintos sistemas según los propósitos y circunstancias.

Para representar los colores que perciben los receptores cono con la base de colores primarios más cercana a los mismos, la *Comisión Internacional de la Iluminación* (*Commission internationale de l'éclairage* o CIE) propuso el espacio de color CIE XYZ (ver Figura 4.3).

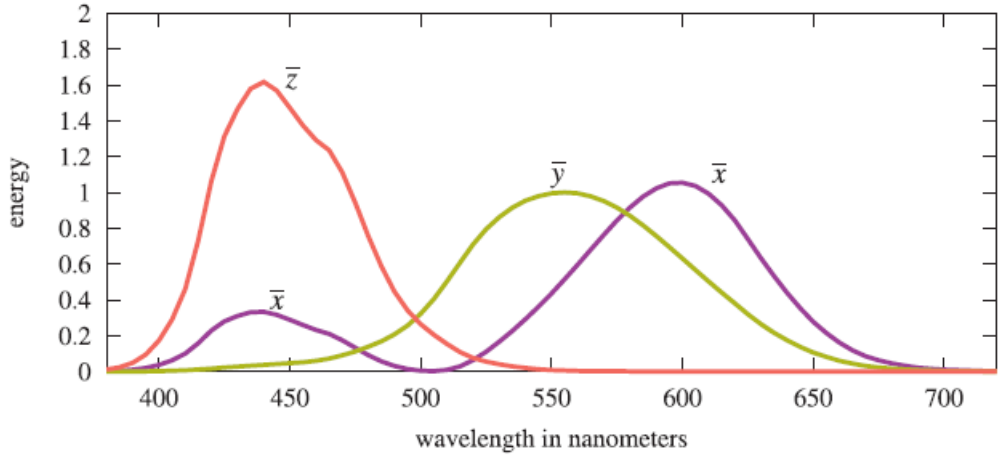


Figura 4.3: Los colores primarios del espacio de color CIE XYZ son $\bar{x}$, $\bar{y}$ y $\bar{z}$.
(*Imagen de Real-Time Rendering* [2])

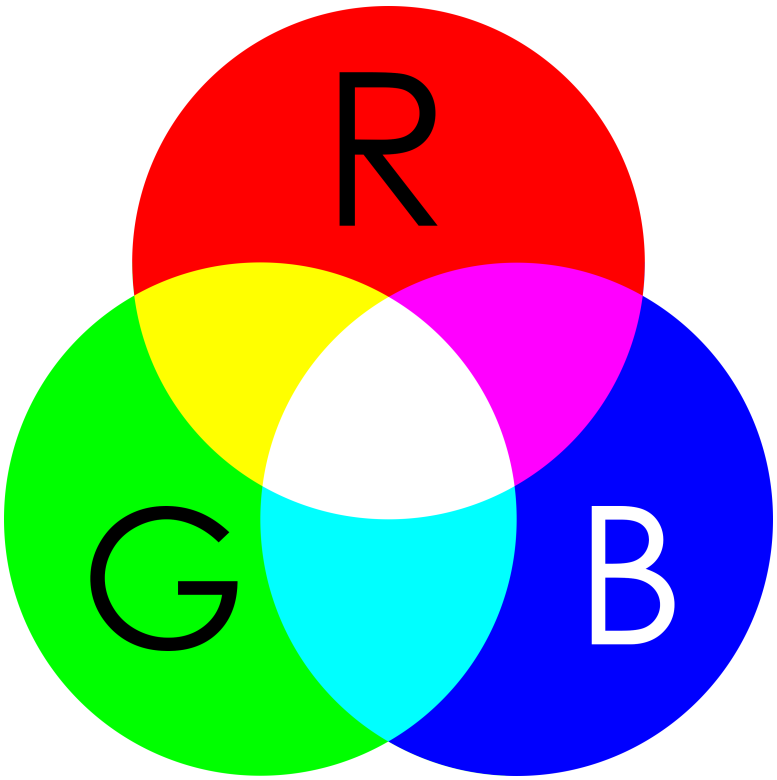


Figura 4.4: Espacio de color RGB. Los colores primarios son: rojo, verde y azul.

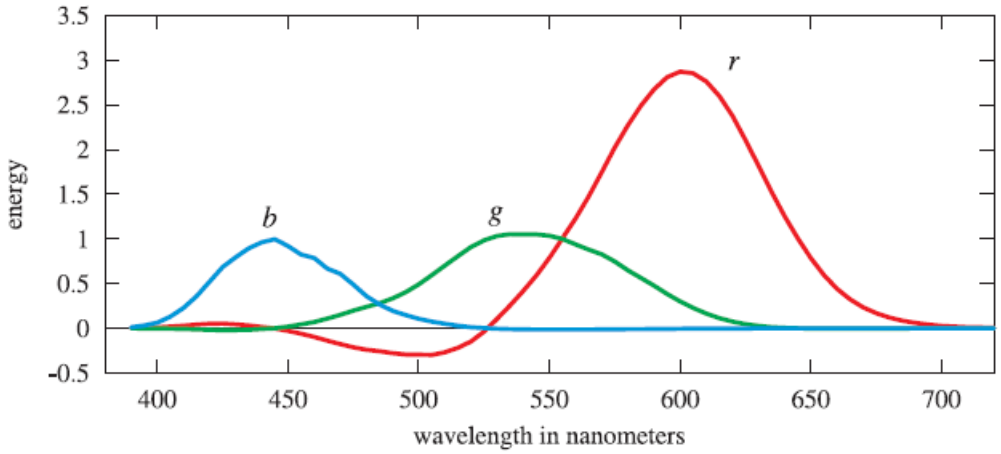


Figura 4.5: Colores primarios del espacio de color RGB.
(*Imagen de Real-Time Rendering* [2])

Sin embargo, los monitores y pantallas de hoy en día trabajan en el espacio de color RGB o sRGB, ya que utilizan los colores base de este espacio para obtener el color de un píxel. Este espacio de color utiliza tres colores primarios - *r* (rojo), *g* (verde) y *b* (azul) (ver Figura 4.4 y Figura 4.5) - y pueden mostrarse con 3 luces monocromáticas. No obstante, este espacio no puede representar todos los colores del espacio CIE XYZ ya que, como puede observarse en la Figura 4.5, para representar colores de ciertas longitudes de onda, las luces base deberían tener energía negativa y eso no es posible.

4.3. Modelos geométricos

Los modelos geométricos tridimensionales o modelos 3D vienen dados por *vértices* (puntos en el espacio 3D), *aristas* (segmentos que unen los vértices) y *caras* (polígonos formados por dichos vértices y aristas). Un ejemplo de un modelo 3D puede observarse en la Figura 4.6.

4.4. Fuentes de luz

Una *fuente de luz* o una fuente luminosa es un objeto que emite luz visible que se percibe por los órganos de la visión. Como hemos dicho anteriormente, la luz puede modelarse como *rayos geométricos*, *ondas electromagnéticas* o *fotones*. Independientemente de

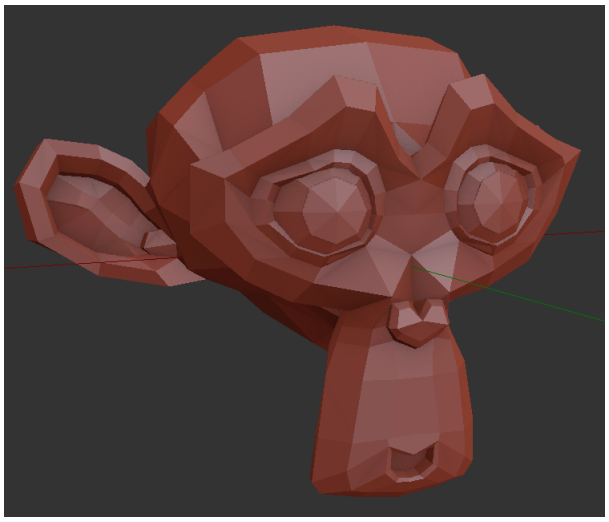


Figura 4.6: *Suzanne*, modelo de ejemplo predefinido en el programa de modelado *Blender*. Compuesto por 507 vértices y 500 caras.

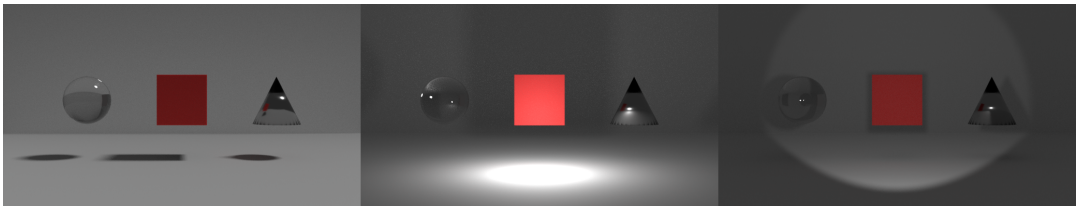


Figura 4.7: Comparativa entre distintas fuentes de luz.
En la izquierda vemos una luz direccional, en el centro una luz puntual colocada enfrente de los modelos y en la derecha una luz spot colocada enfrente de los modelos apuntando hacia los modelos.

cómo decida modelarse, la luz es la energía electromagnética radiante que viaja a través del espacio y que se propaga directamente hacia los objetos de la escena. Por otra parte, las *superficies de los objetos* absorben cierta parte de la luz y dispersan otra parte en diferentes direcciones. La luz que no sea absorbida continuará viajando por la escena y encontrándose con otros objetos. Por tanto, las fuentes de luz *emiten* luz, no la dispersan ni absorben. Finalmente, una parte de la luz acaba topándose con el *sensor* (por ejemplo, el ojo humano o una cámara), y dicho sensor absorbe la luz que le llega y la usa para capturar la imagen.

En el renderizado, las fuentes de luz pueden representarse de diferentes maneras dependiendo del tipo de iluminación que se pretenda simular. Para cada luz debe especificarse su *energía radiante*; esto es, la cantidad de energía que emite la fuente de luz. La irradiancia es equivalente a la suma de las energías de los fotones que pasan por la superficie en

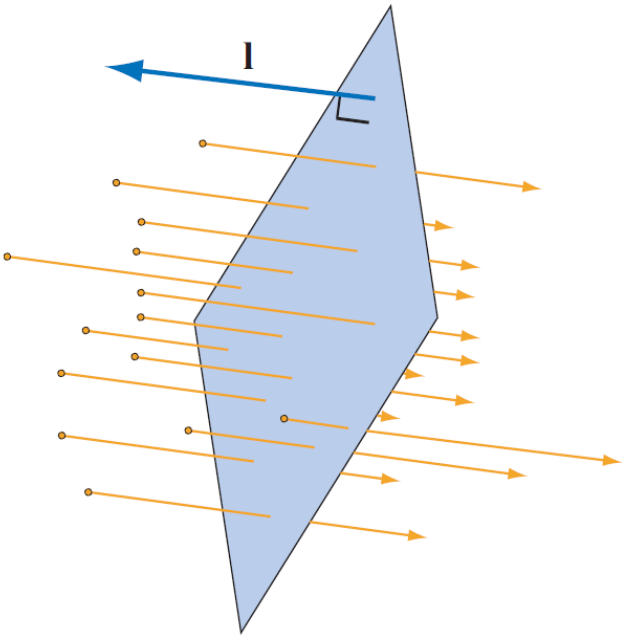


Figura 4.8: Dirección de la luz.
(*Imagen de Real-Time Rendering* [2])

un segundo, y se representa mediante un color en formato RGB (rojo, verde y azul).

La luz rebota múltiples veces sobre los objetos y paredes de una escena antes de iluminar un objeto en particular. A este nivel general de iluminación que no proviene directamente de una fuente de luz se le denomina *luz ambiental*. Es decir, este tipo de luz es solamente una aproximación de los múltiples reflejos de la luz en una escena y modela cómo la luz se puede dispersar o reflejar muchas veces produciendo un efecto uniforme.

A continuación, describimos las fuentes de luz más comunes usadas en el renderizado.

4.4.1. Luces direccionales

Las luces direccionales que simulan las fuentes de luz extremadamente lejanas (como por ejemplo, el sol) son las más simples de modelar: Su luz viaja en una única dirección y permanece en la misma dirección en toda la escena.

La dirección de la luz se describe mediante el vector normalizado l (o *vector de luz*) que apunta hacia la *dirección contraria* a la dirección en la que viaja la luz de la fuente y se especifica en el espacio de coordenadas del mundo (véase Figura 4.8).

