Universidad Zaragoza
1542

Trabajo Fin de Grado

Renderizado transitorio de materiales con retrasos temporales

Transient rendering of time delaying materials

Autor

David Jiménez Omeñaca

Directores

Sergio Cartiel Embid

Jorge García Pueyo

Ponente

Adolfo Muñoz Orbañanos

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2025

Renderizado transitorio de materiales con retrasos temporales

RESUMEN

La velocidad de la luz es tan alta que su propagación resulta imperceptible para el ser humano. Por este motivo, en campos como la informática gráfica, se suele asumir que la luz se desplaza de forma instantánea, es decir, con velocidad infinita. Sin embargo, la aparición de la **femto-fotografía** (captura de imágenes a escalas de picosegundos) dió lugar al campo de la imagen transitoria, donde la información relativa al tiempo de viaje de la luz se utiliza para resolver problemas como la estimación de la profundidad. Pese a su potencial, esta tecnología depende de equipamiento muy costoso, lo cual limita la experimentación. En respuesta a esta limitación, el **renderizado transitorio** surge como una alternativa que simula computacionalmente cómo viaja la luz a lo largo del tiempo.

En este tipo de simulaciones, es crucial considerar cómo interactúan los distintos materiales con la luz. La mayoría de los renderizadores transitorios asumen que los materiales reemiten la luz de manera instantánea. Sin embargo, en la realidad existen materiales, como aquellos con **fluorescencia**, que reemiten la luz con un retraso temporal.

El objetivo de este trabajo es ampliar el software **Mitransient**, una extensión del renderizador basado en físicas **Mitsuba 3** al estado transitorio, para incluir la capacidad de simular materiales con retrasos en la reemisión de la luz.

Para alcanzar este objetivo, en primer lugar hemos desarrollado los fundamentos matemáticos que permiten modelar este tipo de materiales. A continuación, hemos implementado las herramientas necesarias para definir materiales transitorios en el entorno de Mitsuba 3, incluyendo materiales comunes en la naturaleza. Este trabajo sienta así las bases para la definición de materiales transitorios más complejos, con el fin de lograr simulaciones cada vez más fieles al comportamiento real de la luz en el dominio temporal.

Índice

Capítulo 1

Introducción y objetivos

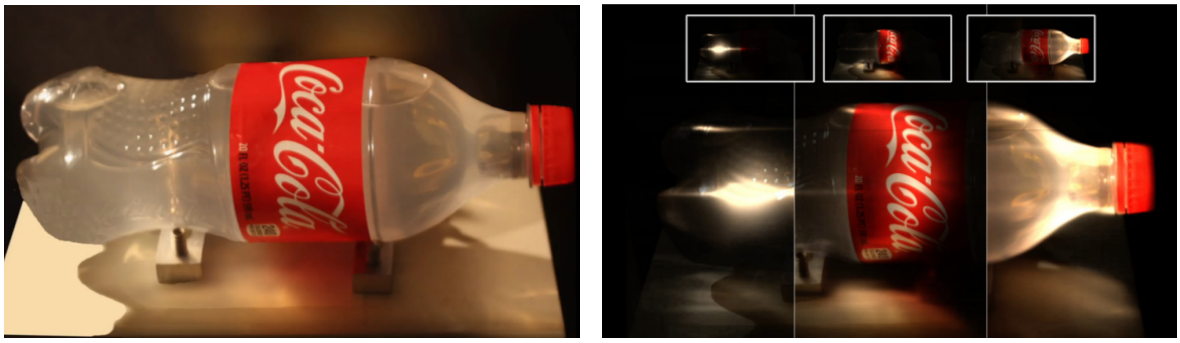
Durante toda la historia, el ser humano ha buscado representar su entorno: desde las pinturas rupestres del Paleolítico hasta la invención de la fotografía a mediados del siglo XIX, que permitió capturar imágenes detalladas tal y como nuestros ojos perciben el mundo. Sin embargo, ni nuestra percepción visual ni las primeras cámaras eran capaces de mostrar el proceso dinámico de la luz, que no actúa de forma instantánea, sino que viaja, rebota, se refracta y se dispersa al interactuar con las superficies, todo ello en escalas de tiempo extremadamente pequeñas.

Posteriormente, en los años 60 surgió el **renderizado por ordenador** (*rendering*), que permitía generar imágenes sintéticas simulando el comportamiento de la luz en escenas virtuales, lo que revolucionó campos como el cine, la arquitectura o los videojuegos. Estos sistemas, en general, suponían que la velocidad de la luz era infinita, produciendo resultados visuales estáticos en el tiempo, al igual que nuestros ojos o estas primeras cámaras.

Sin embargo, la luz no viaja de forma instantánea. En el vacío, su velocidad es de unos 300,000 kilómetros por segundo, lo que la hace parecer casi infinita a simple vista. No obstante, en campos como la astronomía, las distancias involucradas son tan grandes que hacen que esta velocidad sea relevante. Por ejemplo, la luz de las estrellas más lejanas tarda millones de años en llegar hasta nosotros, permitiéndonos ver el universo tal como era en el pasado.

Imagen Transitoria. En 2013, Velten *et al.* presentaron la llamada **femto-fotografía** [1] que, combinando láseres pulsados, sensores ultra-rápidos y computación, lograba capturar billones de imágenes por segundo. Este avance permitió, por ejemplo, visualizar cómo un pulso de luz atraviesa una botella de plástico fotograma a fotograma (ver Figura 1.1), lo que inauguró el campo de la imagen transitoria (*transient imaging*) [2].

Las aplicaciones de esta tecnología son incontables, como la estimación de



(a) (b)

Figura 1.1: Fotografías capturadas mediante **femto-fotografía** [1] de una botella de Coca-Cola iluminada por un pulso de luz. (a) Imagen estática de la escena. (b) Secuencia que muestra la propagación del pulso de luz a través de la botella. Extraído de Jarabo *et al.* [3].

profundidad [4], captura de reflectancias [5] o la visión de escenas ocluidas a través de esquinas (*Non-line-of-sight* o NLOS) [6, 7, 8, 9]

Sin embargo, la imagen transitoria presenta desafíos prácticos. La captura directa de la luz a resoluciones temporales tan pequeñas requiere hardware altamente especializado, como cámaras ultra-rápidas y láseres de alta precisión, cuyos costes ascienden fácilmente a cientos de miles de euros. Esto limita su accesibilidad y dificulta su uso en muchas áreas de investigación y desarrollo.

Renderizado Transitorio. Es aquí donde el renderizado transitorio (*transient rendering*) [3] juega un papel crucial. A diferencia del *rendering* convencional, este tiene en cuenta la velocidad finita de la luz, lo que permite representar cómo los rayos de luz viajan, interactúan con los materiales y son percibidos en intervalos temporales específicos. Este enfoque amplía las capacidades de simulación y modelado, haciéndolo especialmente útil para estudiar fenómenos ópticos avanzados.

Materiales con retrasos temporales. No obstante, el concepto de *transient rendering* tiene una complejidad intrínseca. Algunos materiales exhiben **retrasos temporales** en la reemisión de la luz, lo que complica significativamente la simulación. Este fenómeno exige técnicas avanzadas para modelar con precisión cómo la luz interactúa con dichos materiales, especialmente en escenarios donde los efectos temporales son cruciales. Un ejemplo de material que presenta este fenómeno son los **materiales fluorescentes**. Estos absorben la luz incidente, emitiendo parte de esta instantáneamente en una frecuencia concreta, mientras que el resto la emiten en otra frecuencia tras un tiempo específico. Este efecto se puede ver en la Figura 1.2.

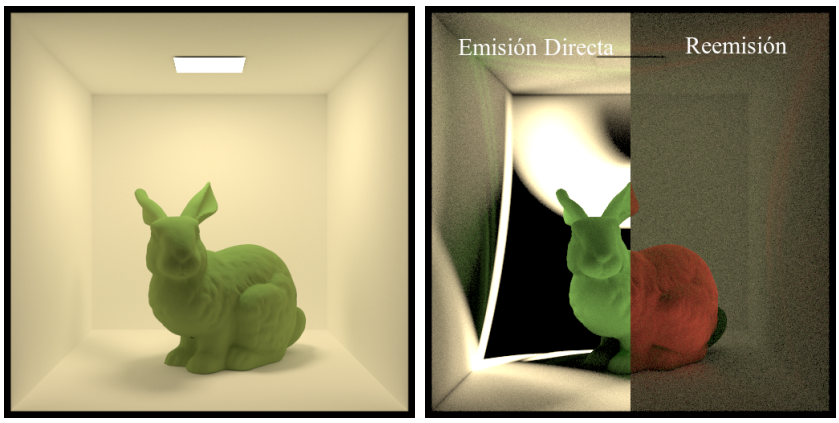


Figura 1.2: Representación gráfica de un conejo fluorescente. **(Izquierda)** imagen estática del conejo (como lo vería el ojo humano). **(Derecha)** Pulso emitido instantáneamente al incidir la luz (color verde) y reemisión tras un tiempo concreto en otra frecuencia (color rojo).

Objetivos. Este proyecto se enfoca en modelar materiales con retrasos temporales en el contexto de la informática gráfica. Aunque estos fenómenos se han estudiado en óptica y física de materiales, su incorporación en motores transitorios ha sido poco explorada o incluso ignorada por la complejidad que introduce. El objetivo principal de la presente memoria es tanto desarrollar una base teórica rigurosa para modelar estos fenómenos (Capítulos 2, 3), como desarrollar una implementación funcional que permita simularlos en motores gráficos (Capítulos 4, 5).

La implementación se ha basado en extender el sistema de renderizado físico de **Mitsuba 3** [10] junto a su extensión transitoria **Mitransient** [11] para permitir definir materiales con retrasos temporales e implementar ciertos materiales transitorios reales, como los observados en fenómenos de fluorescencia.

Contribuciones. La contribución de este trabajo es doble: por un lado, se ha desarrollado una nueva formulación matemática que representa de manera genérica los materiales con retrasos temporales y, por otro lado, se ha implementado dicho modelo dentro de **Mitsuba 3** y **Mitransient**. Esto sienta las bases para futuras extensiones que implementen materiales transitorios complejos para obtener simulaciones más precisas.

El código implementado es *open-source* y se puede consultar en Github. Asimismo, el material complementario de la memoria (videos transitorios) es accesible desde Google Drive. Finalmente, la planificación final del trabajo se puede ver en el Apéndice C.

Este trabajo ha sido parcialmente financiado por el Programa de Becas y Ayudas del Instituto de Investigación en Ingeniería de Aragón (I3A).

Capítulo 2

Conocimiento Previo

En el campo de la computación gráfica, el **renderizado basado en física** es un enfoque fundamental para obtener imágenes fotorrealistas. Inicialmente, se desarrolló asumiendo que la velocidad de la luz es infinita, denominado **renderizado estacionario** y, posteriormente, se extendió al **renderizado transitorio**, donde se tiene en cuenta la velocidad finita de la luz y el dominio temporal del transporte de luz.

Esta sección abarca el marco teórico necesario para entender ambos enfoques. Por ello, en la Sección 2.1 se desarrollará la base matemática del transporte de luz estacionario. A continuación, en la Sección 2.2, se abordará el concepto de transporte de luz transitorio, adaptando las definiciones previas a este nuevo dominio temporal. Finalmente, la Sección 2.3 introducirá el concepto de material transitorio, el cual describe materiales cuya reemisión de luz no es instantánea, sino que experimenta un retraso temporal.

2.1. Transporte de luz estacionario

El **renderizado basado en física** [12] busca simular con precisión cómo la luz se comporta al interactuar con distintos materiales, apoyándose en las leyes físicas del transporte de luz. Es una herramienta fundamental en áreas que requieren imágenes fotorrealistas generadas por ordenador, como la animación digital, los videojuegos o la arquitectura.

Para lograr dicho realismo, es necesario modelar con precisión cómo la luz se transporta en una escena tridimensional y cómo contribuye a la apariencia final de los objetos. La unidad básica de energía usada es la **radiancia** $L(\mathbf{x}, \omega)$, que describe cuánta luz incide en el diferencial de área dA alrededor del punto $\mathbf{x}$ desde el cono de direcciones (ángulo sólido) diferencial $d\Omega$ alrededor de ω y se mide en $Wm^{-2}sr^{-1}$.

La ecuación de render

La **ecuación de render** [13] es la ecuación fundamental para la generación de imágenes por ordenador. Esta ecuación describe la radiancia saliente $L_o(\mathbf{x},\ \omega_o)$ desde un punto $\mathbf{x}$ en una dirección ω_o como la suma entre la radiancia que emite el propio material en esta dirección, denotada por $L_e(\mathbf{x},\ \omega_o)$, y la radiancia reflejada (por otras fuentes de luz), $L_r(\mathbf{x},\ \omega_o)$. A su vez, la luz reflejada se puede expresar como la integral de la **radiancia incidente** $L_i(\mathbf{x},\ \omega_i)$ sobre las direcciones ω_i en la semiesfera unitaria Ω centrada en $\mathbf{n_x}$:

$$L_o(\mathbf{x},\ \omega_o) \ = \ L_e(\mathbf{x},\ \omega_o) \ + \overbrace{\int_\Omega L_i(\mathbf{x},\ \omega_i) f_r(\mathbf{x},\ \omega_o,\ \omega_i) |\mathbf{n_x} \cdot \omega_i| d\omega_i}^{L_r(\mathbf{x},\omega_\mathbf{o})}, \tag{2.1}$$

donde la función $f_r(\mathbf{x},\ \omega_o,\ \omega_i)$ es la función de distribución de dispersión bidireccional (*bidirectional scattering distribution function,* o BSDF. Ver Sección 2.1). Por otro lado, el término $|\mathbf{n_x} \cdot \omega_i|$[1] es un factor geométrico que pondera la contribución de la luz en función del ángulo de incidencia ω_i y la normal de la superficie $\mathbf{n_x}$.

Intuitivamente, la integral de la Ecuación 2.1 se puede entender como la suma de toda la radiancia que llega al punto $\mathbf{x}$ desde todas las direcciones de entrada ω_i y reflejada en la dirección de salida ω_o (ver Figura 2.1).