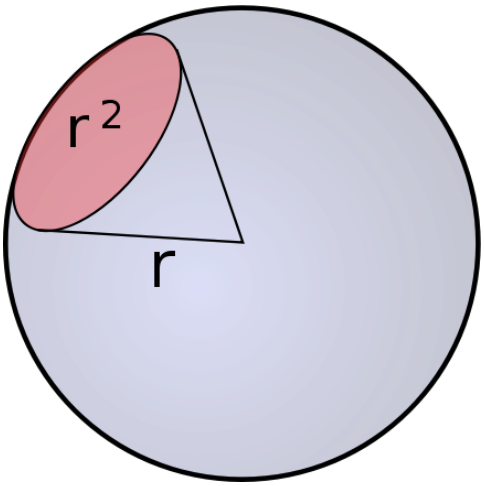


Figura 4.9: Un esteorradián.
(*Imagen de Wikimedia*)

La *intensidad* o *intensidad radiante* I es el flujo radiante[1] transportado en una dirección dada en un ángulo sólido unitario[2]; es decir, un ángulo sólido cónico de 1 estereorradián (ver Figura 4.9). Su unidad de medida es el vatio por estereorradián (W/sr).

4.4.2. Luces puntuales

Las fuentes de luz puntuales (o luces posicionales) son luces definidas por su posición p_L y su intensidad I_L. Las luces puntuales con valor constante de intensidad reciben el nombre de *luces omni*.

Las ecuaciones de sombreado para este tipo de luces usan el vector de incidencia de la luz l y la irradiancia E_L. Ambos pueden ser calculados usando la intensidad y la posición de la fuente de luz, I_L y p_L, y la posición p del vértice o píxel a iluminar:

$$r = ||p_L - p||$$
$$l = \frac{p_L - p}{r}$$
$$E_L = I_L f_{dist}(r)$$

[1] El flujo radiante es la potencia de energía o radiación electromagnética transportada por las ondas.
[2] Un ángulo sólido es un conjunto continuo de direcciones, medido en *esteorradianes* (abreviado como sr). El esteorradián se define como el área cubierta por el ángulo sólido en la superficie de una esfera de radio unitario.

donde $f_{dist}(r)$ es la función que describe cómo se reduce la irradiancia E_L a medida que la distancia entre la fuente de luz y la superficie aumenta. Estas funciones se utilizan para representar el comportamiento de las fuentes de luz en la vida real.

4.4.3. Luces spot

Las luces spot o *spotlights* son luces puntuales que presentan una variación direccional de su intensidad. En su forma más básica se definen mediante una posición p_L, una dirección d, un ángulo de corte θ_u llamado *ángulo de umbra*, su intensidad máxima $I_{L_{max}}$ y un exponente s_{exp}.

El vector de dirección de la luz y su irradiancia, l y E_L , se calculan de la misma forma que para las luces puntuales. Sin embargo, la intensidad de la luz spot, I_L, es ahora una función dependiente del vector de dirección de la luz l que se calcula de la siguiente forma:

$$I_L(l) = \begin{cases} I_{L_{max}}(\cos\theta_s)^{s_{exp}} & \theta_s \leq \theta_u \\ 0 & \theta_s > \theta_u \end{cases}$$

siendo θ_s el ángulo formado entre el vector de dirección d y el vector $-l$.

4.5. Irradiancia

Como hemos descrito, la energía radiante es la cantidad de energía que emite una fuente de luz. Esta cantidad puede ser cuantificada midiendo la energía que pasa en un segundo por una superficie de área unitaria perpendicular al vector de dirección de la luz, l. La cantidad de energía emitida por unidad de tiempo y por área de superficie es la irradiancia.

Por tanto, para calcular la iluminación que recibe una superficie, hay que medir la irradiancia en un *plano paralelo* a la misma. La irradiancia recibida por una superficie es equivalente a la irradiancia medida en la fuente de luz (irradiancia medida en un plano perpendicular a la dirección de la luz, l) multiplicada por el coseno de ángulo[3] formado entre la dirección de la luz l y la normal de la superficie n (ángulo al que nos referiremos como θ_i).

$$E = E_L \cos\theta_i$$

[3]Este coseno puede calcularse mediante un producto escalar cuando se realicen los cálculos computacionales.

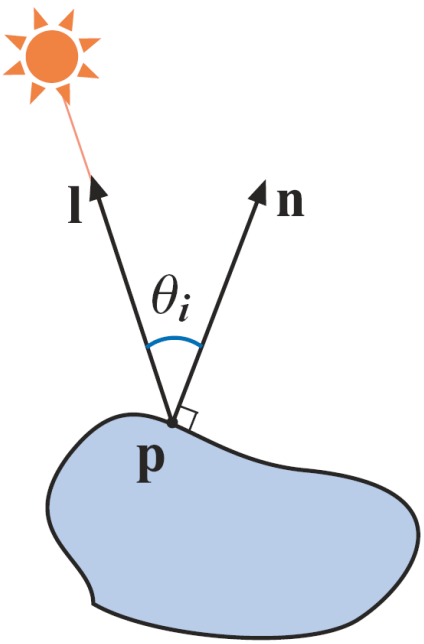


Figura 4.10: Incidencia de la luz en una superficie. El vector l indica la dirección de la luz y el vector n es la normal en el punto p.
(*Imagen de Real-Time Rendering* [2])

Debemos hacer notar que utilizaremos el término E para denotar a la irradiancia perpendicular al vector normal a la superficie n (irradiancia que recibe la superficie) y el término E_L, para referirnos a la irradiancia perpendicular a la dirección de la luz l (irradiancia de la fuente de luz). Por tanto, la irradiancia viene dada por la siguiente ecuación[4]:

$$E = E_L \overline{\cos}\theta_i = E_L max(n \cdot l, 0)$$

Como la irradiancia es aditiva, la irradiancia total de K fuentes de luz que recibe una superficie es la suma de la irradiancia de cada fuente de luz:

$$E = \sum_{k=1}^{K} E_{L_k} \overline{\cos}\theta_{i_k}$$

[4]El valor negativo del coseno corresponde al caso en el que la luz proviene de la parte trasera de la superficie. En este caso la luz no ilumina la superficie. Por lo tanto, en el área de los gráficos por computador se usa el coseno restringido a valores no-negativos: $\overline{\cos}\theta_i$.

4.6. Radiancia

Los sensores (como por ejemplo una cámara) reciben la luz emitida, que ha rebotado sobre diferentes partes de la escena, y la absorben para capturar la imagen. Los sensores, en realidad, están compuestos de muchos pequeños sensores (como por ejemplo fotodiodos en el caso de una cámara digital) donde cada uno de estos sensores detecta el valor de irradiancia en su superficie y produce una señal de color.

Sin embargo, los sensores de irradiancia realizan un promedio de los rayos de luz que vienen de todas las direcciones. Los sensores se encapsulan con una única *apertura* con el objetivo de restringir las direcciones desde las que la luz pueda entrar y además se coloca una *lente* en la apertura para enfocar la luz de manera que cada sensor solo reciba luz de un conjunto pequeño de direcciones entrantes. Estos sensores completos con apertura y lente miden la ***radiancia*** media, en vez de medir la irradiancia media. [2]

La radiancia (L) es la densidad del flujo de luz por área y por dirección entrante. Se representa como un vector RGB sin cotas de valores y se puede definir como *medir el brillo y color de un rayo de luz*. [2]

En el área de los gráficos por computador, también se mide la radiancia mediante cálculos usando un sensor virtual (aunque se trata de un modelo simplificado e idealizado de lo que sería un sensor real). La detección de la radiancia por el sensor se reemplaza por la evaluación de la ecuación de sombreado, cuyo objetivo es calcular la radiancia del rayo apropiado. La dirección de este rayo se representa mediante el *vector de vista v* (donde v es un vector unitario).

4.7. Materiales

Para mejorar la apariencia de los objetos de la escena, incluimos *materiales* en los modelos de la escena. Los materiales se utilizan para simular la interacción de la luz con el objeto. A cada material se le asocian ciertas propiedades para determinar sus características o comportamiento específico.

Todas las interacciones de la luz con la materia son el resultado de dos fenómenos llamados *dispersión* y *absorción*.

La ***dispersión*** se produce cuando la luz se encuentra con cualquier tipo de discontinuidad óptica (ver Figura 4.11). Este fenómeno da lugar a un cambio en la dirección de la luz.

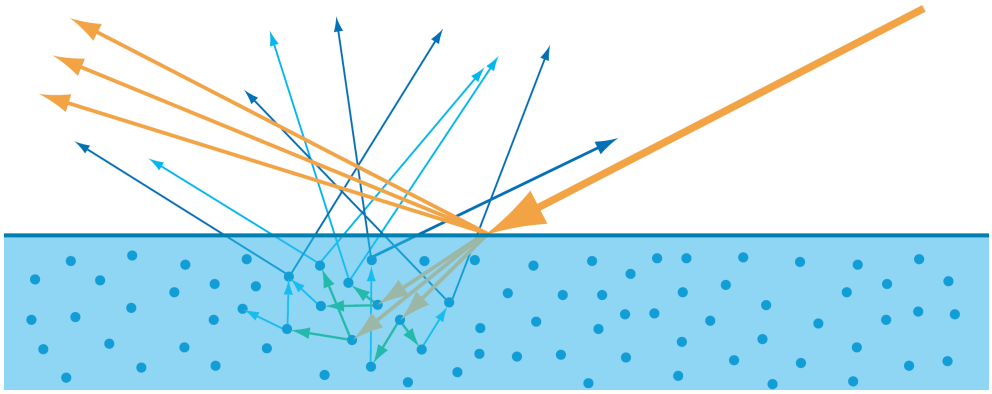


Figura 4.11: Interacciones de la luz en una superficie como resultado de la dispersión. La luz es dispersada en dos conjuntos de direcciones: hacia fuera de la superficie (reflexión) y hacia adentro (refracción o transmisión).
(*Imagen de Real-Time Rendering* [2])

En el proceso de renderizado, la interfaz entre el aire y el objeto en la superficie del modelo suele ser la discontinuidad óptica más importante. Las superficies dispersan la luz en dos conjuntos de direcciones distintos: hacia adentro de la superficie (fenómeno de *refracción* o *transmisión*) y hacia fuera de la superficie (fenómeno de *reflexión*). En el caso de la refracción o transmisión, cuando se trata de objetos transparentes, la luz transmitida continúa viajando a través del objeto; mientras que en el caso de objetos opacos, la luz transmitida se ve afectada por diversos eventos de dispersión y absorción hasta que finalmente una parte de la luz se refracta de vuelta a la superficie.

La ***absorción*** ocurre dentro de la materia y causa que parte de la luz se convierta en otro tipo de energía (por ejemplo, calor) y desaparezca. En contraposición con la dispersión, esto causa que la cantidad de luz se reduzca pero que no cambie su dirección.

Por otro lado, es necesario representar la cantidad y dirección de la luz saliente en función de la cantidad y dirección de la luz entrante para modelar el comportamiento del material mediante una ecuación de sombreado. La iluminación entrante se trata de la irradiancia en la superficie, mientras que la luz saliente se trata de la *emitancia* o *radiancia saliente* (cuyo símbolo es M). La relación entre la irradiancia y la emitancia es lineal.

En el área de los gráficos por ordenador, generalmente se incluyen dos términos en las ecuaciones de sombreado: el *término especular* y el *término difuso*. Esta distinción se realiza porque la luz reflejada en la superficie tiene una distribución de direcciones y de color distinta a la de la luz que pasa por las fases de transmisión, absorción y refracción de vuelta a la superficie. Por lo tanto, el término especular representa la luz reflejada

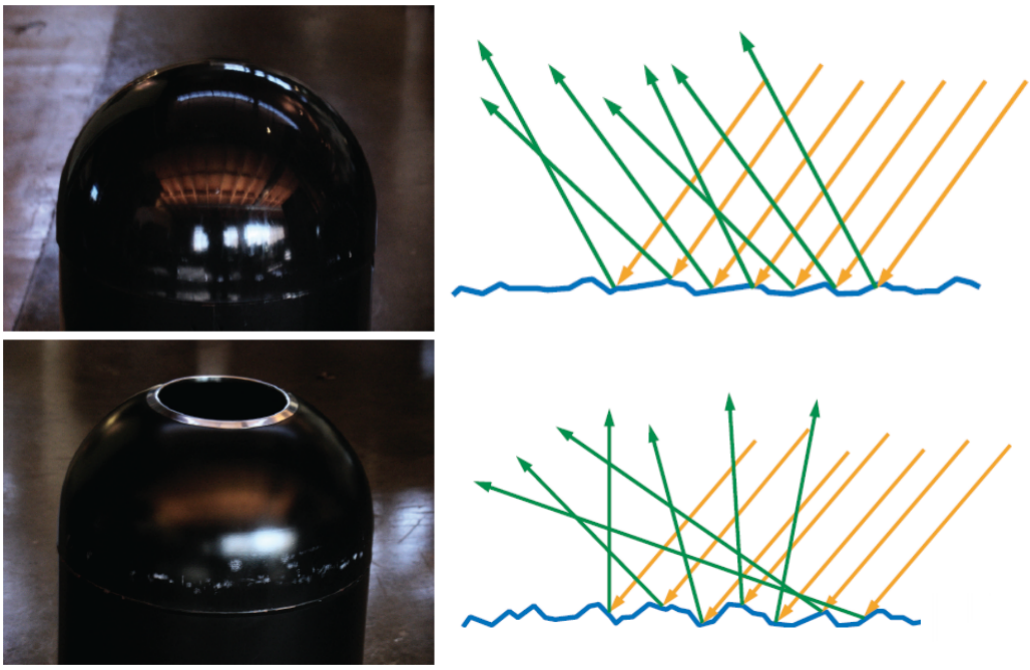


Figura 4.12: Superficies de distinta rugosidad presentan reflejos distintos.
(*Imagen de Real-Time Rendering* [2])

en la superficie, mientras que el término difuso representa la luz que es afectada por la transmisión, absorción y dispersión.