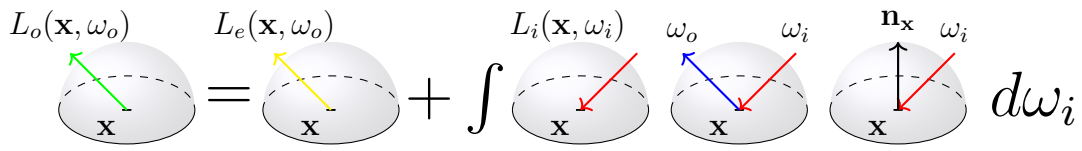


Figura 2.1: Representación gráfica de la ecuación de render. La radiancia saliente L_o (rayo verde) se calcula como la suma de la radiancia emitida (rayo amarillo) más la integral sobre todas las direcciones ω_i (rayo rojo) de la radiancia incidente L_i por la BSDF y el término geométrico.

Nótese que la Ecuación 2.1 también plantea un esquema recursivo, donde la radiancia incidente $L_i(\mathbf{x},\ \omega_i)$ a su vez se puede expresar como:

$$L_i(\mathbf{x}_t,\ \omega_i) = L_o(\mathbf{x}_{t+1},\ \omega_i) G(\mathbf{x}_t \leftrightarrow \mathbf{x}_{t+1}) V(\mathbf{x}_t \leftrightarrow \mathbf{x}_{t+1}) \tag{2.2}$$

donde $\mathbf{x}_{t+1} = \mathbf{x}_t - t\omega_i$ es un punto en otra superficie en la dirección ω_i, L_o es la radiancia saliente desde ese otro punto, la cual es atenuada por el factor geométrico $G(\mathbf{x} \leftrightarrow \mathbf{x}_t)$ y finalmente, $V(\mathbf{x} \leftrightarrow \mathbf{x}_t) \in \{0, 1\}$ es un término de visibilidad entre ambos puntos,

[1] Denotamos $|\overrightarrow{v_1} \cdot \overrightarrow{v_2}|$ al producto escalar **usual** entre los vectores $\overrightarrow{v_1}$ y $\overrightarrow{v_2}$

valiendo 1 si no hay ningún oclusor entre ellos y 0 en caso contrario. Este esquema se puede ver en la Figura 2.2.

Para obtener la imagen sintética de una escena concreta, se evalúa cuánta radiancia $L_i(\mathbf{x}_i, \omega)$ llega a una cámara virtual a través de los píxeles que forman la imagen, los cuales actúan como sensores.

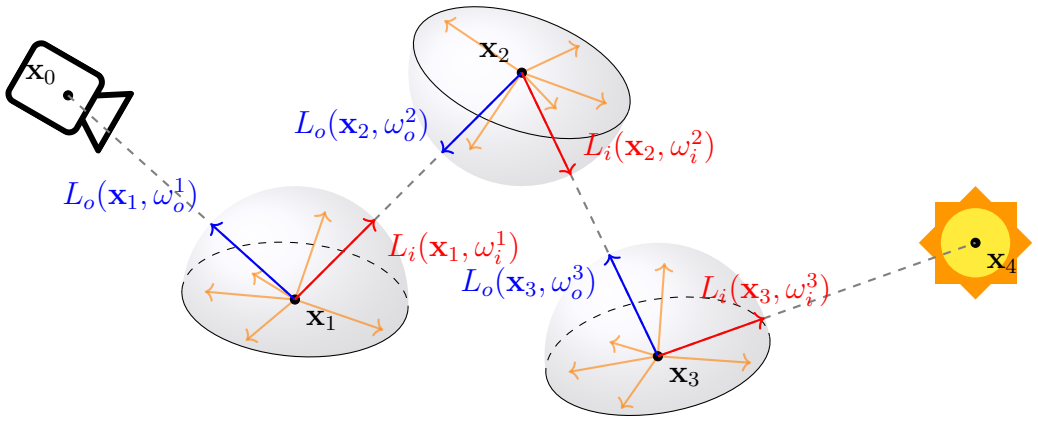


Figura 2.2: **Esquema recursivo de la ecuación de render 2.1**. Para el cálculo de la radiancia que llega a la cámara, $L_0(\mathbf{x}_1, \omega_0^1)$, se necesita el cálculo de $L_i(\mathbf{x}_1, \omega_i^1)$ que, a su vez, se puede calcular utilizando $L_i(\mathbf{x}_2, \omega_i^2)$ y así sucesivamente hasta llegar a una fuente de luz.

BSDFs

La **función de distribución de dispersión bidireccional** [14] (*bidirectional scattering distribution function*, o BSDF) describe cómo se dispersa la luz en un material en función de la dirección desde la que llega ω_i y hacia la que se propaga ω_o:

$$f_r(\mathbf{x}, \omega_o, \omega_i). \tag{2.3}$$

Las BSDFs cumplen tres propiedades esenciales [15]:

- **Positividad**: la función es no negativa en su dominio

$$f_r(\mathbf{x},\ \omega_o,\ \omega_i) \geq 0. \tag{2.4}$$

- **Propiedad de conservación de la energía**: no se puede reflejar más energía de la que llega al material

$$\int_\Omega f_r(\mathbf{x},\ \omega_o,\ \omega_i)|\mathbf{n}_\mathbf{x} \cdot \omega_i| d\omega_i \leq 1. \tag{2.5}$$

- **Propiedad de simetría**: las direcciones de entrada ω_i y de salida ω_o son intercambiables

$$\int_\Omega f_r(\mathbf{x},\ \omega_o,\ \omega_i) d\omega_i = \int_\Omega f_r(\mathbf{x},\ \omega_i,\ \omega_o) d\omega_i. \tag{2.6}$$

El valor de la BSDF define la apariencia física de los materiales. En la Figura 2.3 se exponen tres casos diferentes de BSDF y cómo esto afecta a la apariencia de los mismos.

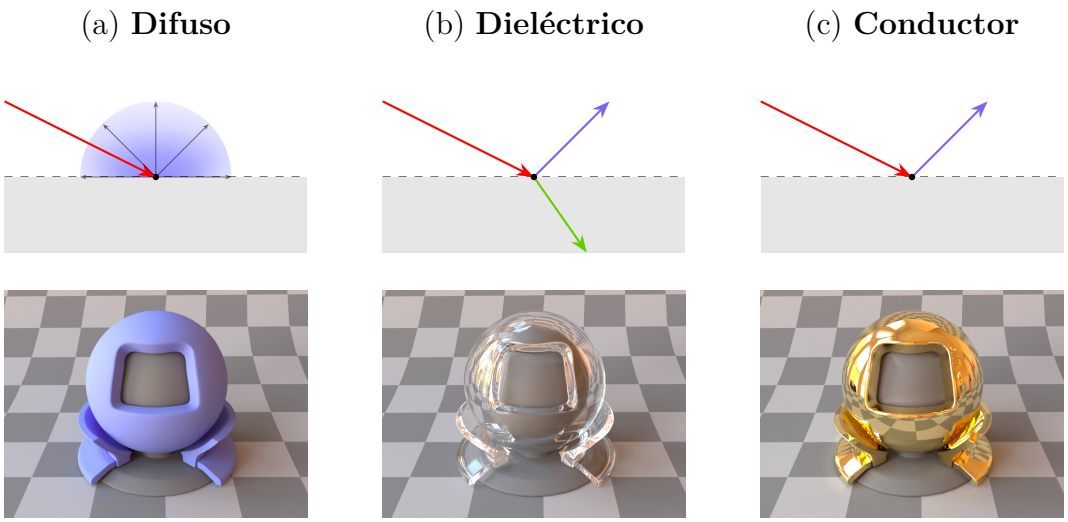


Figura 2.3: Representación gráfica de la BSDF junto a su apariencia física para tres tipos diferentes de BSDF: **(a) Difuso** que refleja en todas las direcciones por igual, **(b) plástico** (dieléctrico) que tiene una componente especular (rayo azul) y otra refractiva (rayo verde) y **(c) metal** (conductor) que solo tiene una componente especular (rayo azul). Imágenes obtenidas de la documentación de Mitsuba 3 [10].

Monte Carlo

La integral planteada en la ecuación de render (Ecuación 2.1) no tiene una solución analítica para un escenario general. Esto se debe a la complejidad derivada de la geometría y la apariencia de los materiales de la escena a renderizar. Por lo tanto, es necesario recurrir a métodos que aproximen dicha solución, siendo los más comunes por su adaptabilidad y generalidad los métodos de Monte Carlo.

Los métodos de **Monte Carlo** [16] engloban una serie de algoritmos estocásticos que aproximan esperanzas de la forma

$$\mathbb{E}_\mu\{h(X)\} = \int_\Omega h(x)\, d\mu(x)$$

donde X es una variable aleatoria continua sobre Ω con función de densidad (o pdf) p, h es una función medible y $\mu(x)$ es la medida de probabilidad definida por p. Estas esperanzas se pueden aproximar como

$$\mathbb{E}_p\{h(X)\} \approx \frac{1}{N}\sum_{i=1}^{N}\frac{h(X_i)}{p(X_i)},$$

donde $X_1, \ldots, X_N$ son muestras aleatorias de X. Aplicado a la Ecuación 2.1, si obtenemos N muestras aleatorias de direcciones $\omega_j \in \Omega$ que provienen de una distribución p, se obtiene una aproximación para la ecuación de render:

$$L_o(\mathbf{x},\ \omega_o) \approx L_e(\mathbf{x},\ \omega_o) + \frac{1}{N}\sum_{j=1}^{N} \frac{L_i(\mathbf{x},\ \omega_j) f_r(\mathbf{x},\ \omega_o,\ \omega_j)|\mathbf{n}_\mathbf{x} \cdot \omega_j|}{p(\omega_j)}. \tag{2.7}$$

Resolver la Ecuación 2.1 mediante las aproximaciones de Monte Carlo (2.7) da lugar a un algoritmo **recursivo** para el cálculo de las radiancias, conocido como trazado de rayos o *ray-tracing*. Sin restricciones, implica un crecimiento **exponencial** del número de muestras, ya que cada nueva dirección genera a su vez N nuevas ramas que deben evaluarse en distintos puntos de la escena, generando un árbol de evaluación con N^M términos para una profundidad M (ver Figura 2.2). Estas características complican significativamente su implementación desde el punto de vista de la programación.

Path Tracing

Para resolver este problema, Kayija [13] introduce el conocido algoritmo de *path tracing*. Este consiste en trazar caminos aleatorios (*random walks*) desde la cámara de la escena hasta encontrarse con una fuente de luz (ver Figura 2.4). Intuitivamente, en vez de escoger N direcciones diferentes para cada aproximación, se elige una sola dirección, eliminando el crecimiento exponencial del algoritmo presentado previamente.

Posteriormente, Veach [17] introduciría el marco teórico para el algoritmo de *path tracing*. Así surge la *path integral* que define la radiancia I_j en el píxel j como la integral de todos los caminos de luz que llegan al área relativa al píxel en la cámara:

$$I_j = \int_\Gamma f^{(j)}(\bar{\mathbf{x}}) d\mu(\bar{\mathbf{x}}), \tag{2.8}$$

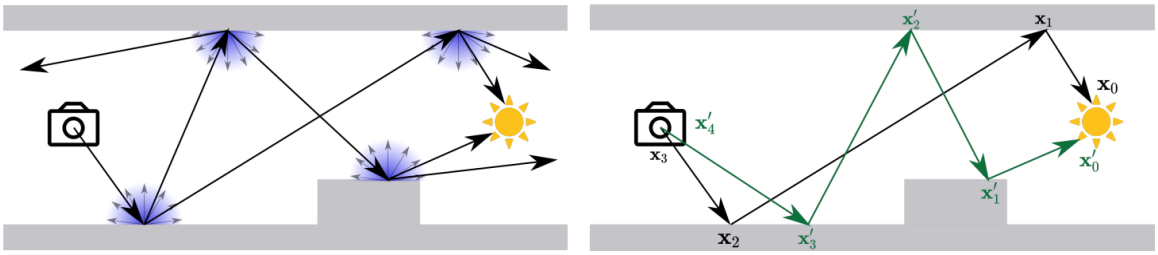


Figura 2.4: Comparación del algoritmo de trazado de rayos **(Izquierda)** y el de trazado de caminos **(Derecha)**. En el primero, por cada rayo trazado de la cámara, se muestrean diferentes direcciones, que, a su vez, muestrean otros hasta que llegan a una fuente de luz (sol). Esto supone un algoritmo de coste exponencial. En el segundo, se muestrean directamente caminos hasta una fuente de luz. Esto supone un algoritmo que depende del número de caminos trazados.

donde Γ es el conjunto de caminos $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_k)$ de longitud arbitraria, que comienzan en el sensor de la cámara $\mathbf{x}_k$ y terminan en una fuente de luz $\mathbf{x}_1$ pasando por $\mathbf{x}_j$ vértices intermedios. Finalmente, la función $f^{(j)}(\bar{\mathbf{x}})$ es la función de contribución a la medida (*measurement contribution function*) definida como

$$f^{(j)}(\overline{\mathbf{x}}) = L_e\left(\mathbf{x}_0 \to \mathbf{x}_1\right) \mathfrak{T}(\overline{\mathbf{x}}) W_e^{(j)}\left(\mathbf{x}_{k-1} \to \mathbf{x}_k\right) \tag{2.9}$$

donde $L_e\left(\mathbf{x}_0 \to \mathbf{x}_1\right)$ es la radiancia emitida por la fuente de luz, $W_e^{(j)}\left(\mathbf{x}_{k-1} \to \mathbf{x}_k\right)$ es la importancia del sensor en $\mathbf{x}_k$ y $\mathfrak{T}(\overline{\mathbf{x}})$ es el *path throughput*, que representa la pérdida de energía debido a las interacciones luz-materia en los $k-1$ vértices internos del camino $\overline{\mathbf{x}}$. A su vez, esta función está definida de la siguiente manera:

$$\mathfrak{T}(\bar{\mathbf{x}}) = \prod_{i=1}^{k-1} f_r(\mathbf{x}_{i-1} \to \mathbf{x}_i \to \mathbf{x}_{i+1}) \prod_{i=0}^{k-1} G(\mathbf{x}_i \to \mathbf{x}_{i+1}) V(\mathbf{x}_i \to \mathbf{x}_{i+1}), \tag{2.10}$$

donde $G(\mathbf{x}_i \to \mathbf{x}_{i+1})$ es el término geométrico, $V(\mathbf{x}_i \to \mathbf{x}_{i+1})$ es el término de visibilidad y f_r son las BSDFs asociadas al camino.