El *color de la superficie c* es una de las propiedades características que se asocia a un material. Como hemos dicho anteriormente, las ecuaciones de sombreado suelen tener dos términos distintos, especular (también llamado color especular o brillo c_{spec}) y difuso (conocido como color difuso c_{diff}). Estos dos términos dependen de la composición del material. El término especular depende de la dirección de la luz. Además, la distribución direccional del término especular depende de la rugosidad de la superficie. Las superficies lisas presentan reflejos muy detallados y realces brillantes, mientras que las superficies rugosas muestran reflejos borrosos y realces más apagados (ver Figura 4.12).

4.8. Reflectancia Fresnel

Las ecuaciones de Fresnel, desarrolladas por *Augustin-Jean Fresnel* en el siglo XIX, describen la reflexión y la transmisión de la luz en la superficie de contacto (plano de incidencia) de dos sustancias diferentes con distinto índice de refracción (por ejemplo, el aire y la

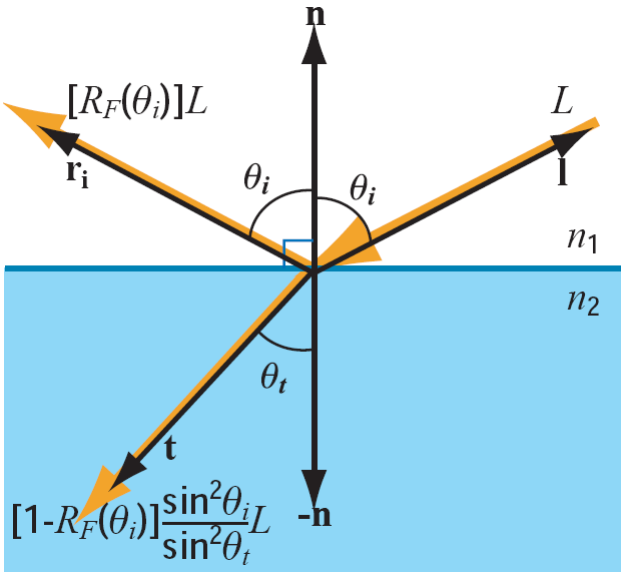


Figura 4.13: Reflejo y refracción en una superficie planar.
(*Imagen de Real-Time Rendering* [2])

materia del objeto a iluminar). Estas ecuaciones son un conjunto de relaciones matemáticas que relacionan las amplitudes de las ondas reflejadas y refractadas (o transmitidas) en función de la amplitud de la onda incidente. O sea, proporcionan los coeficientes de reflexión y transmisión de las ondas paralelas y perpendiculares al plano de incidencia. En un medio dieléctrico donde se pueda usar la ley de Snell (ver Figura fig:snell), para relacionar los ángulos de incidencia y transmisión, se pueden establecer las ecuaciones de Fresnel en términos de los ángulos de incidencia y transmisión.

Las ecuaciones Fresnel requieren de un plano perfecto, pero en realidad ninguna superficie es plana a nivel atómico. Las irregularidades menores que la menor longitud de onda de la luz (en el caso de la luz visible es de 400 nm) no tienen efecto en la luz, y por lo tanto las superficies que no presentan irregularidades mayores a esa longitud de onda, se consideran planos perfectos en términos de la óptica.

Una interfaz ópticamente plana entre dos sustancias es un caso especial de discontinuidad óptica, y produce que cada rayo de luz se disperse en exactamente dos direcciones: la *dirección de reflejo ideal* (indicada por el vector r_i) y la *dirección de refracción ideal* (indicada por el vector t). El vector r_i forma el mismo ángulo con la normal de la superficie n que el generado con la dirección de incidencia de la luz l (ángulo θ_i) y dicha normal. El vector t forma un ángulo distinto con la normal en dirección contraria $-n$ (ángulo θ_t). Ver Figura 4.13 para ilustración de estas variables.

El vector de reflejo r_i puede ser calculado mediante los vectores n y l:

$$r_i = 2(n \cdot l)n - l$$

La relación entre la radiancia incidente y la transmitida viene dada por la siguiente ecuación:

$$L_t = (1 - R_F(\theta_i))\frac{\sin^2\theta_i}{\sin^2\theta_t}L_i$$

La función de reflectancia Fresnel R_F es una función característica de cada superficie que depende del ángulo de incidencia de la luz θ_i, y tiene unas características particulares:

- El valor $R_F(0^\circ)$ es el *color especular característico de la sustancia* y aumenta a medida que el ángulo θ_i incrementa.
- El valor de $R_F(\theta_i)$ tiende a incrementar hasta llegar al color blanco (cuando el ángulo θ_i sea 90°).

Tanto el coeficiente de reflexión de Fresnel R_F como el ángulo de transmisión θ_t dependen del ángulo de incidencia θ_i y de una propiedad óptica de las dos sustancias llamada *índice de refracción*. La relación entre estos ángulos y los índices de refracción viene dada por la ecuación conocida como la ***ley de Snell***:

$$n_1 \sin(\theta_i) = n_2 \sin(\theta_t)$$

donde n_1 y n_2 son los coeficientes de refracción de ambos materiales.

Combinando la ley de Snell con la ecuación anterior obtenemos otra forma de representar la relación entre la radiancia incidente y la transmitida:

$$L_t = (1 - R_F(\theta_i))\frac{n_2^2}{n_1^2}L_i$$

La utilización de las ecuaciones de Fresnel en el área de gráficos por computador es escasa debido a su complejidad computacional. No obstante, existen aproximaciones como la *aproximación de Schlick* para la función de reflectancia Fresnel que son de uso extendido en este área.

La aproximación de Schlick nos proporciona una aproximación de la función de reflec-

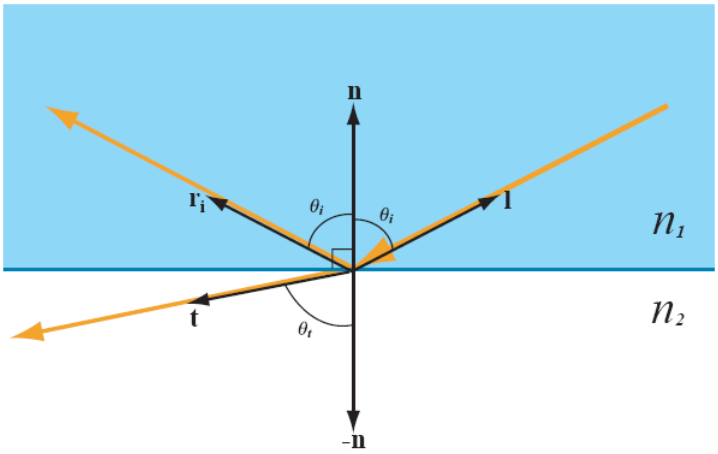


Figura 4.14: Reflexión interna en una superficie planar.
(*Imagen de Real-Time Rendering* [2])

tancia de Fresnel bastante precisa para la mayoría de sustancias:

$$R_F(\theta_i) \approx R_F(0^\circ) + (1 - R_F(0^\circ))(1 - \overline{\cos}\theta_i)^5$$

Los valores de índice de refracción para muchos materiales del mundo real están disponibles en diversas fuentes de información (por ejemplo, en la página RefractiveIndexINFO [12] disponemos de una base de datos de múltiples materiales listados mediante categorías). Mediante dichos índices de refracción se pueden calcular los valores $R_F(0^\circ)$. En el caso de que la interfaz sea el plano entre el aire y el material del objeto, se puede usar el índice de refracción de la sustancia (n_2, que llamaremos n en esta ecuación) para calcular su valor:

$$R_F(0^\circ) = (\frac{n-1}{n+1})^2$$

4.9. Microgeometría

La mayoría de las superficies presentan una rugosidad o estructura que afecta cómo se refleja la luz en ella. Estas irregularidades se encuentran a nivel microscópico (*microgeometría*). En el caso del rendering, son más pequeñas que la escala visible, es decir, son más pequeñas que un píxel. Sin embargo, estas estructuras microscópicas son significativamente más grandes que la longitud de onda de la luz visible y por lo tanto afectan a la

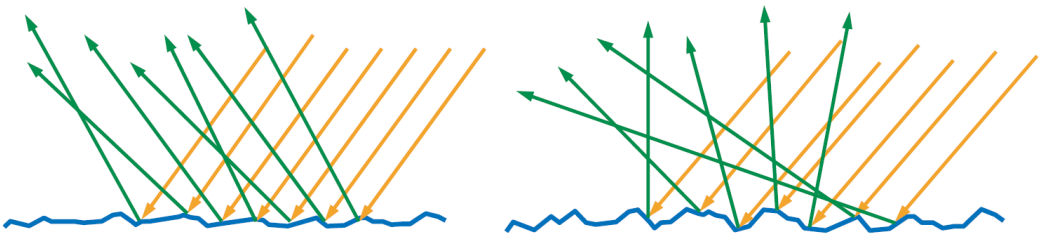


Figura 4.15: Luz reflectando en dos superficies con rugosidad distinta.
(*Imagen de Real-Time Rendering* [2])

interacción de la luz.

La estructura de la microgeometría causa que cada rayo de luz incidente se refleje en distintas direcciones (ver Figura 4.15). Este efecto visual se debe a que muchas normales están presentes en cada punto visible de la superficie, en vez de haber solamente una normal a nivel macroscópico. En rendering, modelamos este efecto mediante modelos estadísticos en la refracción de la luz desde la superficie.

Estas normales microscópicas se modelan estadísticamente como una distribución donde la rugosidad de la superficie controla la concentración de la distribución. Para la mayoría de las superficies, la distribución de normales microscópicas es *isotrópica* (es decir, rotacionalmente simétrica), pero otras superficies tienen una estructura *anisotrópica* lo cual da lugar a una distribución de normales anisotrópica y reflejos y realces emborronados direccionalmente (ver Figura 4.16).

Además de los efectos que tiene la microgeometría debido a la distribución de las normales, hay otras consecuencias importantes de la microgeometría a tener en cuenta (ver Figura 4.17):

- **Sombreado o *Shadowing*:** La estructura microscópica de la superficie puede crear oclusiones de la fuente de luz.
- **Enmascarado o *masking*:** La estructura microscópica de la superficie puede crear oclusiones de visibilidad.
- **Interreflexión o *interreflection*:** La estructura microscópica de la superficie causa que la luz se refleje varias veces antes de alcanzar al observador.

La teoría de *microfacetas* se basa en modelar la microgeometría como una colección de microfacetas donde cada una de ellas es un pequeño espejo Fresnel en la superficie,

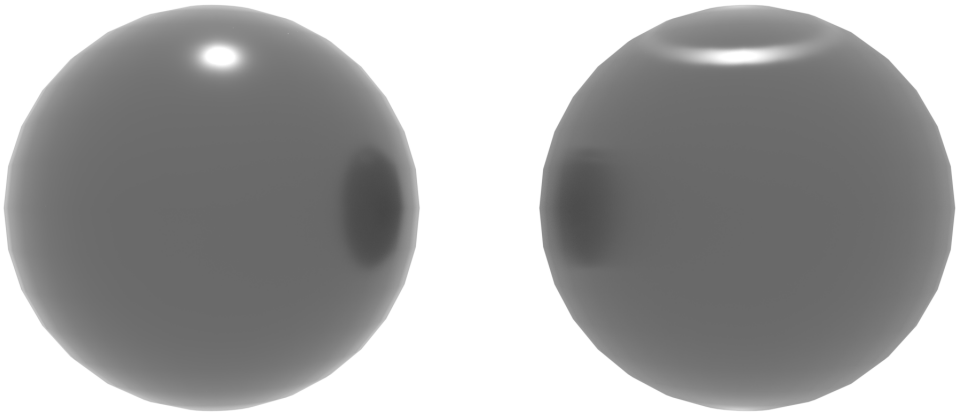


Figura 4.16: En izquierda una superficie con distribución isotrópica, en la derecha una superficie con distribución anisotrópica.

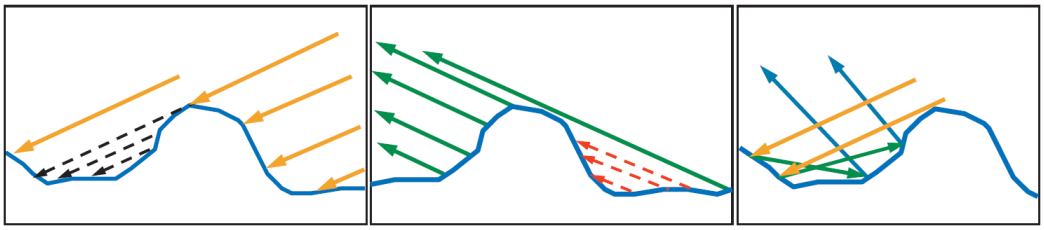


Figura 4.17: Efectos geométricos de la microgeometría. De izquierda a derecha: Sombreado, enmascarado e interreflexión.
(*Imagen de Real-Time Rendering* [2])

y la superficie se caracteriza por la distribución de las normales de las superficies de las microfacetas. Muchos modelos de iluminación basados en funciones de distribución de reflectancia bidireccional (*Bidirectional Reflectance Distribution Function*, BRDF) se basan en esta teoría, como veremos en un apartado posterior.

Por último, debemos hacer notar que la teoría de microfacetas se centra en modelar la reflexión especular en el primer rebote de luz, no modela múltiples rebotes o la reflectancia en el objeto.

4.10. Sombreado

El sombreado es el proceso de calcular la radiancia saliente L_o teniendo en cuenta la dirección de vista de la cámara (v) y las propiedades de los materiales y de las fuentes de luz mediante el uso de una ecuación de sombreado.

Se pueden emplear distintas ecuaciones de sombreado dependiendo del modelo de iluminación que se decida implementar. En este proyecto hemos utilizado ecuaciones para iluminación basadas en propiedades físicas utilizando modelos de iluminación basados en funciones de distribución de reflectancia bidireccional (*Bidirectional Reflectance Distribution Function*, BRDF). En nuestro caso, la ecuación de sombreado utilizada es:

$$L_0(v) = \sum_{k=1}^{n} f(l_k, v) \otimes E_{L_k} \overline{\cos}\theta_{i_k}$$

donde la función $f(l,v)$ es la función BRDF que decidamos usar para calcular el sombreado y el símbolo $\otimes$ denota una multiplicación de vectores componente a componente.

4.11. Modelos de iluminación BRDF

La función que describe cómo se reflecta la luz en una superficie se denomina función de distribución de reflectancia bidireccional (*Bidirectional Reflectance Distribution Function*, BRDF): $f(l,v)$. Esta función establece la relación que existe entre la radiancia saliente y la irradiancia. Es decir, describe cómo se refleja la luz desde una superficie dadas dos direcciones (la dirección de incidencia de luz l y la dirección de vista v). Su unidad

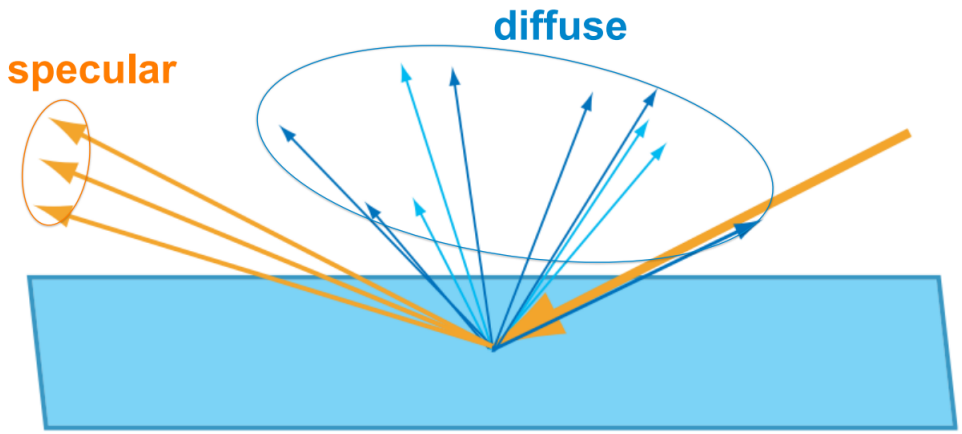


Figura 4.18: Componentes difusos y especulares.
(*Imagen de Background: Physics and Math of Shading [9]*)

es el esteorradián (*steradian*, sr).

$$f(l,v) = \frac{dL_o(v)}{dE(l)}$$

Las funciones BRDF constan de dos términos que modelan el término especular (fenómenos en la superficie, reflexión en la superficie) y el término difuso (fenómenos que ocurren en el interior del objeto, resultado de la transmisión, absorción y dispersión). Véase Figura 4.18.

Las funciones BRDF deben cumplir ciertas restricciones impuestas por las leyes de la física para ser consideradas físicamente plausibles:

- **Positividad:** Para toda luz entrante, el valor de la función BRDF debe ser no-negativo.
$$\forall l,v : f(l,v) \geq 0$$
- **Linealidad:** Los términos de la función BRDF (término especular y difuso) se combinan linealmente.
- **Reciprocidad de Helmholtz:** El intercambio de la dirección de incidencia con la dirección de vista no altera el valor de la función.
$$f(l,v) = f(v,l)$$

- **Conservación de energía:** La energía saliente de la superficie no puede ser superior a la energía incidente. Para medir esta energía, se utiliza la *función de reflectancia direccional-hemisférica* $R(l)$, cuya relación con la BRDF viene dada por:

  $$R(l) = \int_{\Omega} f(l,v) \cos\theta_o d\omega_o$$

  siendo θ_o el ángulo formado entre la normal a la superficie n y el vector de vista v.

  Se debe verificar que esta función sea menor o igual a la unidad para todo valor posible del vector dirección de la luz, l:

  $$\forall l : R(l) \leq 1$$

  Esto implica que los materiales no deben añadir energía ni pueden reflejar más luz de la que reciben y, como los valores de esta función se encuentran entre 0 y 1, si la función BRDF verifica la conservación de energía los materiales pueden absorber parte de la energía. De la misma manera, la energía debe "repartirse" entre los dos componentes del BRDF; esto es, la componente especular "quita" energía a la componente difusa.

Muchos modelos BRDF se basan en la teoría de microfacetas como comentamos anteriormente. Y por tanto, la función de distribución de normales (*Normal Distribution Function* o *NDF*) viene dada por la función de distribución de probabilidad de las normales de las superficies de las microfacetas. Esta función no define una microgeometría única, sin embargo captura el efecto visual más importante de la microgeometría.

Además, como ya comentamos en el apartado anterior, la teoría de microfacetas se enfoca en modelar la reflexión especular en el primer rebote de luz, no modela múltiples rebotes o la reflectancia en el objeto y, por lo tanto, a los términos BRDF basados en microfacetas se les añade un término difuso.

4.12. Modelo BRDF Cook-Torrance

El modelo BRDF Cook-Torrance [6], que fue presentado por *Robert L. Cook* y *Kenneth E. Torrance* en 1982, es un modelo matemático de reflexión para superficies basado en microfacetas.

El BRDF Cook-Torrance consta de un término difuso y un término especular:

$$f(l,v) = k_d * f_{dif}(l,v) + (1 - k_d) * f_{spec}(l,v)$$

donde k_d es una constante para regular la cantidad de la aportación especular que se usa en el material.

El término difuso es una constante:

$$f_{dif}(l,v) = \frac{albedo}{\pi}$$

siendo *albedo* el color del material[5,6].

El modelo especular es un BRDF de microfacetas:

$$f_{spec}(l,v) = \frac{D(h) * F(l,h) * G(l,v,h)}{4((n \cdot l)(n \cdot v))}$$

siendo $D(h)$ la función de distribución de normales, $G(l,v,h)$ el término geométrico, $F(l,h)$ el término de Fresnel y h el vector medio normalizado que se calcula como el vector medio entre la dirección de la luz l y el vector de vista v:

$$h = \frac{l+v}{||l+v||}$$

La función $D(h)$ es la función de distribución de Beckmann y viene dada por:

$$D(h) = e^{\frac{(n \cdot h)^2 - 1}{r^2 (n \cdot h)^2}} \frac{1}{\pi r^2 (n \cdot h)^4}$$

donde r es la rugosidad de la superficie (*roughness*).

Para el término de Fresnel se utiliza la función de aproximación de Schlick para la reflectancia Fresnel:

$$F(l,h) = R_F(0^\circ) + (1 - R_F(0^\circ)) * (1 - (l \cdot h))^5$$

[5]El término albedo viene de la astronomía y se utiliza para caracterizar a los planetas y los de elementos de la propia superficie de la Tierra. Se denomina albedo a la relación entre la luz reflejada y la luz incidente. Aunque el rango ideal es de 1.0 a 0.0, el albedo de los materiales corrientes varía entre 0.9 (nieve) a 0.04 (carbón). En simulación de materiales se utiliza a veces para referirse al comportamiento global de la superficie de un objeto, a su coeficiente medio de reflexión, principalmente en simulación de exteriores.

[6]El albedo es el porcentaje de radiación que cualquier superficie refleja respecto a la radiación que incide sobre la misma. Las superficies claras tienen valores de albedo superiores a las oscuras, y las brillantes más que las mates.

donde $R_F(0°)$ es la reflectancia de Fresnel del material cuando el ángulo es 0º.

Y el término de atenuación geométrica de Cook-Torrance es:

$$G(l,v,h) = \text{mín}\left(1, \frac{2(n\cdot h)(v\cdot n)}{(h\cdot v)}, \frac{2(n\cdot h)(l\cdot n)}{(h\cdot v)}\right)$$

En el caso de este modelo hay más variantes de estas funciones, pero estas son las típicas que se suelen utilizar y son muy similares a las de la publicación original, es por ello que las hemos escogido frente a otras posibles variantes.

4.13. Modelo BRDF Principled

El *modelo BRDF Principled* [4] fue presentado por *Brent Burley* en 2012 como el modelo de iluminación BRDF que se utiliza en *Walt Disney Animation Studios*. Se trata de un modelo complejo pero muy útil para modelar una gran variedad de materiales manteniendo un cierto control artístico.

Para el diseño del modelo BRDF, se tomaron en cuenta ciertos requisitos, a petición de los artistas, para que el modelo pudiera permitir ciertas concesiones artísticas y no ser estrictamente físico. Por lo tanto el modelo puede no ser "físicamente correcto" cuando se realizan ciertos cambios en los parámetros.

Los requisitos bajo los que se diseñó el modelo son:

1. Los parámetros usados deben ser intuitivos en vez de parámetros físicos.
2. El número de parámetros debe ser el menor posible.
3. El rango de los parámetros debe estar entre 0 y 1.
4. Los parámetros deben poder "ser llevados más allá" de su rango si tuviera sentido poder hacerlo.
5. Todas las combinaciones de los parámetros deben ser lo más robustas y plausibles posibles.

Los parámetros que tiene el modelo son un parámetro de color (RGB) y diez parámetros escalares:

- *baseColor*: Color de la superficie (RGB).
- *subsurface*: Controla el aspecto difuso usando una aproximación para la dispersión dentro de la superficie.
- *metallic*: Controla "lo metálico" que es el material (0 indica que es dieléctrico, 1 indica que es metálico). Es una combinación lineal entre dos modelos. El modelo metálico no tiene componente difusa y el especular "está teñido" por el color base **baseColor**.
- *specular*: Controla la cantidad de especularidad incidente. Este parámetro se calcula a partir del índice de refracción.
- *specularTint*: Controla cuánto se tiñe el color especular hacia el color **baseColor** (Concesión artística).
- *roughness*: La rugosidad de la superficie.
- *anisotropic*: Controla la cantidad de anisotropía de la distribución de microfacetas: 0 indica que la distribución es isotrópica, 1 que la distribución es totalmente anisotrópica.
- *sheen*: Añade una componente adicional para simular el aspecto de los tejidos.
- *sheenTint*: Controla cuánto se tiñe la componente para los tejidos hacia el color **baseColor**.
- *clearcoat*: Se trata de una capa especular adicional de propósito especial que da un efecto de capa acrílica.
- *clearcoatGloss*: Controla el brillo de la capa anterior.

El efecto de estos parámetros puede observarse en la figura 4.19.