Al igual que la ecuación de render, la *path integral* se puede aproximar con métodos de Monte Carlo sobre N caminos muestreados aleatoriamente siguiendo una distribución p:

$$I_j \approx \frac{1}{N} \sum_{j=1}^{N} \frac{f^{(j)}(\overline{\mathbf{x}}_j)}{p(\overline{\mathbf{x}}_j)} \tag{2.11}$$

Nótese que la Ecuación 2.10 introduce intuitivamente un esquema *iterativo* para su resolución que presenta ventajas considerables desde un punto de vista computacional, ya que:

1. Un algoritmo iterativo es más **eficiente** que uno recursivo.
2. Un algoritmo iterativo es más fácilmente **paralelizable**.

2.2. Transporte de luz transitorio

En los modelos clásicos de transporte de luz utilizados en informática gráfica, se asume que la velocidad de la luz es **infinita**. Bajo esta hipótesis, la radiancia en una escena solo depende de la componente espacial, pero no del tiempo.

Esta suposición es válida en la mayoría de los contextos prácticos, debido a que la mayoría de los sensores y cámaras tradicionales son extremadamente lentos en comparación con la velocidad de la luz. Como resultado, el comportamiento dinámico de la luz queda oculto, dando la percepción de que el transporte de luz es instantáneo.

El **renderizado transitorio** (o *transient rendering*) extiende el modelo estacionario a la dimensión temporal. En este nuevo dominio, la radiancia ya no

depende solamente del espacio sino también del **tiempo de vuelo** de la luz. Además, el modelo transitorio también modela los efectos de retraso temporal inducidos por algunos materiales, como es el caso de materiales fluorescentes. Estos no reemiten la luz de manera instantánea, sino que introducen retrasos temporales siguiendo normalmente una distribución probabilística.

Jarabo *et al.* [3] presenta una formulación que adapta la teoría existente sobre el transporte de luz estacionario para incluir la nueva dimensión temporal. Así, la intensidad del píxel j se calcula como la integral sobre el espacio de caminos finitos Γ y sobre el espacio de retrasos temporales ΔT:

$$I_j = \int_{\Gamma} \int_{\Delta T} f^{(j)}(\bar{\mathbf{x}}, \overline{\Delta \mathbf{t}}) d\mu(\overline{\Delta \mathbf{t}}) d\mu(\bar{\mathbf{x}}) \tag{2.12}$$

donde $\overline{\Delta \mathbf{t}} = \Delta t_0 \dots \Delta t_k$ es el conjunto de retrasos temporales introducidos en cada vértice (ver Figura 2.5). En comparación con la fórmula presentada en la Ecuación 2.8, se ha incluido el dominio temporal en la función de contribución a la medida (ver Ecuación 2.9) $f^{(j)}(\bar{\mathbf{x}}, \overline{\Delta \mathbf{t}})$. Así, la BSDF de los materiales (ver Ecuación 2.10) depende explícitamente del retraso temporal Δt_i

$$f_r(\mathbf{x}_{i-1} \to \mathbf{x}_i \to \mathbf{x}_{i+1}, \Delta t_i).$$

La estimación mediante Monte Carlo en el dominio transitorio se basa en una extensión de la formulación estacionaria presentada en la Ecuación 2.11. En este nuevo contexto, no solo se deben muestrear caminos aleatorios $\overline{\mathbf{x}}_j$, sino también los retrasos temporales $\overline{\Delta t}_j$ asociados a cada vértice del camino. Esto da lugar a la siguiente expresión:

$$I_j \approx \frac{1}{N} \sum_{j=1}^{N} \frac{f^{(j)}(\overline{\mathbf{x}_j}, \overline{\Delta \mathbf{t}_j})}{p(\overline{\mathbf{x}}_j, \overline{\Delta \mathbf{t}_j})}. \tag{2.13}$$

donde cada muestra $(\overline{\mathbf{x}}_j, \overline{\Delta \mathbf{t}_j})$, representa un camino de la luz junto a sus retrasos en el tiempo, generados según una distribución conjunta de probabilidad p.

En este caso, cada vértice $\mathbf{x}_j$, que representa una interacción luz-material, tiene asociado un tiempo de llegada t_j^-, es decir, el instante en el que la luz alcanza ese punto antes de interactuar. Una vez se produce la interacción, se introduce el retraso Δt_j correspondiente a la reemisión de la luz. La suma de ambos tiempos da como resultado el tiempo total $t_j = t_j^- + \Delta t_j$, que indica el momento cuándo la luz vuelve a propagarse desde ese vértice.

De este modo, el tiempo total que tarda la luz en recorrer un camino completo depende de dos contribuciones distintas:

– **Retrasos de propagación**, que corresponden al tiempo que tarda la luz en desplazarse físicamente entre dos vértices consecutivos $\mathbf{x}_j$ y $\mathbf{x}_{j+1}$:

$$t(\mathbf{x}_j \longleftrightarrow \mathbf{x}_{j+1})$$

– **Retrasos de dispersión**, que son los tiempos Δt_j introducidos por los materiales en cada vértice.

La Figura 2.5 ilustra el camino y los retrasos asociados para un ejemplo simplificado de la Figura 2.2. En ella se puede observar cómo se combinan los retrasos de propagación entre puntos y los retrasos de dispersión en cada interacción para definir el tiempo total de vuelo de la luz.

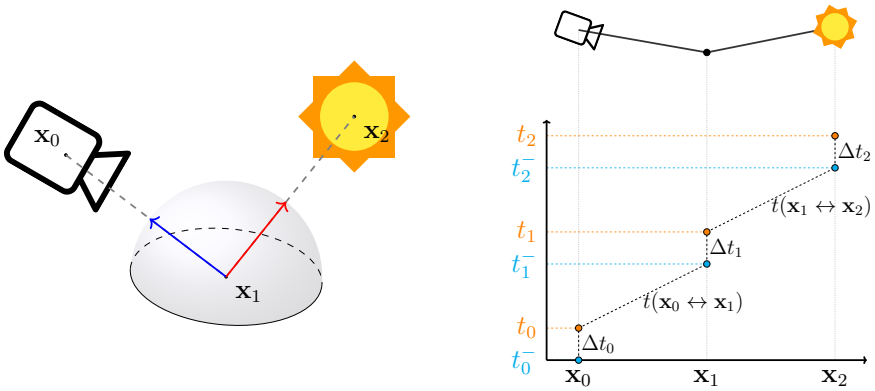


Figura 2.5: Ilustración del tiempo de vuelo de la luz para un camino de luz similar al presentado en la Figura 2.2. Aquí se representa el camino (figura izquierda) y las componentes temporales (figura derecha), donde t_i^- (azul) representa el momento en el que llega la luz y t_i (naranja) cuando la luz se re-emite tras el retraso temporal Δt_i.

La incorporación explícita del dominio temporal en los modelos de transporte de luz introduce importantes desafíos computacionales. Uno de los principales problemas es el aumento del tiempo de **convergencia** cuando se extienden algoritmos estacionarios clásicos al dominio transitorio. Esto se debe a que es difícil encontrar caminos de luz cuyo tiempo de recorrido total coincida con el intérvalo temporal que se está analizando.

2.3. Materiales con retrasos temporales

En la naturaleza, existen materiales que retrasan la emisión de la luz por distintos motivos. A pesar de esto, en la práctica, estos **retrasos de dispersión** suelen ser ignorados. Esto se debe a que, a pesar de ser fundamentales en las interacciones luz-material, la ausencia de un marco teórico sólido para modelar estos efectos ha llevado a que, en la mayoría de las simulaciones, se asuma que los retrasos son despreciables.

Desde un punto de vista físico este fenómeno puede ocurrir por muchos motivos. Uno de los más relevantes es la **dispersión inelástica**, donde la energía del fotón cambia tras interactuar con el medio, provocando además un retraso temporal antes de la reemisión, como sucede, por ejemplo, en materiales fluorescentes.

Adicionalmente, la geometría interna de un material puede introducir retrasos temporales, incluso cuando las interacciones son elásticas. La complejidad estructural del medio en estos casos, como ocurre en los **materiales multicapa** o en las **interacciones multifacéticas**, da lugar a caminos de luz complejos. En el transporte de luz estacionario, estos caminos se suelen modelar de forma simplificada, recurriendo a funciones de distribución [18, 19] que buscan simular el comportamiento de los materiales para lograr apariencias realistas (ver Figura 2.6). Sin embargo, el incremento en el tiempo de vuelo de la luz que introducen estos caminos complejos debe ser considerado explícitamente en el dominio transitorio para obtener resultados precisos.

(a) **Plástico Multicapa** (b) **Conductor Multifacetas**

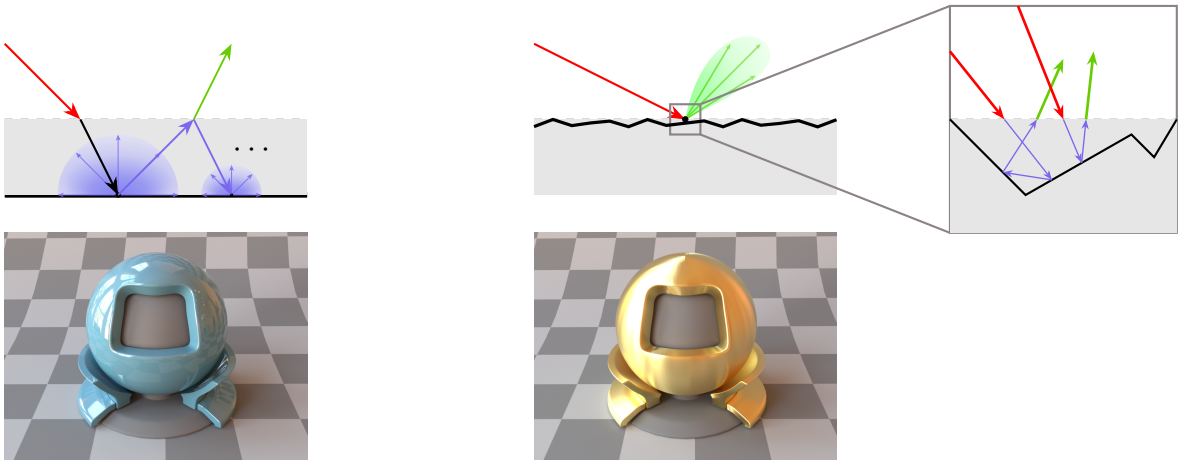


Figura 2.6: Representación gráfica de la BSDF junto a su apariencia física para **(a)** un material plástico multicapa, que tiene una capa que puede refractar o transmitir y otra capa con una componente difusa. Por otro lado, **(b)** representa un material condutor con multifacetas que le dan una apariencia más "áspera" al obtenido con un conductor básico en la Figura 2.3(c). Imágenes obtenidas de la documentación de Mitsuba 3 [10].

Hasta la fecha, los trabajos previos en el campo de los gráficos por ordenador han abordado los retrasos temporales de los materiales desde dos perspectivas distintas. Por un lado, se ha estudiado la variación de propiedades materiales a gran escala de tiempo, como la corrosión o el secado de pinturas, utilizando modelos de apariencia dinámicos [20, 21]. Estos enfoques se centran en los cambios en la apariencia estática del material a lo largo de minutos, horas o días.

Por otro lado, se ha investigado la fluorescencia, donde la luz absorbida se reemite con una longitud de onda mayor, pero generalmente sin tener en cuenta el retraso temporal asociado a este proceso. Modelos como el de Fichet et al. [22] se han centrado en la representación del cambio de color en el estado estacionario, asumiendo una reemisión instantánea. El trabajo más cercano a nuestra investigación es el de Nalbach et al. [23], que se centró en la adquisición y el modelado de materiales fosforescentes, que exhiben retrasos temporales perceptibles para el ojo humano (minutos a horas), pero no en la simulación de estos efectos en el contexto del transporte de luz transitorio a escalas de tiempo de nanosegundos o picosegundos.

Este proyecto presenta una nueva formulación matemática que integra por primera vez los retrasos temporales de la luz en el renderizado transitorio, superando las limitaciones de los modelos existentes de fluorescencia y de materiales con propiedades variables a gran escala. Nuestro enfoque es general, aplicable a diversos fenómenos de dispersión a escalas de tiempo de picosegundos o nanosegundos, lo que abre la puerta a simulaciones más precisas en campos como la microscopía avanzada [24] y el diseño de materiales nano-ópticos [25] donde los retrasos de los materiales en el transporte de luz son relevantes.

Capítulo 3

Modelado de materiales con retrasos temporales

En la naturaleza existen materiales, conocidos como **materiales transitorios** (ver Sección 2.3), que al interactuar con la luz no responden instantáneamente, sino que introducen un retraso temporal antes de la reemisión. A pesar de ser fundamentales en el dominio transitorio, la investigación sobre estos materiales ha sido escasa.

En la Sección 3.1 se comenzará introduciendo los conceptos básicos sobre estos tipos de materiales. Se seguirá con un modelo simplificado donde las interacciones espaciales y temporales son completamente separables (Sección 3.2). Finalmente, se avanza a modelos más complejos (Secciones 3.3 y 3.4) que permiten una dependencia más compleja entre el espacio y el tiempo.

3.1. Introducción a los materiales transitorios

Definimos un **material transitorio** como aquel cuya función de BSDF (ver Ecuación 2.3) depende explícitamente del retraso temporal de reemisión de la luz Δt:

$$f_r(\mathbf{x}, \omega_o, \omega_i, \Delta t).$$

Si no depende de la dimensión temporal, diremos que es un **material estacionario**.

Por comodidad, en este capítulo se utiliza la formulación usual, aunque se puede utilizar la de la *path integral* $f_r(\mathbf{x}_i \to \mathbf{x} \to \mathbf{x}_o, \Delta t)$ donde $\omega_o = \overrightarrow{\mathbf{x} - \mathbf{x}_o}$ y $\omega_i = \overrightarrow{\mathbf{x} - \mathbf{x}_i}$ (ver Sección 2.1).

Dado que la literatura existente apenas estudia cómo incorporar retrasos temporales en la reemisión de luz, este trabajo propone un nuevo marco matemático **propio** que permite definir materiales transitorios para su futura implementación. El enfoque parte de un modelo simplificado con separación espacial y temporal, y se generaliza hacia modelos más complejos.

3.2. Materiales Transitorios Separables

En la naturaleza, la mayoría de los materiales transitorios presentan retrasos temporales que se rigen por una función de distribución independiente del punto de intersección $\mathbf{x}$ y de las direcciones de entrada y salida ω_i, ω_o. Es decir, la parte espacial de esta función es totalmente independiente de la parte temporal.