El modelo difuso viene dado por:

$$f_{dif} = \frac{baseColor}{\pi}(1+(F_{D90}-1)(1-\cos\theta_l)^5)(1+(F_{D90}-1)(1-\cos\theta_v)^5)$$

donde F_{D90} es:

$$F_{D90} = 0{,}5 + 2\cos\theta_d^2 * roughness$$

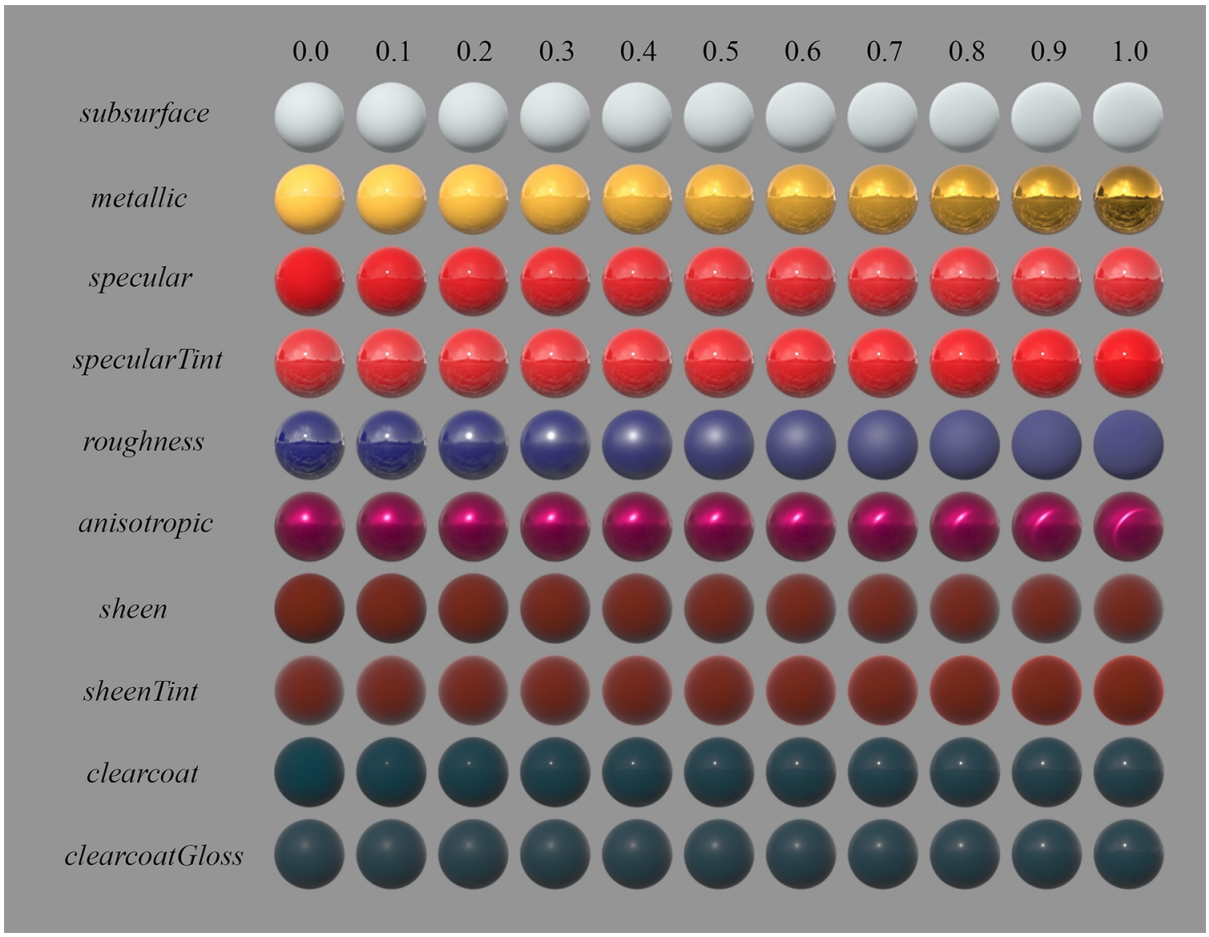


Figura 4.19: Efecto de los parámetros del BRDF Principled.
(*Imagen de Physically-Based Shading at Disney* [4])

siendo θ_l el ángulo formado entre los vectores l y n, θ_v el ángulo formado entre los vectores v y n y θ_d el ángulo formado entre el *vector medio h* y el vector v (o el ángulo formado entre h y l, debido a la simetría).

Si se realiza una interpolación lineal entre el modelo difuso y un modelo basado en el BRDF Hanrahan-Krueger [8] para aproximar un modelo físico de iluminación isotrópico basado en funciones de distribución de la reflectancia de la dispersión superficial (*Bidirectional Surface Scattering Distribution Function*, BSSRDF)[7], esta interpolación será controlada mediante el parámetro $subsurface$. En este caso, el modelo es el siguiente:

$$\begin{aligned} f_{ss} = 1{,}25 * ((1 + (F_{SS90} - 1) * \\ \left(1 - \cos\theta_l)^5\right) * \\ \left(1 + (F_{SS90} - 1)\,(1 - \cos\theta_v)^5\right) * \\ \left(\frac{1}{\cos\theta_l * \cos\theta_v} - 0{,}5\right) + 0{,}5) \end{aligned}$$

donde F_{SS90} viene dado por:

$$F_{SS90} = \cos\theta_d^2 * roughness$$

El modelo especular es un modelo de microfacetas:

$$f_{spec} = D(h)F(l,h)G(l,v,n)$$

donde $D(h)$ es la función de distribución de normales, $G(l,v,n)$ es el término geométrico y $F(l,h)$ es el término de Fresnel como ya apuntamos anteriormente.

El parámetro conocido como capa *clearcoat* es un componente especular adicional:

$$f_{ccoat} = D(h)F(l,h)G_{cc}(l,v,n)$$

Y el componente adicional denominado *sheen* es el siguiente:

$$f_{sheen} = ((1 - sheenTint) + sheenTint * baseColor)(1 - \cos\theta_d)^5$$

[7]Estos modelos calculan el modo en que la luz se dispersa en determinados medios como líquidos oleosos, leche, piel de poco espesor, ciertos tipos de mármol, etc.

En este modelo, la función de distribución de normales $D(h)$ es la distribución Trowbridge-Reitz generalizada (*Generalized-Trowbridge-Reitz* o *GTR*) que viene dada por:

$$D_{GTR} = c/(\alpha^2 \cos^2 \theta_h + \sin^2 \theta_h)^\gamma$$

La función tiene como parámetros una *constante de escalado c* y el exponente γ, la variable α es la rugosidad de la superficie (el parámetro *roughness*), aunque en este caso usaremos $\alpha = roughness^2$ debido a que el algoritmo BRDF *Principled* lo utiliza de esta manera para que resulte un cambio lineal más perceptible en la rugosidad. [4]

En cuanto a las constantes, para el paramétro *specular* se usa $\gamma = 2$, mientras que para la capa *clearcoat* se usa $\gamma = 1$. Ya que no escalamos la función, la constante c es $c = 1$. El ángulo θ_h es el ángulo formado entre el vector medio h y el vector normal n.

Las funciones utilizadas en este trabajo son la función D_{GTR_2} para la componente especular o la función $D_{GTR_{2aniso}}$ para la componente especular en caso de usar la distribución anisotrópica y la función D_{GTR_1} para la capa especular adicional *clearcoat*. Para esta capa, el valor de α está en el rango [0.1-0.001] y será controlado por el parámetro $clearcoatGloss$:

$$D_{GTR_2}(h) = \frac{\alpha^2}{\pi} \frac{1}{(1+(\alpha^2-1)(h\cdot n)^2)^2}$$

$$D_{GTR_{2aniso}}(h) = \frac{1}{\pi} \frac{1}{\alpha_x \alpha_y} \frac{1}{((h\cdot x)^2/\alpha_x^2 + (h\cdot y)^2/\alpha_y^2 + (h\cdot n)^2)^2}$$

$$D_{GTR_1}(h) = \frac{\alpha^2-1}{\pi \log \alpha^2} \frac{1}{(1+(\alpha^2-1)(h\cdot n)^2)}$$

En el caso de la distribución anisotrópica se introducen cuatro nuevas variables α_x, α_y, x e y. Las variables x e y son el vector tangente y bitangente respectivamente. Las variables α_x y α_y se calculan mediante una tercera variable denominada $aspect$ y la rugosidad de la siguiente manera:

$$aspect = \sqrt{1 - 0{,}9 * anisotropic}$$

$$\alpha_x = roughness^2/aspect$$

$$\alpha_y = roughness^2 * aspect$$

Para el término de Fresnel, $F(l,h)$, se utiliza la función de aproximación de Schlick para

la reflectancia Fresnel que viene dado por:

$$F(l,h) = R_F(0^\circ) + (1 - R_F(0^\circ)) * (1 - (l \cdot h))^5$$

donde $R_F(0^\circ)$ es la reflectancia fresnel del material cuando el ángulo es 0º.

Para el término geométrico, $G(l,v,n)$, utilizaremos la aproximación de la función de sombreado GGX [16] por Smith:

$$G(l,v,n) = G_{SmithGGX}(l,n)G_{SmithGGX}(v,n)$$
$$G_{SmithGGX}(v,n) = \frac{1}{(v \cdot n) + \sqrt{((v \cdot x) * \alpha_x)^2 + ((v \cdot y) * \alpha_x y)^2 + (v \cdot n)^2}}$$

siendo el parámetro $\alpha_g = (0{,}5 + roughness/2)^2$.

No obstante, para la capa adicional *clearcoat* utilizaremos como término geométrico la función G GGX con una rugosidad fija de $\alpha_g = 0{,}25$:

$$G_{cc}(l,v,n) = G_{GGX}(l,n)G_{GGX}(v,n)$$
$$G_{GGX}(v,n) = \frac{1}{(v \cdot n) + \sqrt{\alpha_g^2 + (v \cdot n) - \alpha_g^2 * (v \cdot n)}}$$

La función BRDF completa es la siguiente:

$$f = (1 - metallic) * f_{dielectric} + metallic * f_{metal} + 0{,}25 * clearcoat * f_{ccoat}$$

donde:

$$f_{dielectric} = (1 - subsurface) * f_{dif} + subsurface * f_{ss} + sheen * f_{sheen} + ((1 - specularTint) + specularTint * baseColor) * f_{spec}$$
$$f_{metal} = baseColor * f_{spec}$$

5. CAPÍTULO

Desarrollo del proyecto

5.1. Algoritmo principal

El proyecto tiene como objetivo el renderizado de una escena, por ello, la aplicación consta de dos partes principales (Figura 5.1): la inicialización (creación de la escena) y el bucle de renderizado.

5.1.1. Inicialización

En esta fase de inicialización, se crea la escena completa incluyendo cámara, luces y objetos que forman parte de la escena. En esta parte del proceso, también se genera la interfaz para controlar ciertas características de algunos componentes de la escena.

Los objetos pueden ser creados mediante primitivas proporcionadas por el software o bien se pueden cargar las escenas almacenadas en ficheros del tipo ".json". Además, se crean los materiales de dichos objetos que pueden contener texturas las cuales se especifican normalmente en este tipo de ficheros ".json". La cámara utilizada en este proyecto es una cámara en perspectiva y tiene controles asociados para ser manipulada mediante el ratón.

El proceso puede observarse en la Figura 5.2.

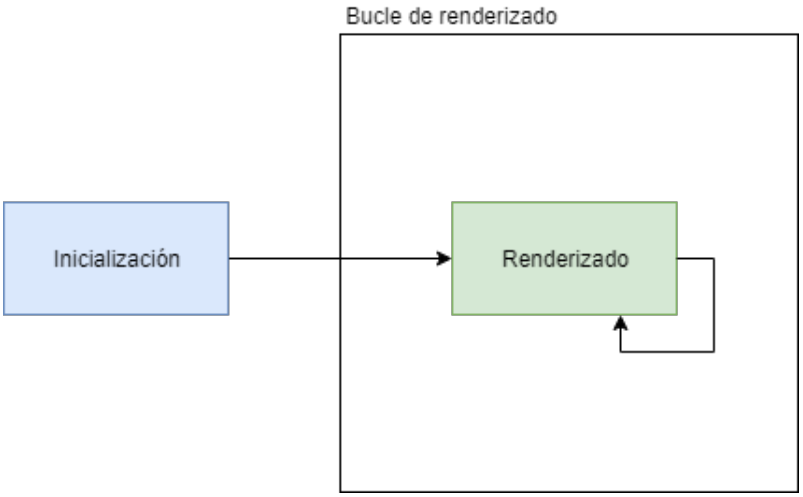


Figura 5.1: Estructura de la aplicación.

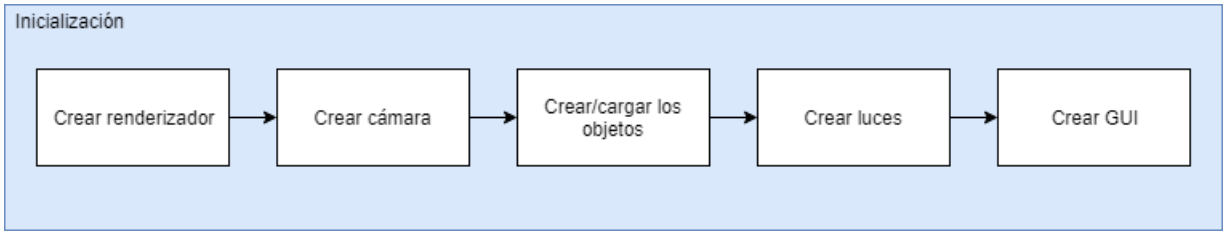


Figura 5.2: Estructura de la inicialización.