Formalmente, decimos que un material tiene un perfil **transitorio separable** si su BSDF se puede expresar de la siguiente forma:

$$f_r(\mathbf{x}, \omega_o, \omega_i, \Delta t) = f_r^S(\mathbf{x}, \omega_o, \omega_i) \cdot f_r^T(\Delta t), \tag{3.1}$$

donde f_r^S es una BSDF estacionaria que modela la apariencia espacial del material y f_r^T es la componente puramente temporal que define cómo son los retrasos temporales.

Naturalmente, surge la idea de definir la componente temporal $f_r^T(\Delta t)$ a través de una función de distribución con soporte positivo (ya que los retrasos lo son):

$$p : \Delta T \subset \mathbb{R}^{\geq 0} \longrightarrow \mathbb{R}.$$

Así, se puede demostrar (ver Teorema 2 del Apéndice) que si la componente temporal se rige por una función de distribución, $f_r^T(\Delta t) = p(\Delta t)$, entonces f_r es una BSDF válida, es decir, cumple las tres propiedades presentadas en las Ecuaciones 2.4 a 2.6.

Esto permite definir de forma sencilla un material transitorio a partir de un **material estacionario** (como pueden ser materiales difusos, conductores, ...) y una distribución de probabilidad que defina el **perfil temporal** del material.

Este tipo de materiales permiten simplificar bastante los modelos ya que, al realizar la aproximación por Monte Carlo, se puede muestrear por separado el retraso temporal Δt_i y la dirección saliente ω_i. Esto se debe a que, bajo estas suposiciones, estas variables aleatorias son independientes; luego, la probabilidad conjunta $p(\omega_i, \Delta t)$ se puede expresar como:

$$p(\omega_i, \Delta t) = p(\omega_i) \cdot p(\Delta t) \tag{3.2}$$

donde $p(\omega_i)$ es la densidad de probabilidad de muestreo de la dirección ω_i asociada a la BSDF estática.

Para obtener muestras de la densidad $p(\Delta t)$, si se conoce una fórmula cerrada para la inversa de la **función de probabilidad cumulativa** (CDF) de la distribución escogida, $F_p^{-1} : [0, 1] \to \Delta T$, se pueden obtener muestras aleatorias siguiendo el esquema algorítmico presentado a continuación. Este algoritmo es conocido como el **Método de Inversión** [26] y su validez matemática se puede ver en el Apéndice 1:

Algorithm 1: Muestreo por CDF Inversa

Data: $F_p^{-1} : [0,1] \to \Delta T$ (función de probabilidad cumulativa inversa)
Result: $\Delta t \in \Delta T$ (muestra aleatoria con densidad $p(x)$)
`// Método de transformación inversa para muestreo aleatorio`
1 Generar $\xi \sim U[0,1]$ (número aleatorio uniforme)
2 $\Delta t = F_p^{-1}(\xi)$
3 **return** Δt

3.2.1. Perfiles Temporales

En el presente trabajo se han estudiado tres **perfiles temporales** (distribuciones de probabilidad): **Constante**, **Exponencial** y **Epanechnikov**. Estos tres perfiles son, bajo ciertas restricciones, distribuciones con soporte positivo que permiten definir materiales transitorios separables.

En primer lugar, el **perfil constante**, Cte(c), permite definir materiales con un retraso constante, es decir, que reemiten la luz después de un tiempo c. Este comportamiento se observa en materiales como los fluorescentes.

En segundo lugar, el **perfil exponencial**, Exp(λ), es muy común cuando se analizan ciertos materiales. Por ejemplo, Datta *et al.* [24] analizan los diferentes perfiles exponenciales presentes en una célula para así diferenciar las diferentes partes que la componen. El parámetro λ controla la rapidez con la que la luz se reemite: valores grandes concentran la reemisión en tiempos cortos, mientras que valores pequeños la extienden en el tiempo.

Finalmente, el **perfil Epanechnikov**, Epan(μ, σ), surge como una adaptación con soporte positivo de la distribución gaussiana, una de las distribuciones más presentes en la naturaleza. Este perfil puede emplearse para aproximar los perfiles temporales de materiales complejos. Los hiperparámetros (μ, σ) controlan la media y la desviación, respectivamente.

Las propiedades básicas de estos perfiles se pueden ver de forma resumida en la Tabla 3.1. También se expone la función inversa de la CDF utilizada para obtener muestras aleatorias de estos perfiles.

Para validar el método de inversión para obtener muestras de estos tres perfiles temporales, en la Figura 3.1 se expone una gráfica con la función de densidad junto al histograma obtenido tras generar $n = 10^5$ muestras utilizando su CDF^{-1}. En el límite $n \to \infty$, este histograma debería ajustarse perfectamente a la función de densidad si el método es correcto.

	Constante	Exponencial	Epanechnikov
Notación	$\mathrm{Cte}(c)$	$\mathrm{Exp}(\lambda)$	$\mathrm{Epan}(\mu, \sigma)$
Restricciones	$c \geq 0$	$\lambda > 0$	$\mu > 0,\ \sigma > 0,\ \mu \geq \sigma$
Soporte	$x \in \{c\}$	$x > 0$	$x \in (\mu - \sigma,\ \mu + \sigma)$
Densidad	$p(x) = \delta_c(x)$	$p(x) = \lambda e^{-\lambda x}$	$p(x) = \frac{3}{4}\left(1 - \left(\frac{x-\mu}{\sigma}\right)^2\right)$
CDF^{-1}	$F^{-1}(z) = c$	$F^{-1}(z) = -\frac{1}{\lambda}\ln(1-z)$	$F^{-1}(z) = 2\sigma \cdot \sin\left(\frac{1}{3}\arcsin(2z-1)\right) + \mu$

Tabla 3.1: Perfiles estudiados y sus propiedades: notación utilizada durante la memoria, restricciones de hiperparámetros, soporte, densidad de probabilidad y función inversa de la CDF.

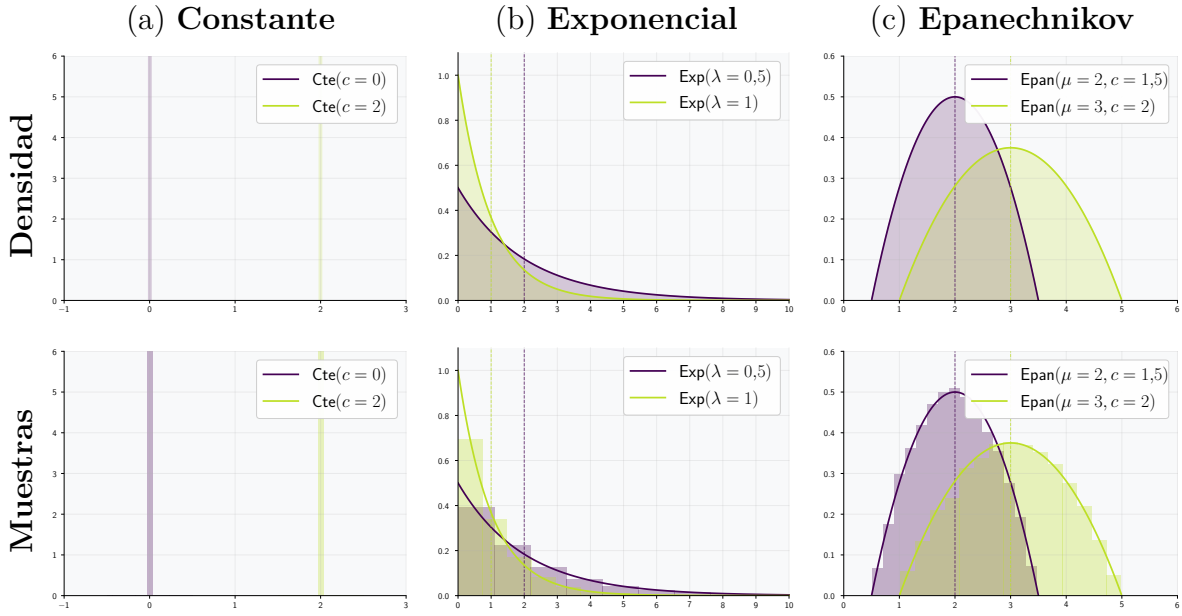


Figura 3.1: Validación del Método de Inversión para los tres perfiles implementados: Constante, Exponencial y Epanechnikov. En la fila "Densidad" se muestran las funciones de densidad $p(x)$ y en la fila "Histograma" los histogramas obtenidos para $n = 10^5$ muestras.

3.3. Materiales Transitorios Separables Modulados Espacialmente

A pesar de que los materiales transitorios separables son suficientes para gran parte de los materiales presentes en la naturaleza, la suposición de total separación puede ser muy restrictiva para muchos otros, donde el retraso temporal depende explícitamente del punto espacial donde ocurre la interacción $\mathbf{x}$.

Así surge el concepto de **material transitorio separable modulado espacialmente**, que, matemáticamente, se puede definir como un material cuya BSDF

es de la forma:

$$f_r(\mathbf{x}, \omega_o, \omega_i, \Delta t) = f_r^S(\mathbf{x}, \omega_o, \omega_i) \cdot f_r^T(\Delta t, \mathbf{x}), \tag{3.3}$$

donde f_r^S es una BSDF estacionaria que modela la apariencia del material, y f_r^T es la componente temporal que depende también del punto de intersección $\mathbf{x}$. La dificultad de estos modelos radica en cómo definir $f_r^T(\Delta t, \mathbf{x})$ para que sea una BSDF válida.

Supongamos que tenemos una distribución temporal base $p(\Delta t \mid \theta)$, donde θ son los hiperparámetros de la distribución. Un ejemplo sería la distribución exponencial que depende de $\theta = \{\lambda\}$. Ahora supongamos que, para cada punto de una superficie $\mathbf{x} \in \Omega$, se tienen unos hiperparámetros concretos $\theta = \Theta(\mathbf{x})$. Así, podemos modelar la componente temporal utilizando esta distribución modulada espacialmente.

$$f_r^T(\Delta t, \mathbf{x}) = p(\Delta t \mid \theta) = p(\Delta t \mid \Theta(\mathbf{x})). \tag{3.4}$$

Se puede demostrar (ver Teorema 3 del Apéndice) que así se obtiene una BSDF transitoria f_r válida.

Para definir esta función $\Theta(\mathbf{x})$ que asigna a cada punto de la superficie $\mathbf{x}$ un valor de los hiperparámetros θ, se puede recurrir a **imágenes de texturas 2D**. Aquí, a cada punto de la geometría $\mathbf{x}$ le corresponde un punto concreto en la imagen de texturas y, por ende, un color concreto. Así, este color puede definir el valor de los hiperparámetros de la distribución.

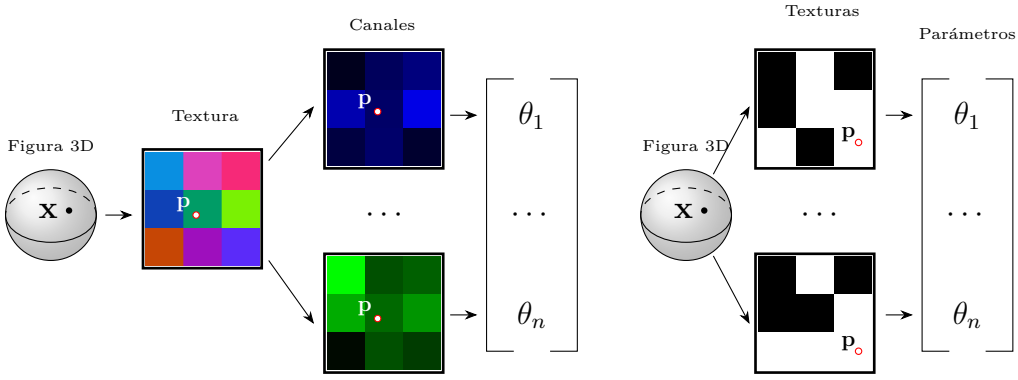
Esto puede hacerse de forma individual, asignando a cada hiperparámetro una imagen de textura de un solo canal (en escala de grises, ver Figura 3.2b), donde la intensidad define el valor del parámetro: valores más oscuros corresponden a valores menores del hiperparámetro. Otra alternativa consiste en utilizar una imagen con múltiples canales de color para definir todos los hiperparámetros, donde el valor de cada canal representa el valor de un hiperparámetro diferente (ver Figura 3.2a).

Intuitivamente, al utilizar texturas, lo que realmente está ocurriendo es que, como el número de píxeles de una imagen es finito, la superficie 3D queda dividida en pequeños parches con distintos valores de hiperparámetros.

La formulación desde un punto más matemático de cómo se obtiene una correspondencia entre una imagen 2D y una superficie 3D para la implementación de texturas se puede ver en el Apéndice A.1.

En cuanto a cómo aplicar Monte Carlo en este tipo de materiales, el muestreo temporal se puede realizar también de forma independiente, simplemente que ahora se necesita saber a priori el punto de intersección $\mathbf{x}$ y por ende los hiperparámetros de la distribución de la que se van a obtener muestras. Consecuentemente, la densidad de probabilidad dado el punto $\mathbf{x}$ es ahora:

$$p(\omega_i, \Delta t \mid \mathbf{x}) = p(\omega_i) \cdot p(\Delta t \mid \Theta(\mathbf{x})) \tag{3.5}$$



(a) (b)

Figura 3.2: Obtención de hiperparámetros para cada punto $\mathbf{x}$ utilizando texturas. **(a)** Todos los hiperparámetros vienen dados por una única textura donde el valor de cada canal representa el valor de un hiperparámetro. **(b)** Cada parámetro tiene asociado una textura concreta de un canal único, luego su valor se obtiene directamente del valor de la textura.

3.4. Composición Materiales Transitorios

Finalmente, se van a estudiar un conjunto de materiales que no se pueden separar completamente en una parte espacial y temporal como, por ejemplo, los materiales fluorescentes.

Intuitivamente, se va a crear una BSDF transitoria como una composición de dos o más BSDF transitorias separables. Esto nos permite definir materiales que, por ejemplo, emiten un color en el instante t_0 y otro diferente en el instante t_1. Matemáticamente, si se parte de n materiales transitorios separables con BSDFs:

$$f_k(\mathbf{x}, \omega_o, \omega_i, \Delta t) = f_k^S(\mathbf{x}, \omega_i, \omega_o) \cdot p_k(\Delta t \mid \Theta_k(\mathbf{x})) \qquad \forall k = 1, \ldots, n$$

y $\{c_k\}_{k=1}^n$ pesos (o probabilidad) positivos cuya suma es igual a 1, entonces podemos definir un nuevo material como la suma ponderada de estos materiales, resultando en una BSDF válida (ver Teorema 4 del Apéndice):

$$f_r(\mathbf{x}, \omega_o, \omega_i, \Delta t) = \sum_{k=1}^{n} c_k \cdot f_k(\mathbf{x}, \omega_o, \omega_i, \Delta t) = \sum_{k=1}^{n} c_k \cdot f_k^S(\mathbf{x}, \omega_o, \omega_i) \cdot p_k(\Delta t \mid \Theta_k(\mathbf{x})). \quad (3.6)$$

Para obtener muestras de esta BSDF, se elige con una probabilidad c_k qué BSDF transitoria se va a muestrear. Una vez seleccionada, se muestrea la densidad tal y como se ha explicado con anterioridad. De forma algorítmica:

Algorithm 2: Muestreo de Composición de Materiales Transitorios

Data: $\mathbf{x}$ (posición), $\xi \sim U[0,1]$ (número aleatorio uniforme)
Result: ω_i (dirección), Δt (tiempo)
`// Seleccionar módulo basado en probabilidades acumulativas`
$k \leftarrow j \mid \left(\sum_{i=1}^{j-1} c_i \leq \xi < \sum_{i=1}^{j} c_i\right)$;
`// Obtener parámetros del módulo seleccionado en la posición`
$\theta \leftarrow \Theta_k(\mathbf{x})$;
`// Samplear el módulo seleccionado`
$(\omega_i, \Delta t) \leftarrow$ Samplear de $p_k(\omega_i, \Delta t \mid \theta)$;
return $(\omega_i, \Delta t)$;

Además, la función de densidad en este caso se puede expresar como:

$$p(\omega_i, \Delta t \mid \mathbf{x}) = \sum_{k=1}^{n} c_k \cdot p_k(\omega_i, \Delta t \mid \mathbf{x}) = \sum_{k=1}^{n} c_k \cdot p_k^S(\omega_i) \cdot p_k(\Delta t \mid \mathbf{x}) \tag{3.7}$$

donde p_k^S es la densidad de probabilidad de muestreo de ω_i asociada a cada BSDF estática f_k^S.

En el caso de que todas las densidades de probabilidad $p_k^S(\omega_i)$ sean iguales, por ejemplo, que todos los materiales sean difusos, esta composición se puede entender intuitivamente como un único material difuso, con un perfil temporal calculado como la suma ponderada de los perfiles temporales individuales de cada material (ver Figura 3.3).

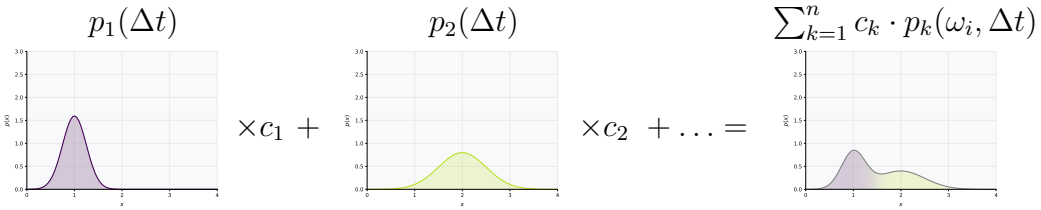


Figura 3.3: Ejemplo de composición de perfiles temporales para materiales con el mismo perfil espacial. El perfil resultante se obtiene como una composición lineal de las componentes.

Un ejemplo básico de este tipo de materiales es la **clorofila**. A nivel físico, este 'material' es un tipo de fluorescente que reemite en el instante $t = 0$ un pulso de color verde y, en un instante posterior $t_0 > 0$, un pulso de color rojo. Este material se puede modelar como la composición de dos ($k = 2$) materiales transitorios separables. Aquí, ambos tienen una BSDF estacionaria difusa, una con color verde y otra con color rojo. Respecto a los perfiles temporales, el verde tiene un perfil constante con $c = 0$ y el rojo tiene un perfil constante con $c = t_0$. Además, vienen modulados por un peso $c_v \in [0,1]$ ($c_1 = c_v$ y $c_2 = 1 - c_v$) que representa la proporción de verde del material.

Capítulo 4

Mitsuba 3 y Mitransient

Este capítulo presenta los conceptos básicos de Mitsuba 3 (Sección 4.1) y Mitransient (Sección 4.2), dos herramientas clave utilizadas para la implementación de los materiales transitorios.

En concreto, se introducen únicamente los aspectos básicos que serán necesarios para entender las diferentes decisiones de diseño tomadas para la implementación de estos materiales (Capítulo 5).

4.1. Mitsuba 3

Mitsuba 3 [10] es un motor de renderizado físico, diferenciable, multiplataforma y de código abierto, diseñado específicamente para la investigación académica.

La arquitectura de este motor de renderizado consiste en una serie de componentes interconectadas que realizan distintas funciones para obtener finalmente una imagen por ordenador. Estas se implementan a través de clases abstractas, entre las que destacan:

- Sensores (`mitsuba.Sensor`): encargados, junto a las películas (`mitsuba.Film`), de procesar y almacenar las radiancias que llegan a cada uno de los píxeles de la imagen.
- Emisores (`mitsuba.Emitters`): representan las fuentes de luz de una escena.
- Geometrías (`mitsuba.Shape`): definen las figuras 3D de las escenas.
- BSDFs (`mitsuba.BSDF`): definen la apariencia de las geometrías.
- Integradores (`mitsuba.Integrator`): componentes que realizan las técnicas de renderizado y calculan las aproximaciones de las integrales involucradas para resolver la ecuación de render (ver Ecuación 2.1)

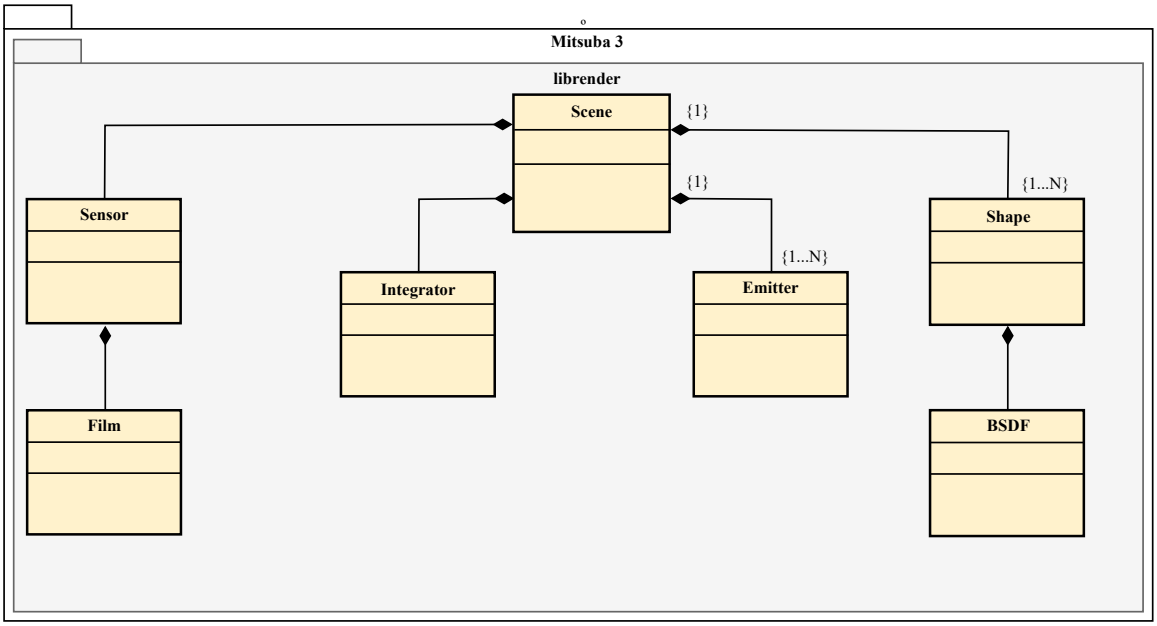


Figura 4.1: Diagrama de Clases simplificado del modelo de Mitsuba 3 que contiene los elementos básicos de un renderizador físico.

El diagrama de clases simplificado de Mitsuba 3 se puede ver en la Figura 4.1.

La principal característica de Mitsuba 3 es que soporta varios **modos de ejecución**, permitiendo elegir distintos backends según el hardware disponible. Entre estos modos se incluye la ejecución **escalar**, que realiza los cálculos instrucción por instrucción, **vectorizada** en **CPU** mediante `LLVM` y ejecución **paralela** en **GPU** usando `CUDA`.

Mitsuba 3 está implementado en `C++17` y utiliza un sistema de compilación basado en `CMake`. Su arquitectura modular y orientada a plugins permite extender el motor fácilmente mediante nuevas clases en `C++`, como integradores, materiales o sensores personalizados. Además, cuenta con una integración nativa con `Python` que no solo permite ejecutar renderizados de forma eficiente, sino también definir nuevos plugins directamente en `Python`, que pueden utilizarse como si fueran componentes nativos del sistema. Este enfoque híbrido facilita el desarrollo tanto a bajo nivel (`C++`) como a alto nivel (`Python`).

4.1.1. Materiales en Mitsuba 3

En Mitsuba 3, los materiales se modelan utilizando la clase `mitsuba.BSDF`. Esta clase incluye varios métodos importantes, necesarios para la evaluación de la Ecuación 2.7, entre los cuales destacan:

– `BSDF.eval`: evalúa la BSDF en un punto específico y la multiplica por el factor geométrico:

$$f_r(\mathbf{x}, \omega_o, \omega_j)|\mathbf{n}_\mathbf{x} \cdot \omega_j|$$

– `BSDF.pdf`: evalúa la probabilidad por ángulo sólido de muestreo de la dirección ω_j, es decir:

$$p(\omega_j).$$

– `BSDF.sample`: obtiene una muestra para una nueva dirección ω_i según la densidad (pdf) del material.

A partir de esta clase, heredan cada una de las BSDF implementadas en Mitsuba 3.

Además, Mitsuba 3 ofrece un plugin específico que permite definir BSDFs directamente desde `Python`, posibilitando redefinir los métodos con implementaciones personalizadas para diferentes materiales.

4.1.2. Texturas en Mitsuba3

En Mitsuba 3, las texturas se modelan a través de la clase `mitsuba.Texture`. Uno de sus métodos más importantes es `eval_1`, que devuelve el primer (y único) canal de color de la textura asociado a un punto de intersección $\mathbf{x}$ (ver Figura 3.2(b)). Este método se utilizará más adelante para implementar los materiales transitorios modulados espacialmente (ver Sección 3.3)

De manera similar a los materiales, Mitsuba 3 permite crear clases personalizadas heredando de esta clase `mitsuba.Texture` mediante plugins en `Python`.

4.2. Mitransient

Mitransient [11] es una librería multiplataforma de `Python` que extiende la funcionalidad de Mitsuba 3 al dominio transitorio.

Sin embargo, esta librería tiene una limitación: supone que no hay retrasos temporales causados por materiales, es decir, $\Delta\mathbf{t} = \mathbf{0}$. Por lo tanto, solo se capturan los retrasos de propagación debidos al tiempo de vuelo de la luz, no permitiendo así la introducción de materiales transitorios.

Además, Mitransient no trabaja directamente en unidades de tiempo físico, ya que la velocidad de la luz es extremadamente alta, lo que implicaría manejar escalas temporales del orden de picosegundos o menores, pudiendo producirse errores de tipo *underflow*. En su lugar, utiliza la **distancia óptica** (*optical path length*, OPL), que representa la distancia efectiva recorrida por la luz en función del medio. Formalmente, la distancia óptica de un camino concreto $\bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_k)$ viene dada por la siguiente fórmula:

$$\mathrm{OPL}(\bar{\mathbf{x}}) = \sum_{i=1}^{k-1} d(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \cdot \nu_i \tag{4.1}$$

donde $d(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1})$ es la distancia física entre los dos vértices y ν_i es el índice de refracción del medio entre los vértices $\mathbf{x}_i$ y $\mathbf{x}_{i+1}$.

Nótese que esta medida se mide en **metros** (m) y se puede convertir a unidades de tiempo (segundos o s) con la siguiente fórmula:

$$t(\bar{\mathbf{x}}) = \sum_{i=1}^{k-1} \frac{d(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1})}{c/\nu_i} = \frac{1}{c}\mathrm{OPL}(\bar{\mathbf{x}}) \tag{4.2}$$

Por ejemplo, si tenemos un camino con distancia óptica $\mathrm{OPL}(\bar{\mathbf{x}}) = 1$ m, esto correspondería a un tiempo de $t(\bar{\mathbf{x}}) = 33{,}36$ ns.

4.2.1. Integrador Transitorio

La principal contribución de Mitransient es la implementación de un integrador transitorio que permite calcular la radiancia que llega a cada píxel de la imagen en cada instante temporal t. Para ello, se sigue el algoritmo iterativo (*path tracing*) planteado en la Ecuación 2.10 y cuyo pseudocódigo se puede ver a continuación:

Algorithm 3: Calcular Aportación camino

Data: Rayo inicial r del pixel (x, y), escena S
Result: Radiancia L y tiempo total t

$T \leftarrow 1$ `// Inicializar throughput (ver 2.10)`
while *verdadero* **do**
  $\mathbf{x} \leftarrow S$.intersectar(r) `// Calcular intersección con la escena`
  $t \leftarrow t + \mathbf{x}$.OPL($r$) `// Sumar tiempo total transcurrido`
  `// Comprobar si el objeto intersectado es una fuente de luz`
  if $\mathbf{x}$.*es_emisor*() **then**
    $L \leftarrow T \cdot \mathbf{x}$.luz_emitida()
    break
  end
  `// Samplear nueva dirección y obtener contrubución de la BSDF.`
  $(r_{out}, f) \leftarrow \mathbf{x}$.sample($r$)
  $T \leftarrow T \cdot f \quad r \leftarrow r_{out}$ `// Actualizar variables`
end
return (L, t)

La principal diferencia con un algoritmo de *path tracing* estacionario (Apéndice B.1) es que, por cada interacción $\mathbf{x}_j \leftrightarrow \mathbf{x}_{j+1}$ (definidas por $(\mathbf{x}, r)$ en el código), se obtiene la distancia óptica que ha recorrido el rayo de luz ($t(\mathbf{x}_j \leftrightarrow \mathbf{x}_{j+1}) \sim \mathbf{x}.\mathrm{OPL}(r)$) con la que se actualiza el tiempo total del camino t. Así se obtienen tanto la radiancia total L como el tiempo total t del camino generado hasta llegar a una fuente de luz.