Creación del renderizador

El *renderizador* o *visualizador* es un componente de la aplicación que permite tomar una imagen de la escena en un instante de tiempo. Para tomar una imagen de la escena se llama a la función *render()* del visualizador. Inicialmente, ThreeJS crea la escena y la cámara y, posteriormente, genera una imagen de la escena usando los programas *shaders* correspondientes a los especificados en el material de cada objeto.

Creación de la cámara

La *cámara* es el componente de la aplicación que captura la imagen de la escena. Se ha utilizado una cámara de perspectiva con un ángulo de apertura focal de 75º.

Creación/Carga de los objetos

Se crean los objetos y sus materiales mediante primitivas o ficheros auxiliares de tipo ".json", que almacenan toda o parte de la escena, y se localizan y se orientan en el espacio según los criterios del usuario.

Creación de las luces

La escena consta de 5 luces: Una luz ambiental, dos luces direccionales, una luz puntual y una luz spot. Todas las luces, a excepción de la luz direccional que es roja, son de color blanco.

Creación de la interfaz de usuario (GUI)

La aplicación consta de una interfaz que permite la modificación de todas las luces de la escena, así como de las posiciones y orientaciones de todos los objetos de la escena. Mediante esta interfaz también pueden modificarse todos los atributos de los materiales asociados a los objetos (ver Figura 5.3).

5.1.2. Bucle de renderizado

Una vez creada la escena completa, se ejecuta un bucle de renderizado en el cual se pueden modificar algunos de los atributos de los elementos de la escena y, posteriormente,

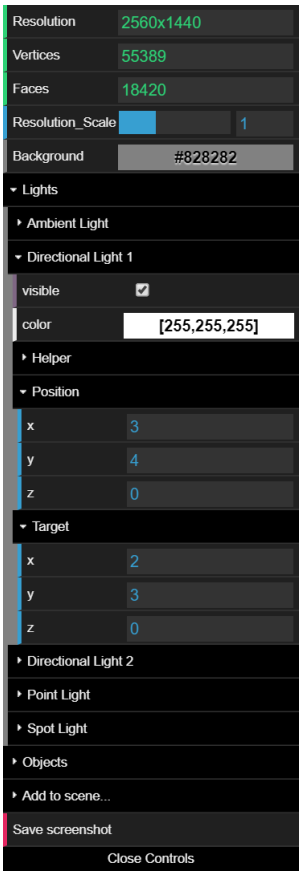


Figura 5.3: Interfaz de usuario de la aplicación.

se renderiza la escena completa. El renderizador realiza las fases de *Geometría* y *Rasterizado* del pipeline de renderizado, utilizando los *vertex-* y *fragment-shaders* especificados en los materiales de cada objeto para realizar el sombreado de vértices y el sombreado de píxel respectivamente. Los modelos de iluminación BRDF se implementan en estos *shaders* e incluyen todos los cálculos necesarios para evaluar las ecuaciones de sombreado específicas de los BRDFs.

5.2. Funciones principales

Nuestro proceso de renderizado consiste en dos funciones: *animate()* y *render()*. La función *animate()* es la encargada de realizar el bucle de renderizado. Dentro de este bucle se pueden realizar varias acciones, como por ejemplo actualizar posiciones para realizar una animación, modificar la escena, etc. En nuestro caso, también realizamos la actualización de los contadores de fotogramas y del *frame-time*). En la parte final del bucle se llama a la función *render()* para generar la imagen de la escena. Una vez realizada esta tarea, termina la iteración y comienza la siguiente y así sucesivamente.

5.3. Librerías utilizadas

Hemos utilizado las siguientes librerías JavaScript para el desarrollo del proyecto:

- *ThreeJS*: Una librería para crear y mostrar escenas 3D mediante WebGL.
- *BufferGeometryUtils*: Una librería de ThreeJS para añadir funciones adicionales - como por ejemplo, calcular tangentes para la geometría - para utilizar con los objetos de tipo *BufferGeometry*.
- *OrbitControls*: Una librería para añadir control de la cámara de ThreeJS mediante el ratón.
- *datGUI*: Una librería para crear interfaces gráficas que permitan modificar variables JavaScript.
- *stats*: Una librería para mostrar estadísticas sobre el renderizado como el número de fotogramas por segundo (FPS) y el *frame-time*.

5.4. Vertex-shader y fragment-shader

Para calcular la iluminación, hemos programado las ecuaciones de sombreado en los *vertex* y *fragment shaders*. La estructura general que siguen estos *shaders* es la misma para los dos modelos BRDF que hemos implementado.

En el caso del *vertex-shader*, realizamos los cálculos necesarios (en el sistema de referencia de la cámara) a fin de obtener las variables de tipo *varying* que posteriormente se usarán en el *fragment-shader*. Además se calcula la posición de los vértices en el sistema de referencia adecuado (el sistema de referencia de la proyección) ya que es necesario para el correcto renderizado de la escena:

Datos: Matrices de transformación (matriz de proyección mPr, matriz de sistema de referencia del objeto a sistema de referencia de la cámara $mObjACam$), posición del vértice vP, normal del vértice vN.
Resultado: Posición del vértice en el sistema de referencia de proyección pos, posición del vértice en el sistema de referencia de la cámara p, vector de vista v, normal del vértice en el sistema de referencia de la cámara n.
inicio
p $\leftarrow premultiplicarMatrizAPunto(vP, mObjACam)$;
v $\leftarrow -p$;
n $\leftarrow premultiplicarMatrizAVector(vN, mObjACam)$;
pos $\leftarrow premultiplicarMatrizAPunto(p, mPr)$;
fin

Algoritmo 1: Programa *Vertex-shader*

En el *fragment shader* calculamos el color de cada píxel visible de los objetos visualizados por la cámara de la escena, mediante la ecuación de sombreado y los datos interpolados

incluidos en las variables de tipo *varying*.

Datos: Posición del píxel en el sistema de referencia de la cámara p, vector de vista v, normal del píxel en el sistema de referencia de la cámara n, para cada luz de la escena: el color de la luz c_{L_k}, posición de la luz k-ésima p_{L_k} (si tiene posición), dirección de la luz k-ésima d_{L_k} (si tiene dirección).
Resultado: Color del píxel c.
inicio
    c $\leftarrow (0,0,0)$;
    para *luz k* **hacer**
        l $\leftarrow calcularVectorDeLuz(k)$;
        $E_L \leftarrow calcularIrradianciaDeLuz(k)$;
        $BRDF \leftarrow f(l,v)$;
        c $\leftarrow c + (BRDF * E_L * max(n \cdot l, 0))$;
    fin
fin

Algoritmo 2: Programa *Fragment-shader*

Como el vector de luz l se calcula de forma distinta para cada tipo de luz, la función *calcularVectorDeLuz* abstrae la implementación de este cálculo. Así mismo, la irradiancia de la luz también se calcula de manera distinta dependiendo del tipo de luz. Por lo tanto, la función *calcularIrradianciaDeLuz* también abstrae la implementación de este otro cálculo. Para terminar, independientemente de qué modelo BRDF escojamos, debemos calcular la función BRDF $f(l,v)$ y tener en cuenta los términos difusos y especulares

de acorde al modelo de iluminación escogido.

Datos: Vector de luz l, vector de vista v, color del material *albedo* y la constante k_d del material.
Resultado: Valor de la función BRDF r.
inicio

$h \leftarrow normalizar(l+v)$;
$c_{diff} \leftarrow albedo/\pi$;
$d \leftarrow D(h)$;
$f \leftarrow F(l,h)$;
$g \leftarrow G(l,v,h)$;
$c_{spec} \leftarrow d * f * g/(4(n \cdot l)(n \cdot v))$;
$r \leftarrow k_d * c_{diff} + (1-k_d) * c_{spec}$;

fin

Algoritmo 3: Función BRDF para el BRDF Cook-Torrance

En el caso del modelo BRDF Cook-Torrance, calcularemos la función BRDF mediante el algoritmo 3. En este caso la función $D(h)$ será la distribución de Beckmann, la función $F(l,h)$ será la función de aproximación de Schlick para reflectancia Fresnel y la función $G(l,v,h)$ será el término de atenuación geométrica de Cook-Torrance. Hemos descrito

estas funciones anteriormente en la sección 4.12 *Modelo BRDF Cook-Torrance*.

Datos: Vector de luz l, vector de vista v.

Resultado: Valor de la función BRDF r.

inicio

$h \leftarrow normalizar(l+v)$;

$c_{diff} \leftarrow interpolar(f_{dif}(), f_{ss}(), subsurface)$;

$c_{sheen} = f_{sheen}()$;

$d_{spec} \leftarrow D_{GTR2_{aniso}}(h)$;

$f_{spec} \leftarrow F(l,h)$;

$g_{spec} \leftarrow G(l,v,n)$;

$c_{spec} \leftarrow d_{spec} * f_{spec} * g_{spec}$;

$d_{cc} \leftarrow D_{GTR1}(h)$;

$f_{cc} \leftarrow F(l,h)$;

$g_{cc} \leftarrow G_{cc}(l,v,n)$;

$c_{clearcoat} \leftarrow d_{cc} * f_{cc} * g_{cc}$;

$f_{dielectric} \leftarrow c_{diff} + sheen * c_{sheen} + interpolar(1, baseColor, specularTint) * c_{spec}$;

$f_{metal} \leftarrow baseColor * c_{spec}$;

$r \leftarrow interpolar(f_{dielectric}, f_{metal}, metallic) + 0{,}25 * clearcoat * c_{clearcoat}$;

fin

Algoritmo 4: Función BRDF para el BRDF Principled

En cuanto al modelo BRDF Principled, calcularemos la función BRDF mediante el algoritmo 4. Donde la función $f_{dif}()$ será el modelo difuso, la función $f_{ss}()$ será el modelo *subsurface*, la función $f_{sheen}()$ será el componente adicional *sheen*, la función $D_{GTR2_{aniso}}(h)$ será la función de distribución *Generalized-Trowbridge-Reitz* anisotrópica, la función $F(l,h)$ será la función de aproximación de Schlick para reflectancia Fresnel, la función $G(l,v,n)$ será el término geométrico, la función $D_{GTR1_{aniso}}(h)$ será la función de distribución *Generalized-Trowbridge-Reitz* también y la función $G_{cc}(l,v,n)$ será el término geométrico para la capa *clearcoat*. Hemos descrito todos estos modelos y funciones anteriormente en la sección 4.13 *Modelo BRDF Principled*. La función $interpolar(a,b,fac)$ realizará una interpolación lineal entre a y b con un factor fac de la siguiente manera: $(1-fac)*a+fac*b$.

6. CAPÍTULO

Análisis del proyecto

6.1. Complicaciones

Durante el proyecto han surgido pocas complicaciones y fueron principalmente en la fase de implementación de la aplicación. A continuación, describimos brevemente estas dificultades o problemas encontrados:

- Problemas cuando se cargan los *shaders* en ThreeJS:

  En los navegadores web modernos no se puede acceder a ficheros del disco local desde código JavaScript. Por lo tanto, no se pueden cargar directamente los *shaders* desde los ficheros de código fuente (los archivos **.vert* y **.frag*). Para solucionar este problema he programado un *script* en Python que, utilizando los ficheros fuente de los *shaders*, crea un archivo JavaScript en el que el código de los *shaders* se encuentra almacenado en variables de tipo *String*, de manera que se pueden utilizar directamente con ThreeJS.

- Problemas a la hora de incluir las variables uniformes en los materiales para su uso en los *shaders*:

  La documentación de ThreeJS sobre variables uniformes para los materiales era muy escasa en el momento del desarrollo. Por lo tanto, tuve que acceder al código fuente de ThreeJS y a varios ejemplos en su repositorio para comprender su funcionamiento y aclarar cómo se pueden utilizar estas variables en los programas escritos en GLSL. Esto fue un pequeño imprevisto pero no conllevó mucho tiempo.

- Problemas a la horar de modificar los valores de las variables uniformes mediante la interfaz GUI:

  Las variables uniformes de los materiales de ThreeJS no pueden modificarse directamente desde los controles de la interfaz. Para solucionar este inconveniente, se han creado variables intermedias a la hora de programar la interfaz y se ha creado una función denominada *onChange()* para modificar las los valores de las variables uniformes cuando queramos cambiar las variables intermedias mediante los controles de la interfaz.

6.2. Resultados

Los resultados del proyecto han sido buenos. La implementación ha dado lugar a una aplicación bastante modular que permite la visualización de escenas y un ajuste extenso y detallado de sus objetos y sus correspondientes materiales. Los modelos de iluminación implementados en los *shaders* son bastante completos acorde al diseño estudiado. No obstante, debido al tiempo limitado del proyecto, no se han implementado características como por ejemplo sombras u otros efectos que se utilizan en gráficos por computador en tiempo real de manera estándar. En un futuro es posible expandir esta implementación para añadir dichas características modificando sólo los módulos necesarios o los *shaders*.