Capítulo 5

Implementación de materiales transitorios

En este capítulo se describe el proceso de implementación de materiales transitorios (Capítulo 3) utilizando el motor de renderizado Mitsuba 3 y la librería Mitransient (Capítulo 4).

En la Sección 5.1 se explica cómo se crearon las clases necesarias para definir los **materiales transitorios separables** y los perfiles temporales (ver 3.2). A continuación, en la Sección 5.2 se presenta la implementación de las texturas para definir los parámetros de los perfiles temporales, algo indispensable para la definición de **materiales separables modulados espacialmente** (ver Sección 3.3). Finalmente, en la Sección 5.3 se explica cómo se implementó la **composición de materiales transitorios** (ver Sección 3.4). Una vez completadas estas implementaciones, se presentan en la Sección 5.4 los cambios realizados en el integrador transitorio de Mitransient (ver Sección 4.2.1).

Desde el inicio de la implementación se definieron dos objetivos clave. Primero, garantizar que los cambios introducidos no alteraran el funcionamiento de las escenas, materiales u otros componentes existentes de Mitsuba. Segundo, asegurar que el sistema fuera compatible con los modos de ejecución vectorial y paralelo (ver Sección 4.2), ya que el modo escalar no es viable en contextos transitorios debido a su baja velocidad de convergencia.

En la Figura 5.1 se presenta el diagrama de clases resultante tras la implementación de los materiales transitorios.

5.1. Materiales separables

En esta sección se detallan las decisiones de diseño adoptadas para implementar los materiales separables descritos en la Sección 3.2. Para ello, fue necesario definir dos

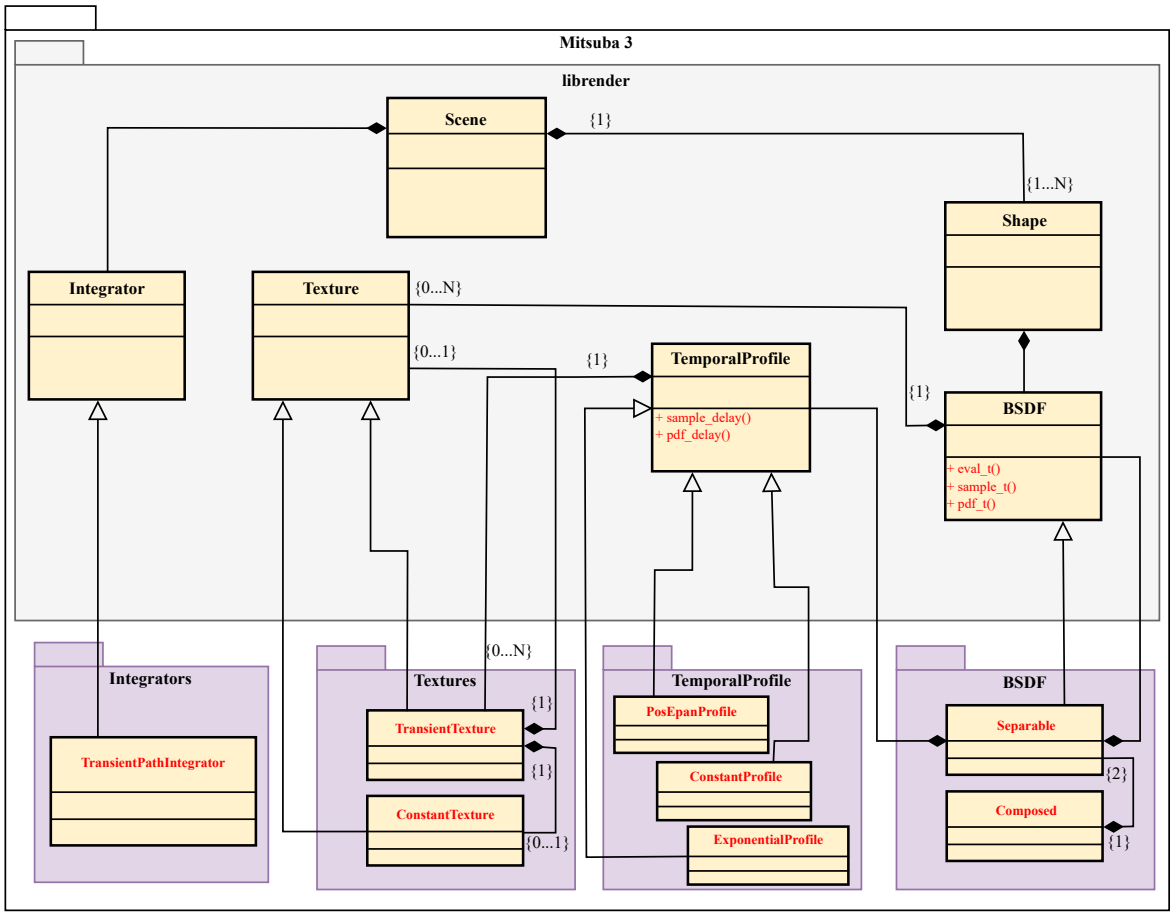


Figura 5.1: Diagrama de Clases simplificado del modelo de Mitsuba 3 tras la adición de las diferentes clases necesarias para la implementación de los materiales con retrasos temporales.

clases: una para representar los materiales transitorios separables y otra para definir sus perfiles temporales.

Como se explicó en el Capítulo 4, Mitsuba 3 permite implementar plugins (clases) tanto en `C++` como en `Python`. Aunque inicialmente se exploraron alternativas para trabajar únicamente con `Python`, lo cierto es que este enfoque presenta limitaciones importantes en cuanto a las posibilidades de implementación. En cambio, `C++` permite implementar cualquier funcionalidad, con el inconveniente de que el usuario debe compilar manualmente el sistema.

5.1.1. Perfiles Temporales

Se comenzó implementando los perfiles temporales (ver Sección 3.2.1) que modelan las distintas formas de reemisión de la luz de los materiales. Para ello, era necesario definir una interfaz `TemporalProfile` que incluyera dos métodos fundamentales:

1. `sample_delay`: devuelve un retraso temporal aleatorio Δt.

2. `pdf_delay`: evalúa la densidad de probabilidad para un retraso concreto $p(\Delta t)$.

Se consideraron dos alternativas para implementar esta interfaz: hacerlo en `Python` o directamente en `C++`. La implementación en `Python` sería más simple, requiriendo solo definir la interfaz `TemporalProfile` y sus distintas implementaciones, mientras que la implementación en `C++` implicaba también desarrollar los bindings necesarios para integrarla con `Python`.

Sin embargo, la implementación en `C++` permitía definir un perfil temporal como una clase nativa de Mitsuba, lo que facilitaba la integración con el resto de componentes del motor. Esto ofrecía varios beneficios entre los que destacan: poder trabajar con perfiles temporales directamente en `C++` y contar con una definición de materiales más intuitiva para el usuario, como se muestra en el Apéndice B.1.1.

Tras valorar las ventajas y desventajas de ambas alternativas, se decidió finalmente una implementación en `C++` para no romper con la doble implementación `Python-C++` propia de Mitsuba.

Una vez definida la interfaz `TemporalProfile` en `C++` se crearon los perfiles temporales: `ConstantProfile`, `ExponentialProfile` y `PosEpanProfile` que implementaban los perfiles Constante, Exponencial y Epanechnikov respectivamente, tal y como se describió en la Sección 3.2.

5.1.2. BSDF Separable

Una vez implementados los perfiles temporales, se creó una nueva clase llamada `Separable`, que hereda de `mitsuba.BSDF` (ver Sección 4.1.1) para definir los materiales separables. Esta nueva clase tiene dos componentes:

- Una **BSDF estacionaria** (`mitsuba.BSDF`), que representa la apariencia del material.

- Un **perfil temporal** (`TemporalProfile`), encargado de modelar los retrasos temporales.

Así, en esta clase `Separable`, los métodos heredados de `mitsuba.BSDF` (ver Sección 4.1.1), como `eval`, `pdf` y `sample`, se implementan delegando su funcionalidad a la BSDF estacionaria que la compone, mientras que el perfil temporal se encarga de manejar todo lo relacionado con los retrasos temporales.

Sin embargo, añadir los retrasos temporales a la clase `Separable` no fue sencillo ya que al heredar de la clase `mitsuba.BSDF`, solo se podían utilizar sus métodos. Esto planteó una serie de problemas ya que se necesitaba un método que devolviese los retrasos temporales Δt obtenidos del perfil temporal, algo que no existía. Este método necesitaba un parámetro $\xi \sim U[0, 1]$ para poder obtener muestras aleatorias por la CDF^{-1} (ver Algoritmo 1). Además, debía devolver un valor flotante Δt que correspondiese al retraso temporal generado.

En una primera instancia, se intentó aprovechar los métodos ya existentes en la clase `BSDF` para así evitar modificar el código en `C++`. En concreto, se planteó utilizar el método `eval_attribute_1` cuyo propósito no era generar retrasos temporales aleatorios, sino evaluar una propiedad escalar del material (por ejemplo, el coeficiente de refracción del mismo). Sin embargo, debido a que devolvía un valor flotante, parecía adecuado. Además, aunque esta función no tenía el parámetro aleatorio ξ, se añadió una componente `mi.Sampler` a la clase `Separable` para generar muestras aleatorias dentro de la misma. A pesar de conseguir en un principio utilizar este método para generar retrasos temporales **constantes**, surgieron varias limitaciones:

1. Los materiales que usaran `eval_attribute_1` para otras operaciones no podían implementarse, ya que este método estaba dedicado a la devolución de retrasos temporales.

2. Aun era necesario calcular la probabilidad $p(\overline{\Delta \mathbf{t}})$ (ver Ecuación 3.2), lo cual requería un método adicional específico para su evaluación.

3. No era posible generar muestras aleatorias independientes utilizando el `mi.Sampler` interno ya que Mitsuba crea un único *sampler* por renderizado y ese es el que debe usarse en todo el proceso.

Esta implementación rompía completamente con las directrices generales planteadas, por lo que se buscó otra alternativa. Para ello, se intentó crear un nuevo método en `Separable` para generar los retrasos, invocándolo únicamente cuando los materiales fueran transitorios. Sin embargo, debido a las restricciones de herencia de `C++`, solo se podían usar los métodos de `mitsuba.BSDF` aunque el código estuviese escrito en Python, donde sí está permitido. Además, realizar *casts* a la clase transitoria interrumpiría la ejecución vectorial y paralela.

Debido a que no se pudieron adaptar completamente los métodos ya existentes a los requerimientos, la solución finalmente adoptada fue modificar directamente la clase `mitsuba.BSDF` desde `C++` para adaptarla al dominio transitorio, añadiendo los métodos necesarios para realizar las aproximaciones de Monte Carlo (Ecuación 2.13).

Para garantizar la compatibilidad con las BSDF ya existentes, en vez de modificar directamente los métodos de la clase `mitsuba.BSDF` presentados en la Sección 4.1.1, se optó por crear la versión transitoria de estos métodos, que, por defecto, devolvieran el valor sin considerar el dominio temporal. Estos son:

- `eval_t`: recibe como parámetro un retraso Δt y devuelve la evaluación de la BSDF temporal:
$$f_r(\mathbf{x}, \omega_o, \omega_i, \Delta t)$$
- `pdf_t`: también recibe un retraso Δt como parámetro y evalúa la densidad en este instante:
$$p(\omega_i, \Delta t) \tag{5.1}$$
- `sample_t`: además de devolver la nueva dirección ω_j también devuelve el retraso temporal asociado Δt_j.

De esta manera, la clase `Separable` puede heredar de `mitsuba.BSDF` e implementar los materiales transitorios separables.

Como ya se ha comentado, esta clase `Separable` está formada por una BSDF estática y un `TemporalProfile` o `TP` para abreviar (ver Sección 5.1.1), donde la primera modela la apariencia física del material y la segunda la parte temporal. Así, la implementación de estos tres métodos para la clase `Separable` es inmediata: para obtener muestras (`sample_t`), se obtienen por **separado** muestras de la BSDF estática (`BSDF.sample`) y del retraso temporal (`TP.sample_delay`), mientras que para

la evaluación de la BSDF (`eval_t`) y de la probabilidad (`pdf_t`), simplemente se escala el valor devuelto por la BSDF estática (`BSDF.eval` y `BSDF.pdf`) por la densidad del retraso (`TP.pdf_delay`). Esto último es necesario según la teoría expuesta en el Capítulo 3, que se puede ver de forma resumida en la Tabla 5.1.

		BSDF		Perfil Temporal
$f_r(\mathbf{x}, \omega_o, \omega_i, \Delta t_j)$	$=$	$f_r^S(\mathbf{x}, \omega_o, \omega_i)$	$\cdot$	$f_r^T(\Delta t_j)$
`Separable.eval_t()`	$=$	`BSDF.eval()`	$\cdot$	`TP.pdf_delay()`
$p(\omega_i, \Delta t_j)$	$=$	$p(\omega_i)$	$\cdot$	$p(\Delta t_j)$
`Separable.pdf_t()`	$=$	`BSDF.pdf()`	$\cdot$	`TP.pdf_delay()`

Tabla 5.1: Evaluación de la BSDF y de la densidad de la distribución en la clase `Separable` como composición de su parte estacionaria y su perfil temporal.

5.2. Materiales separables modulados espacialmente

Para implementar los materiales separables modulados espacialmente, se decidió adaptar los métodos de la clase `TemporalProfile` (ver Sección 5.1.1) para incluir un parámetro adicional que representara la posición de intersección $\mathbf{x}$, pudiendo así generar y evaluar retrasos temporales que dependan de esta posición.

Para su implementación, se utilizó la teoría de texturas presentada en la Sección 3.3, donde los valores de los hiperparámetros del perfil temporal vendrían dados por imágenes de texturas.

Se valoraron las dos alternativas presentadas en la Figura 3.2. Para la primera 3.2(a), sería necesario, para cada perfil implementado, indicar qué canales corresponden a cada hiperparámetro. Además, según el número de hiperparámetros del perfil, se necesitaría admitir texturas con diferente número de canales. Por ejemplo, el perfil Constante (Sección 3.2.1) requiere de un único canal que define el retraso constante c, mientras que el perfil Epanechnikov (Sección 3.2.1) requiere de dos, uno para la media μ y otro para la desviación σ.

Esta primera alternativa, a pesar de ser sencilla de implementar, presenta dificultades de usabilidad para el usuario que debe conocer qué canales corresponden a qué hiperparámetro y cuántos canales debe tener la imagen para cada perfil. Por lo

tanto, se decidió implementar la alternativa (b) de la Figura 3.2, donde cada parámetro de un perfil temporal viene definido por una textura de un solo canal.

Para permitir también la definición de parámetros que no dependan de la posición $\mathbf{x}$ (para materiales no modulados espacialmente, por ejemplo) se decidió permitir que los valores de los hiperparámetros pudiesen ser tanto texturas como valores flotantes constantes. Esta solución se inspiró en el enfoque de Mitsuba, donde ciertos parámetros, como el color de un material difuso, pueden ser representados tanto por un color individual como por una textura. Para ello, fueron necesarias dos implementaciones distintas para la clase `mitsuba.Texture` (ver Sección 4.1.2):

- `ConstantTexture`: representa una textura constante, es decir, siempre devuelve el mismo valor para cualquier punto de la superficie.
- `TransientTexture`: puede ser una `ConstantTexture` o cualquier otra `mitsuba.Texture`.

Así, se puede definir cada atributo de un perfil temporal como una textura `TransientTextura`. Esta estará compuesta por una `ConstantTexture` si se ha definido como un valor constante, o por otra textura.

La necesidad de definir una textura constante, `ConstantTexture`, es garantizar que en todo caso `TransientTexture` esté compuesta por otra textura, a la que delega su evaluación `eval_1` (ver Sección 4.1.2). Con ello se evita que, durante la evaluación, sea necesario comprobar si se trata de un valor constante o de una textura, ya que esto se realiza únicamente en la construcción.

El esquema general para obtener los hiperparámetros, que luego se utilizan tanto para obtener muestras (método `sample_delay`) como para evaluar la densidad (método `pdf_delay`), es el siguiente:

Algorithm 4: Obtener parámetros específicos para un punto $\mathbf{x}$.

Data: $\mathbf{x} \in \Omega$
Result: θ
foreach $par \in BSDF$ **do**
    `// Obtener parámetros individuales`
    $\theta_i = par.\text{eval\_1}(\mathbf{x})$
end
$\theta = \{\theta_1, \ldots, \theta_n\}$
return θ

5.3. Composición de Materiales separables

Para definir la composición de materiales transitorios (Sección 3.4), se implementó una nueva clase `Composed`. Esta combina dos BSDF transitorias distintas y un peso

$w \in [0, 1]$, que determina la contribución de cada una.

Con esta clase, se puede implementar la Clorofila tal y como se explicó en la Sección 3.4. Un esquema resumen de cómo se define a nivel de objetos y clases se puede ver en la Figura 5.2. Para ver la definición exacta en Mitsuba 3 se puede recurrir al Apéndice B.3.

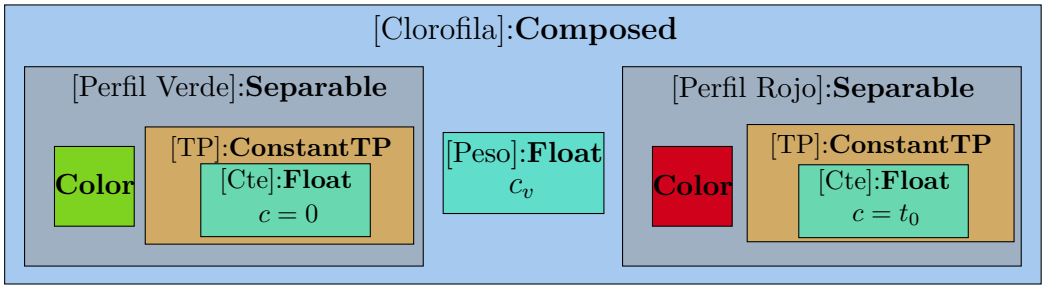


Figura 5.2: Representación esquemática de cómo definir un material de Clorofila (ver Sección 3.4) utilizando las clases implementadas en el Capítulo 5.

5.4. Integrador Transitorio

La incorporación de materiales transitorios obligó a modificar el integrador de Mitransient para considerar los retrasos temporales que estos materiales generan. En el Algoritmo 5 se resumen los ajustes realizados al Algoritmo 3. En esta nueva versión, cuando se obtiene un nuevo rayo r_{out}, también se guarda el retraso temporal asociado Δt que se suma al tiempo total del camino. El código en `C++` está disponible en el Apéndice B.2.

Algorithm 5: Calcular Aportación camino con Retrasos

Data: Rayo inicial r del pixel (x, y), escena S
Result: Radiancia L y tiempo total t

while *verdadero* **do**
  ... (Igual que Algoritmo 3)
  `// Samplear dirección, contribución y retraso.`
  $(r_{out}, f_t, \Delta t) \leftarrow$ **x**.sample_t(r)
  $T \leftarrow T \cdot f_t \quad r \leftarrow r_{out}$ `// Actualizar variables`
  $t \leftarrow t + \Delta t$ `// Añadir retraso temporal`
end
return (T, t)

Capítulo 6

Resultados

En este capítulo se presentan los resultados obtenidos tras la implementación de los materiales transitorios en el entorno de Mitsuba 3. Los resultados están organizados de manera progresiva, comenzando con una explicación de cómo se visualizan los fenómenos transitorios en general (Sección 6.1) y continuando con el análisis de los resultados para diferentes materiales transitorios (Sección 6.2), desde los más simples hasta las implementaciones más complejas.

En concreto, se comenzará introduciendo los resultados para los materiales transitorios separables (Sección 6.2.1), analizando los tres perfiles presentados en las secciones anteriores: Constante, Epanechnikov y Exponencial. A continuación, se explorarán los resultados obtenidos para materiales transitorios separables (Sección 6.2.2) con una textura sencilla. Finalmente, se analizarán diferentes composiciones de materiales transitorios (Sección 6.2.3) como la clorofila y uno que combina este con texturas.

6.1. Visualización transitoria

En escenas estáticas, basta con mostrar la imagen final de la escena renderizada. No obstante, cuando se trata de un entorno transitorio, es necesario representar cómo la luz viaja por la escena a lo largo del tiempo. Así, de una escena transitoria se puede visualizar: una representación **estática** de la escena, un **video** de la propagación de la luz en el tiempo, o diferentes **fotogramas** de este video que representan instantes concretos en el tiempo.

En esta sección, se trabajará en **distancias ópticas** (OPL) de tiempo (ver Sección 4.2) para evitar trabajar con tiempos muy pequeños. Esto se ha decidido para ayudar al lector a poder interpretar mejor los resultados. Además, las unidades de los perfiles temporales también están en estas unidades. Por ejemplo, un perfil Cte($c = 2$) significa un retraso constante de una distancia óptica de 2 m. Los tiempos en segundos

se pueden obtener con la fórmula presentada en la Ecuación 4.2.

En la Figura 6.1, se muestra un ejemplo para una escena básica. Aquí, en el instante inicial (OPL = 0 m o $t = 0$ ns) la luz se propaga desde la fuente de luz del techo hacia las diferentes partes de la escena. Además, es posible distinguir el frente de onda inicial de la luz directa (OPL = 1,6 m o $t = 53{,}37$ ns) de los frentes de onda de la luz indirecta, que se generan con los rebotes de la luz (OPL = 3,4 m o $t = 113{,}41$ ns).

Se recomienda al lector ver el Video Transitorio de esta escena para entender mejor cómo se visualiza realmente un resultado transitorio.

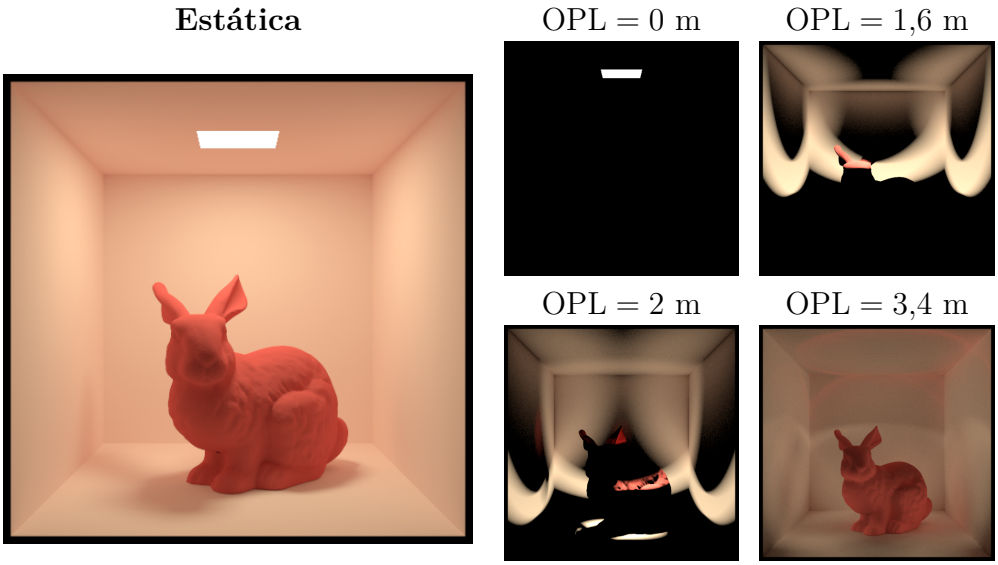


Figura 6.1: **(a)** Imagen de referencia en estado estacionario. **(b)** Imágenes transitorias para diferentes instantes. Video

6.2. Resultados Materiales Transitorios

En esta sección se introducen los resultados visuales obtenidos para los materiales transitorios implementados. Durante esta sección, se va a trabajar mayoritariamente con la escena ya presentada en la Figura 6.1. Esta escena es una Cornel Box sencilla con una figura principal: un Conejo de Stanford (Stanford bunny [27]) de color rojo difuso. Las paredes son totalmente difusas, estacionarias y de color blanco. Además, hay un único emisor en la parte superior. Durante el análisis, se va a modificar el material del conejo por diferentes **material transitorio** para analizar las diferencias con el material estático.

Además, todos los resultados tienen un video asociado accesible desde Google Drive.

6.2.1. Materiales Transitorios Separables

Se comienza analizando los resultados obtenidos para materiales transitorios separables, probando cada uno de los perfiles temporales.

Perfil Constante

En primer lugar, se analizan los perfiles constantes para diferentes valores de la constante c (ver Tabla 3.1), los cuales se muestran en la Figura 6.2.

Aquí se puede observar que la propagación de la luz directa en las paredes (parte blanca), que son estáticas, se mantiene constante para un OPL fijo, mientras que el conejo, que tiene un perfil temporal constante, cuando c aumenta, su reemisión se retrasa hasta el caso de no aparecer. Así, en OPL $= 3$ m y para $c = 2$, el conejo empieza a reemitir como en OPL $= 1$ m y $c = 0$ (sin retrasos), como era de esperar.

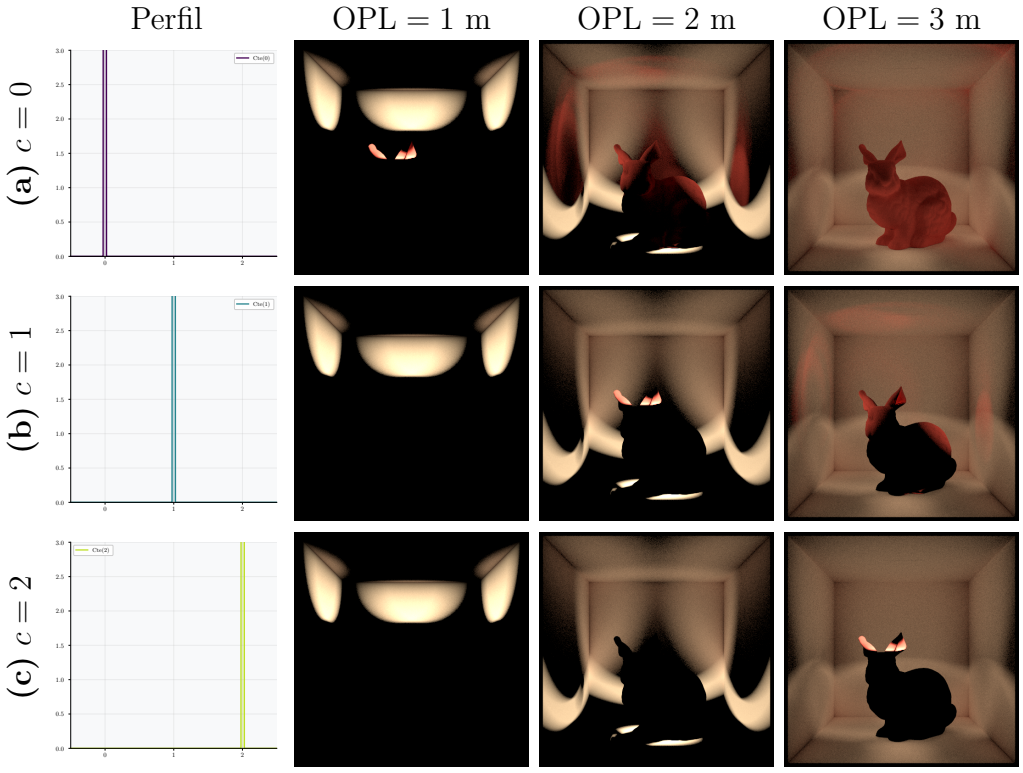


Figura 6.2: Resultados para distintos valores de c en un perfil constante Cte(c). (a) $c = 0$, es decir, sin retraso; **(b)** $c = 1{,}0$ y **(c)** $c = 2{,}0$. En la primera columna se representa una gráfica con la función de densidad del perfil temporal correspondiente. Videos.

Perfil Exponencial

A continuación se analiza el perfil exponencial Exp(λ) para diferentes valores de λ (ver Tabla 3.1). En la Figura 6.3 se puede ver que, con $\lambda = 2$, la reemisión se realiza casi completamente en los primeros instantes con toda la intensidad, mientras que con $\lambda = 0{,}5$, esta se reparte más en el tiempo.