En cuanto al rendimiento del renderizado, hemos realizado pruebas con un equipo con una GPU NVIDIA GeForce GTX 970 a una resolución de 2560x1440. La aplicación fue capaz mantener los 60 FPS en una escena con objetos con una gran cantidad de polígonos (en total en la escena aproximadamente 40 millones). El uso de GPU fue de un 83 % durante esta prueba. También se añadieron un total de 5 esferas (con aproximadamente 8 millones de polígonos cada esfera) usando el *shader* BRDF más complejo de nuestra aplicación: *Principled* BRDF.

En la segunda prueba, la aplicación mantuvo también 60 FPS siendo el uso de GPU de un 8 %. La escena constaba de 20 esferas con una cantidad de polígonos mucho menor. Esta escena se utilizó para evaluar el efecto de los distintos parámetros del *shader Principled* BRDF.

A la vista de estos resultados, considero que el rendimiento de la aplicación es bastante óptimo y que las imágenes obtenidas mediante estos modelos de iluminación basados en la física son de alta calidad y de gran realismo.

En las siguientes figuras se pueden observar escenas renderizadas para las pruebas de la aplicación.

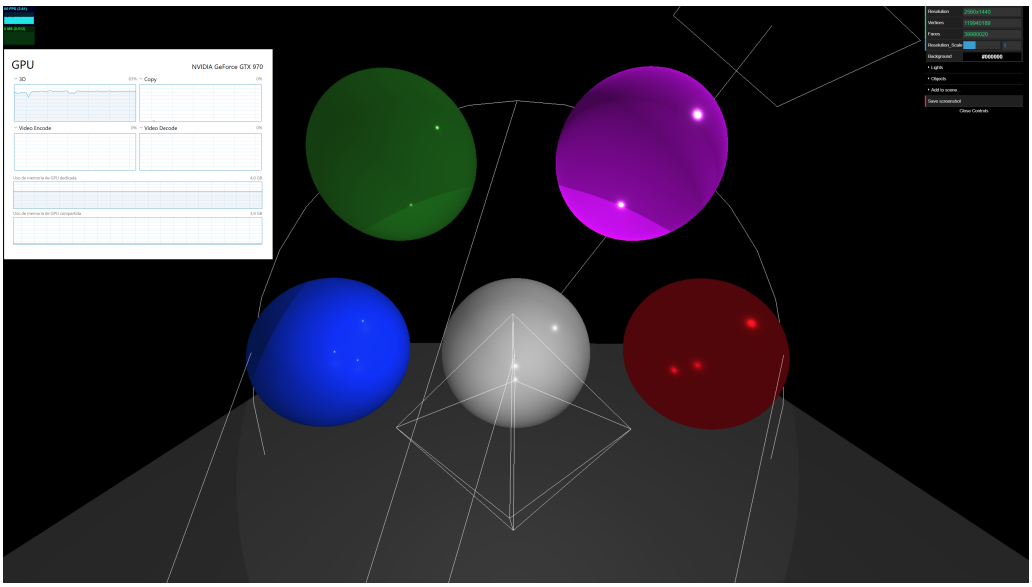


Figura 6.1: Prueba realizada con objetos con una gran cantidad de polígonos. Aproximadamente 40 millones de polígonos en la escena.

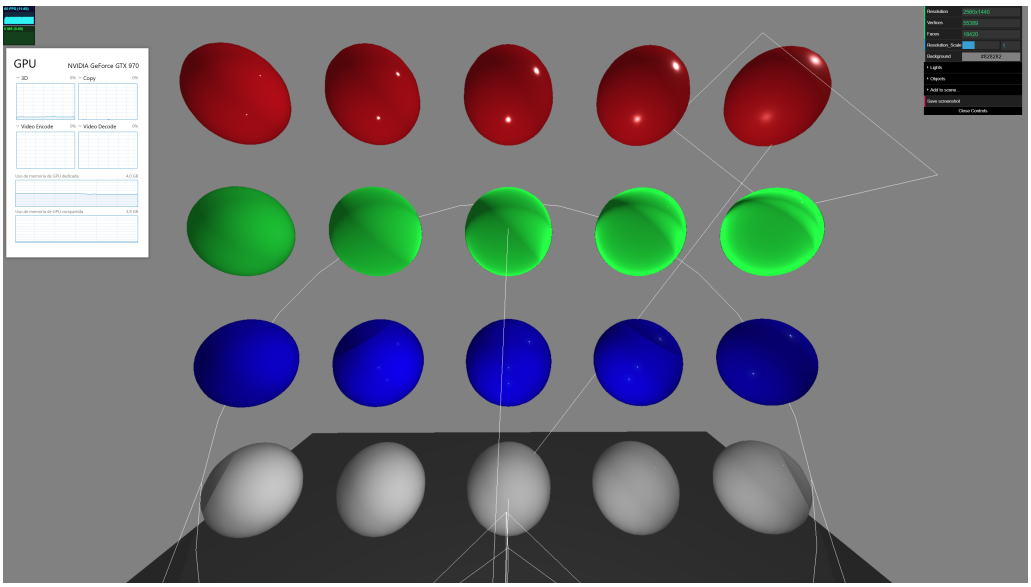


Figura 6.2: Prueba realizada con muchos objetos para probar los distintos parámetros. Las esferas rojas muestran el efecto del parámetro *roughness* (donde $specular = 1$), las verdes reflejan el efecto del parámetro *subsurface*, las azules muestran el efecto del parámetro *clearcoat* (siendo $clearcoatGloss = 1$) y las grises reflejan el efecto del parámetro *sheen* (con $sheenTint = 1$).

7. CAPÍTULO

Conclusiones

7.1. Conclusiones generales del proyecto

Los objetivos principales del proyecto se han cumplido ya que hemos implementado una aplicación con dos modelos de iluminación basada en física como fue establecido al comienzo de este proyecto: Uno inicial más sencillo (el modelo BRDF Cook-Torrance) y uno más complejo y completo (el modelo BRDF *Principled*).

Antes de comenzar la implementación hemos estudiado los conceptos básicos (también expandiendo los conocimientos aprendidos anteriormente en asignaturas ya cursadas) para comprender las técnicas utilizadas en la iluminación basada en física para gráficos por computador. También hemos estudiado en profundidad las técnicas que se utilizan en este ámbito.

En el proceso de implementación hemos aprendido a utilizar la librería ThreeJS, que se trata de una herramienta muy potente y con potencial para ser usada en futuros proyectos. Además, hemos vuelto a hacer uso del lenguaje de los *shaders* GLSL, lenguaje que ya utilicé anteriormente en una asignatura de la rama de Computación. Mi experiencia previa nos ha permitido trabajar más ágilmente con este lenguaje y profundizar un poco más en su uso.

En general, ha sido un proyecto muy interesante. He aprendido y he profundizado mucho en conceptos estudiados anteriormente sobre gráficos por computador, lo cual establece una base que puede servirme para futuros proyectos aún mayores.

7.2. Líneas futuras

Los modelos de iluminación analizados e implementados en este proyecto no tienen en cuenta las interacciones sub-superficie, por lo tanto, un paso obvio de mejora es estudiar los modelos BSDF, que tienen en cuenta la dispersión que ocurre debajo de la superficie del objeto.

Por otro lado, se puede mejorar la aplicación para conseguir una apariencia mejor en el renderizado, ya que debido al tiempo limitado del proyecto se ha implementado lo relevante al proyecto en la aplicación. Se pueden implementar diversas técnicas adicionales que son usadas actualmente en gráficos por computador en tiempo real en la aplicación actual, por ejemplo: *Sombras* (ya que la implementación actual carece de ellas), *oclusión ambiental* para mejorar la percepción de profundidad de las escenas y que la visualización sea más realista, *reflejos de otros objetos* (reflejos del espacio de pantalla en el caso de renderizado en tiempo real) para añadir realismo a los materiales reflectivos, efectos fílmicos como *profundidad de campo*, *resplandor* o *luces volumétricas*, entre otras... Todas estas ampliaciones pueden dar lugar a una aplicación más completa y realista.

La base estudiada para la realización de este proyecto no sólo sirve para gráficos en tiempo real, también se puede implementar una aplicación para *renderizado offline* utilizando técnicas de *ray-tracing*, lo cual puede ser interesante para conseguir imágenes muy realistas con las escenas adecuadas.

Anexos

A. ANEXO

Tipos de variables GLSL

En este anexo se recogen descripciones de los tipos de variables GLSL utilizados en la implementación del proyecto. GLSL dispone de diversos tipos de datos utilizables y se puede consultar una amplia lista en la página de referencia de OpenGL [7] o la lista de tipos de variables GLSL de Shaderific [15].

A.1. Tipos escalares

Tipo	Descripción
void	Tipo de datos utilizado para indicar que una función no devuelve ningún valor
int	Tipo de datos numérico entero
float	Tipo de datos numérico de coma flotante (precisión simple)

A.2. Tipos vectoriales

Tipo	Descripción
vec3	Vector compuesto por tres elementos float
vec4	Vector compuesto por cuatro elementos float

A.3. Tipos matriciales

Tipo	Descripción
mat3	Matriz 3x3 de elementos float
mat4	Matriz 4x4 de elementos float

A.4. Estructuras

El tipo de datos ***struct*** se usa para definir *estructuras*. Con las estructuras definiremos un tipo de variables propio compuesto de múltiples campos de distintos tipos. El funcionamiento de las estructuras en **GLSL** es idéntico al funcionamiento de las estructuras en **C**. Por ejemplo podríamos definir una estructura *posición* que esté compuesta por tres elementos enteros:

```
struct posicion{
    int x;
    int y;
    int z;
}
```

De esta manera al declarar una variable del tipo posición, está dispondrá de tres campos *x*, textity y *z*. Podremos utilizar cada campo como cualquier otra variable, por ejemplo:

```
posicion unaPosicion;

// Ejemplo de una asignación
unaPosicion.x = 10;

// Ejemplo de aplicarle una función
int aux = unaFuncion(unaPosicion.x);
```

B. ANEXO

Funciones predefinidas en GLSL

En este anexo se recogen descripciones de las funciones GLSL utilizadas en la implementación del proyecto. GLSL dispone de diversas funciones predefinidas utilizables y se puede consultar una amplia lista en la página de referencia de OpenGL [7] o la lista de funciones predefinidas en GLSL de Shaderific [14].

B.1. Funciones geométricas

Función	Descripción
vec3 normalize (vec3 x) vec4 normalize (vec4 x)	Normaliza el vector x, haciendo que sea de longitud unitaria en la misma dirección.
vec3 dot (vec3 x, vec3 y)	Realiza el producto escalar entre los vectores x e y.

B.2. Funciones exponenciales

Función	Descripción
float pow(float x, float y)	Calcula x^y.
float sqrt(float x)	Calcula $\sqrt{x}$.
float log(float x)	Calcula $\log x$.

B.3. Otras funciones

Función	Descripción
float max(float x, float y)	Devuelve: x si $x \geq y$ o y si $y > x$.
vec3 mix(vec3 a, vec3 b, float f)	Realiza una interpolación lineal entre los vectores a y b. El factor f controla la interpolación: $f * a + (1 - f) * b$

C. ANEXO

Fichero HTML base para la aplicación

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset=utf-8>
        <meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0,
     maximum-scale=1.0">
        <title>Aplicación WebGL</title>
        <style>
            body { margin: 0; overflow: hidden }
            canvas { width: 100%; height: 100% }
        </style>
    </head>
    <body>
    </body>
</html>
```

D. ANEXO

Inclusión de una librería JavaScript

Para incluir una librería JavaScript en la aplicación incluiremos el siguiente código en la sección ***body*** del documento HTML:

```
<script type="text/javascript" src="*NOMBRE_DE_FICHERO*"></script>
```

Donde **NOMBRE_DE_FICHERO** es el nombre del fichero de la librería que se quiere cargar. Del mismo modo también se podrá cargar código JavaScript que separemos en ficheros distintos.

E. ANEXO

Creación de una escena básica en ThreeJS

Antes de comenzar, hay que crear un contenedor en el fichero HTML donde ThreeJS colocará el *canvas*. Un canvas es un elemento HTML sobre el que se pueden dibujar imágenes (mediante JavaScript o en nuestro caso, WebGL). Para crearlo, añadiremos la siguiente línea en la sección ***body*** del documento HTML:

```
<div id="container"></div>
```

A continuación hay que crear lo necesario para poder mostrar la escena a crear: un renderer, añadir el canvas del renderizador de ThreeJS al contenedor creado anteriormente, la cámara y la función de render. En el proyecto esto se ha realizado en un fichero llamado *init.js*, incluido dentro de la carpeta *code*.

```
// Crear el renderer y ajustar su tamaño al tamaño de la ventana
var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);

// Añadir el canvas al contenedor
document.getElementById('container').appendChild(renderer.domElement);

// Crear la cámara y colocarla en una posición para que se pueda ver la escena
var camera = new THREE.PerspectiveCamera(75, window.innerWidth/window.innerHeight, 0.1, 1000);
camera.position.z = 5;

// Crear la función de render
var render = function (scene)
{ renderer.render(scene, camera); };
```

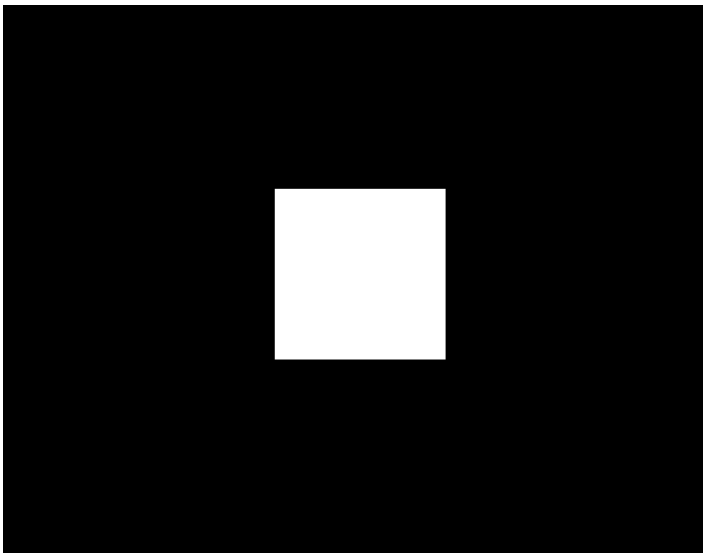


Figura E.1: Escena básica creada en *ThreeJS*.

Mediante el siguiente código JavaScript creamos una escena básica con un cubo y un material simple, también creamos el bucle de renderizado de la escena. En el proyecto esto se ha realizado en un fichero separado para cada escena, estos ficheros se encuentran en la carpeta *scenes*.

```
// Crear la escena
var scene = new THREE.Scene();

// Crear la geometría del cubo
var geometry = new THREE.BoxGeometry(1, 1, 1);

// Crear un material básico para añadir al cubo
var material = new THREE.MeshBasicMaterial({color: 0xffffff});

// Crear el mesh del cubo y añadirlo a la escena
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);

// Bucle de renderizado
function animate() {
  requestAnimationFrame(animate);
  render(scene);
}
animate();
```

Al finalizar se verá una escena idéntica a la mostrada en la figura E.1.

F. ANEXO

Añadir luces a la escena con ThreeJS

ThreeJS dispone de varios tipos de fuentes de luz predefinidas, entre ellas: luces ambientales (*AmbientLight*), luces direcionales (*DirectionalLight*), luces posicionales (*PointLigth*) y luces spot (*SpotLight*).

Para añadir una luz a la escena, hay que crear un objeto del tipo de la luz que se desee añadir. No es necesario añadir parámetros ya que los inicializa con un valor por defecto aunque no se especifique[1].

```
1 var pointLight = new THREE.PointLight();
2 scene.add(pointLight);
```

Tras crear la luz y añadirla a la escena se podrán modificar las variables del objeto cuando se desee. De hecho, se pueden añadir a la GUI para un control más sencillo.

Si se deseara representar la luz visualmente para saber su color, posición (si tiene), dirección (si tiene), etc. ThreeJS dispone de objetos llamados *helpers* definidos para cada tipo de luz. Estos objetos se pueden añadir de la siguiente forma:

```
1 var pointLightHelper = new THREE.PointLightHelper(pointLight);
2 scene.add(pointLightHelper);
```

[1] Si los quisiéramos añadir, los parámetros y su descripción se encuentran disponibles para consultar en la documentacion sobre luces de ThreeJS: https://threejs.org/docs/index.html#api/lights/Light

G. ANEXO

Crear materiales con shaders personalizados en ThreeJS

Los materiales se crean mediante un objeto *ShaderMaterial* de ThreeJS y después se añade ese objeto a la geometría (malla) para añadir a la escena.

Se definirá un material de la siguiente manera:

```
var materialShader = new THREE.ShaderMaterial({
  uniforms: THREE.UniformsUtils.merge([THREE.UniformsLib['lights'],
  {
      */ Las variables uniform van aquí /*
  }]),
  lights: true,
  vertexShader: *VERTEX_SHADER_A_UTILIZAR*,
  fragmentShader: *FRAGMENT_SHADER_A_UTILIZAR*
});
```

Deberemos añadir las variables uniform a usar en los shader en la zona comentada y las variables que contengan el código de los vertex y fragment shaders en un string en **VERTEX_SHADER_A_UTILIZAR** y **FRAGMENT_SHADER_A_UTILIZAR** respectivamente.

ThreeJS añadirá mediante *UniformsLib* todos los datos sobre las luces en variables uniform, si añadimos lo siguiente en los shader se podrán utilizar:

```
struct DirectionalLight {
  vec3 color;
  vec3 direction;
};

```

```
struct PointLight {
  vec3 color;
  vec3 position;
};

struct SpotLight {
  vec3 color;
  vec3 position;
  vec3 direction;
  float coneCos;
  float penumbraCos;
  float decay;
};

uniform vec3 ambientLightColor;

#if NUM_DIR_LIGHTS > 0
uniform DirectionalLight directionalLights[NUM_DIR_LIGHTS];
#endif
#if NUM_POINT_LIGHTS > 0
uniform PointLight pointLights[NUM_POINT_LIGHTS];
#endif
#if NUM_SPOT_LIGHTS > 0
uniform SpotLight spotLights[NUM_SPOT_LIGHTS];
#endif
```

Tras crear el material con el shader, lo asignaremos a la geometría al crear el objeto mesh. Por ejemplo:

```
var cube = new THREE.Mesh(geometry, materialShader);
scene.add(cube);
```

H. ANEXO

Creación de una interfaz con datGUI

Para añadir la interfaz a la aplicación, añadimos el siguiente código en el principio del código de la escena:

```
gui = new dat.GUI();
```

Una vez realizado eso, se pueden incluir controles para las variables de un objeto mediante los métodos *.add()* o *.addColor()* (para que muestre un selector de color en caso de que la variable sea un color). Cabe destacar que los controles aparecen en el orden en el que se añadan a la interfaz o carpeta correspondiente, así que el orden en el que se utilicen estos métodos es importante si se desea una interfaz ordenada. A continuación, se ilustra el uso de estos métodos añadiendo un control para la rotación en el eje *X* del cubo de la escena y añadiendo un control para modificar el color guardado en una variable en formato hexadecimal:

```
// Control para la rotación del eje x del cubo
gui.add(cube.rotation, "x");

// Objeto con una variable de color en hexadecimal
var obj = {color: "0x000000"};

// Control para el color de la variable
gui.addColor(obj, "color");
```

Además, los controles disponen de una propiedad llamada *onChange* que puede ser per-

sonalizada para que ejecute una función cada vez que se cambie el valor del control. Se ilustra el uso de esta propiedad con el control para la rotación del eje *X* del cubo:

```
gui.add(cube.rotation, "x").onChange(
function (){
    // Código a ejecutar cuando se modifique el valor
});
```

Para terminar, la interfaz permite añadir carpetas para agrupar controles en grupos. Esto se realiza mediante el método *.addFolder()*. Además, los métodos *.open()* y *.close()* permiten abrir o cerrar las carpetas mediante código (por defecto, se crean cerradas). A continuación, se ilustra el uso de las carpetas con el ejemplo anterior:

```
// Carpeta para controles del cubo
var controlesCubo = gui.addFolder("Controles para el cubo");

// Carpeta para controles del color
var controlesColor = gui.addFolder("Colores");

// Añadir algunos controles para el cubo
controlesCubo.add(cube.rotation, "x");
controlesCubo.add(cube.rotation, "y");
// Dejar la carpeta abierta
controlesCubo.open();

var obj = {color1: "0x000000", color2: "0xFFFFFF"};

// Añadir controles para los colores
controlesColor.addColor(obj, "color1");
controlesColor.addColor(obj, "color2");
// No se hace .open() así que aparecerá cerrada por defecto
```

I. ANEXO

Creación de un gráfico de monitorizado con stats

Para añadir un gráfico de monitorización añadiremos el siguiente código al principio del código de la escena:

```
var stats = new Stats();
stats.showPanel(*ÚNMERO_DE_REFERENCIA*);
document.body.appendChild(stats.dom);
```

Reemplazaremos **NÚMERO_DE_REFERENCIA** por uno de los siguientes valores dependiendo del gráfico que queramos utilizar:

- 0: Mostrar gráfico de monitorización de FPS.
- 1: Mostrar gráfico de monitorización de tiempo de ejecución.
- 2: Mostrar gráfico de monitorización de uso de memoria.
- 3: Mostrar gráfico personalizado[1].

Una vez añadido dicho código, deberemos modificar el comienzo y final de la función *animate()* de la escena para que la monitorización funcione correctamente:

```
function animate() {
  // Añadir la siguiente línea al comienzo:
```

[1] No se ha utilizado en este proyecto. En la documentación oficial de la librería se explica cómo utilizarlo y posibles casos de uso.

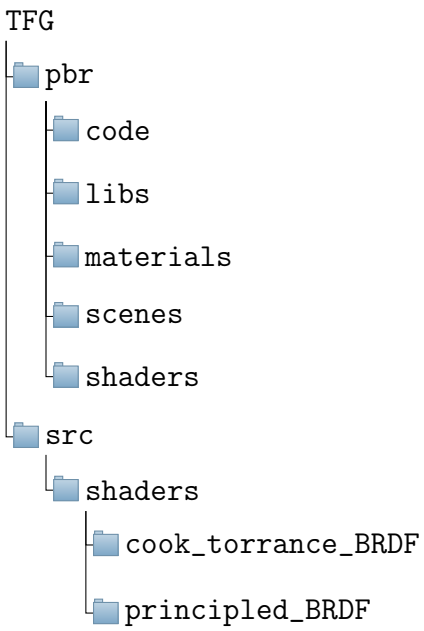
```
3     stats.begin();
4
5     */ El código sin modificar de la función original va aquí ... /*
6
7     // Añadir la siguiente línea al final:
8     stats.end();
9   }
```

J. ANEXO

Código del proyecto

J.1. Organización del proyecto

Para tener el proyecto más organizado, se ha creado la siguiente estructura de carpetas:



En la carpeta *pbr* se encuentran los ficheros de la aplicación, para abrir la aplicación abriremos el fichero *pbr.html*. Dentro de la carpeta *pbr*, en la carpeta *code* se encuentra todo el código JavaScript principal de la aplicación, en la carpeta *libs* se encuentran las librerías JavaScript utilizadas en el proyecto, en las carpetas *materials* y *scenes* se encuentran los

materiales y escenas respectivamente, y finalmente, en la carpeta *shaders* se encuentra el fichero *shaders.js* que contiene todos los shaders en variables JavaScript. Por otro lado, en la carpeta *src/shaders* se encuentran los ficheros fuente (.vert y .frag) de los shaders GLSL.

J.2. Descarga del código del proyecto

El código del proyecto está disponible en el siguiente repositorio GitHub: TFG-PBR.

En este repositorio también está disponible el enlace a una demo online.

Bibliografía

[1] Three.js documentation: https://threejs.org/docs/.

[2] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[3] Brent Burley. Extending the Disney BRDF to a BSDF with integrated subsurface scattering. *Physically Based Shading in Theory and Practice SIGGRAPH Course*, 2015.

[4] Brent Burley and Walt Disney Animation Studios. Physically-based shading at Disney. In *ACM SIGGRAPH*, volume 2012, pages 1–7, 2012.

[5] Guillaume Caurant and Thibault Lambert. The Lighting Technology of 'Detroit: Become Human'. In *GDC Vault*.

[6] Robert L Cook and Kenneth E Torrance. A reflectance model for computer graphics. In *ACM Siggraph Computer Graphics*, volume 15, pages 307–316. ACM, 1981.

[7] The Khronos Group. OpenGL 4 reference pages. https://www.khronos.org/registry/OpenGL-Refpages/gl4/.

[8] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 165–174. ACM, 1993.

[9] Naty Hoffman and 2K. Background: Physics and math of shading. In *SIGGRAPH*, 2012.

[10] Jon Irwin. 25 years on, devs reflect on the influence and impact of Star Fox. *Gamasutra Article*.

[11] Miguel Jorge. Pioneros del CGI: historia y evolución de los FX en el cine. *Gizmodo*, Mar 2016.

[12] Mikhail Polyanskiy. Refractive index database. https://refractiveindex.info/.

[13] Christophe Schlick. An inexpensive BRDF model for Physically-based Rendering. In *Computer graphics forum*, volume 13, pages 233–246. Wiley Online Library, 1994.

[14] Shaderific. GLSL functions. http://www.shaderific.com/glsl-functions.

[15] Shaderific. GLSL types. http://www.shaderific.com/glsl-types.

[16] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 195–206. Eurographics Association, 2007.