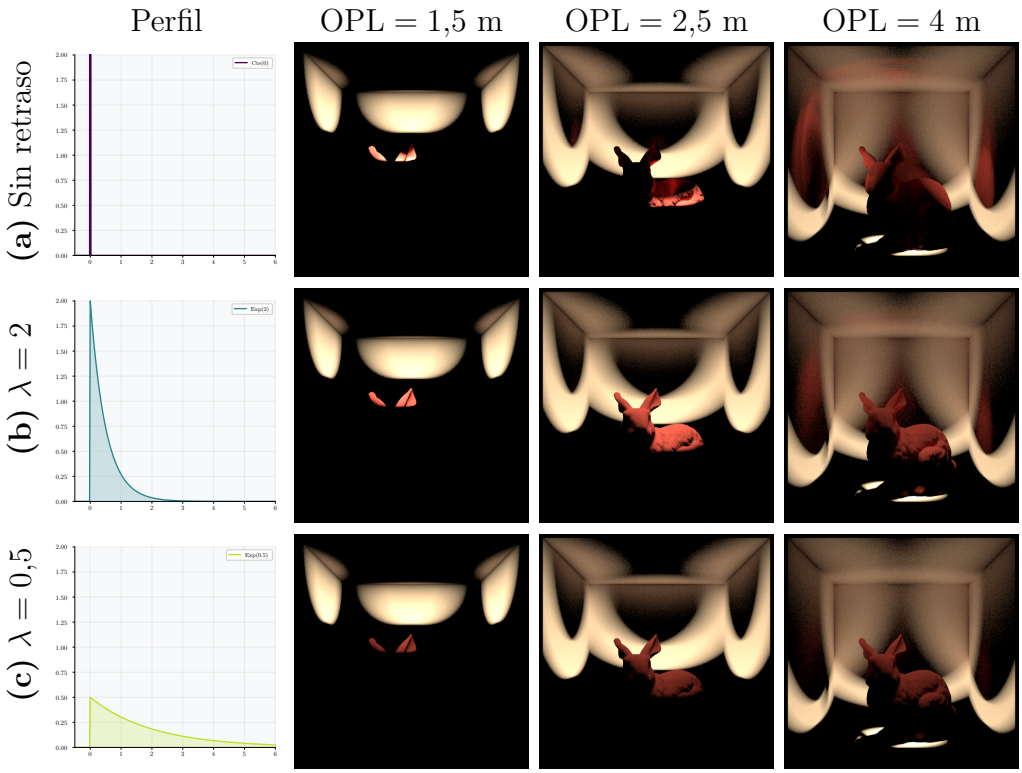


Figura 6.3: Resultados para distintos valores de λ en un perfile Exponencial Exp(λ). **(a)** Representa un material sin retrasos Cte(0), **(b)** un material con kernel Exponencial Exp(2) y **(c)** con Exp(0,5). En la primera columna se representa una gráfica con la función de densidad del perfil temporal correspondiente. Videos.

Perfil Epanechnikov

Analizamos ahora el perfil Epanechnikov para diferentes valores de μ y σ, que controlan la media y la dispersión de los retrasos temporales respectivamente (ver Tabla 3.1). Los resultados gráficos se pueden ver en la Figura 6.4. Aquí se pueden llegar a una serie de conclusiones. En primer lugar, cuando σ es alto, la propagación de la luz se distribuye más en el tiempo, perdiendo intensidad, mientras que se concentra y gana intensidad cuando σ es bajo. En segundo lugar, se puede ver que para $\mu = \sigma = 0{,}1$ el resultado es casi idéntico al caso sin retrasos, ya que se está concentrando toda la densidad cerca del cero.

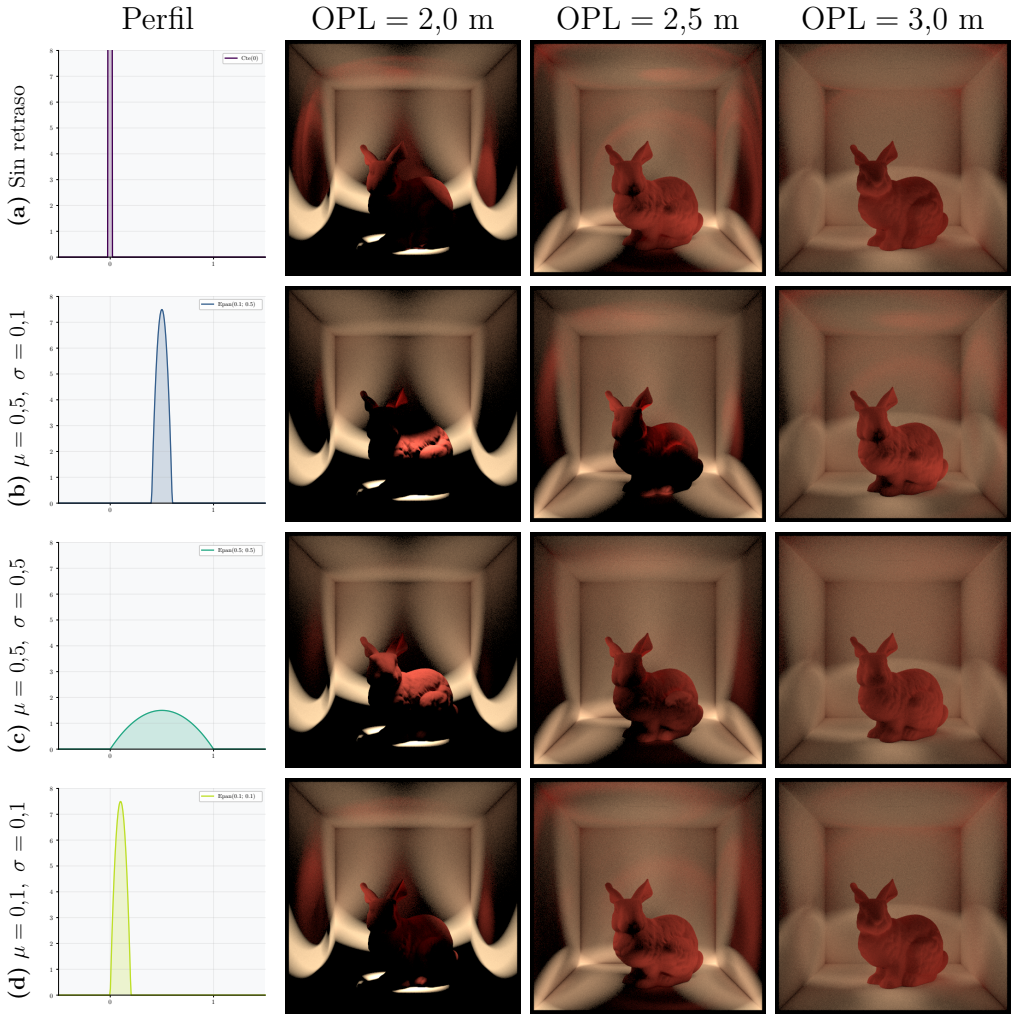


Figura 6.4: Resultados para distintos valores de μ y σ en un perfil Epanechnikov Epan(μ, σ). **(a)** Representa un material sin retrasos Cte(0), **(b)** un material con kernel Epanechnikov con $(\mu, \sigma) = (0{,}5, 0{,}1)$, **(c)** con $(0{,}5, 0{,}5)$ y **(d)** con $(0{,}1, 0{,}1)$ simulando un material sin retraso. En la primera columna se representa una gráfica con la función de densidad del perfil temporal correspondiente. Videos.

6.2.2. Materiales Transitorios Separables Modulados Espacialmente

Ahora se van a analizar los materiales modulados espacialmente con una textura sencilla, que se puede ver a nivel de reflectancia en la Figura 6.5. Esta está compuesta por dos colores, blanco y negro, que representan dos zonas distintas que tienen diferentes perfiles temporales.

En este caso, se va a tomar que el color blanco corresponde a un perfil constante con retraso $c = 1$ y el negro toma un perfil constante sin retraso ($c = 0$). Los resultados obtenidos se pueden ver en la Figura 6.5.

Nótese que la emisión en las partes blancas del conejo se retrasa en el tiempo, emitiéndose 1 m después de la primera emisión en las partes negras. Sin embargo, esto no se ve en la imagen estática debido a que el perfil de emisión (color) es idéntico en ambas partes, solo que está desplazado en el tiempo. Esto pone en manifiesto que el dominio transitorio codifica información que es imperceptible en el dominio estacionario.

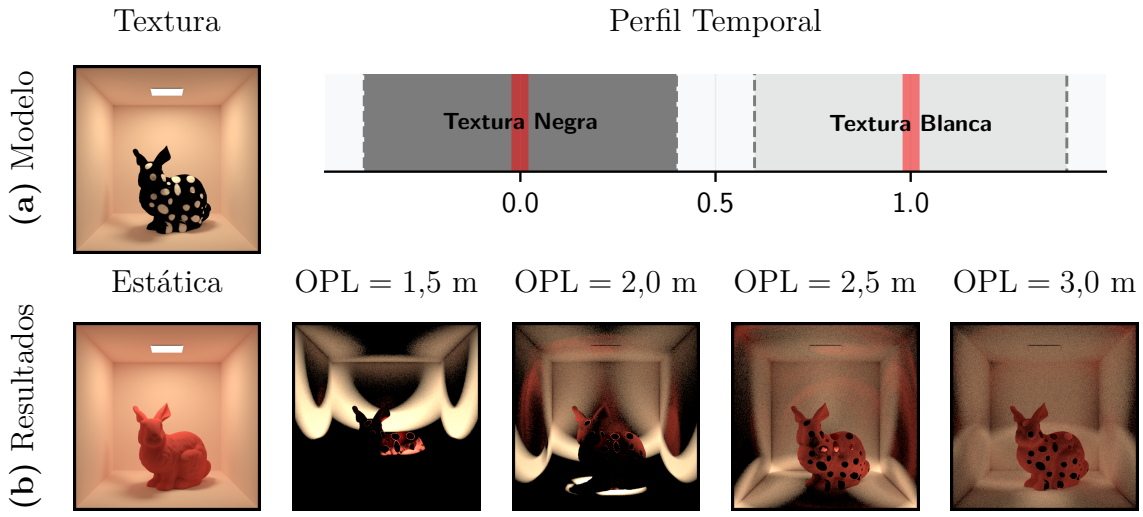


Figura 6.5: **(a)** Modelo y **(b)** Resultados para un material transitorio básico con texturas. Video

6.2.3. Composición Materiales Transitorios

A continuación se exponen los resultados obtenidos para la composición de materiales transitorios. En primer lugar, se muestra el material de clorofila explicado en la Sección 5.3 y, en segundo lugar, se expone un ejemplo más complejo que compone dos materiales transitorios modulados espacialmente por la textura expuesta previamente.

Clorofila

Se han analizado los resultados de la escena con perfil de clorofila (ver Sección 5.3) con retraso $t_0 = 1$ m y para dos proporciones de verde distintas: $c_v = 30\,\%$ y $c_v = 70\,\%$. Los resultados transitorios para este material se pueden ver en la Figura 6.6.

En ambos casos se observa una emisión difusa de verde en el instante t, seguida de una emisión roja equivalente en el instante $t + 1$. La intensidad de la emisión verde es directamente proporcional al porcentaje de verde c_v. Además, esto también afecta al color final de la figura estática, donde para $c_v = 70\,\%$ el color estático es más verde debido a que la emisión de verde en el tiempo es mayor que la de rojo.

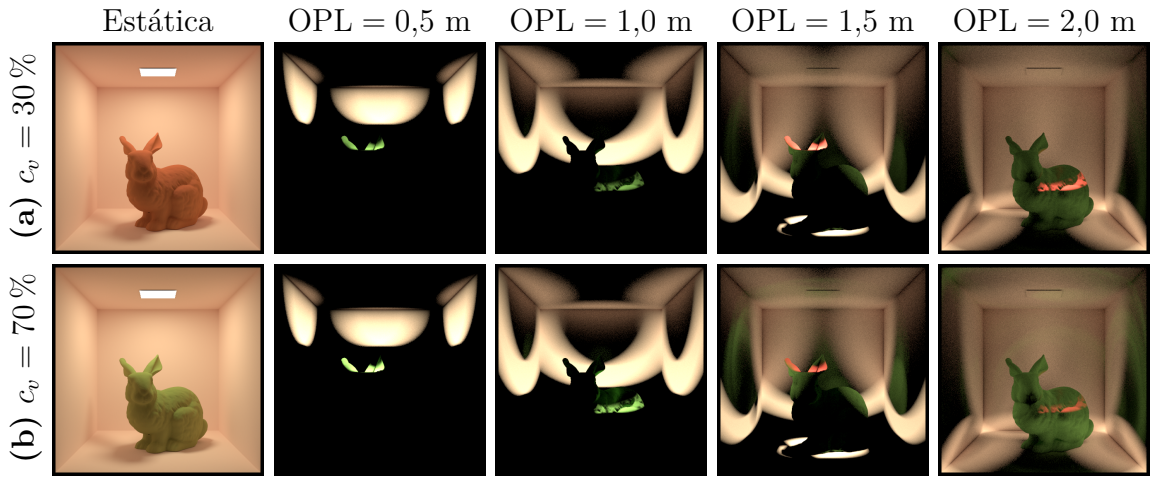


Figura 6.6: Resultados obtenidos para un perfil temporal de clorofila (ver Sección 5.3) para $t_0 = 1$ m para diferentes valores de c_v. **(a)** $c_v = 30\,\%$ y **(b)** $c_v = 70\,\%$. Videos.

Ejemplo complejo

Finalmente, se ha implementado un ejemplo más complejo que combina dos materiales transitorios con texturas. Los resultados se pueden ver en la Figura 6.7.

En cada uno de los fotogramas de esta figura se aprecian cada uno de los eventos del perfil temporal. Se comienza con una emisión azul únicamente en la parte negra del conejo, seguido de otra reemisión sobre la misma zona de color amarillo. Después ocurre lo mismo pero en la parte blanca del conejo, tal y como se describe en el perfil temporal expuesto en la misma imagen.

Además, el color de la imagen estática es la combinación de ambos colores. Como ocurría en el ejemplo de texturas, las diferentes partes no se ven en la imagen estática por el mismo motivo: el perfil de emisión es idéntico, pero desplazado en el tiempo.

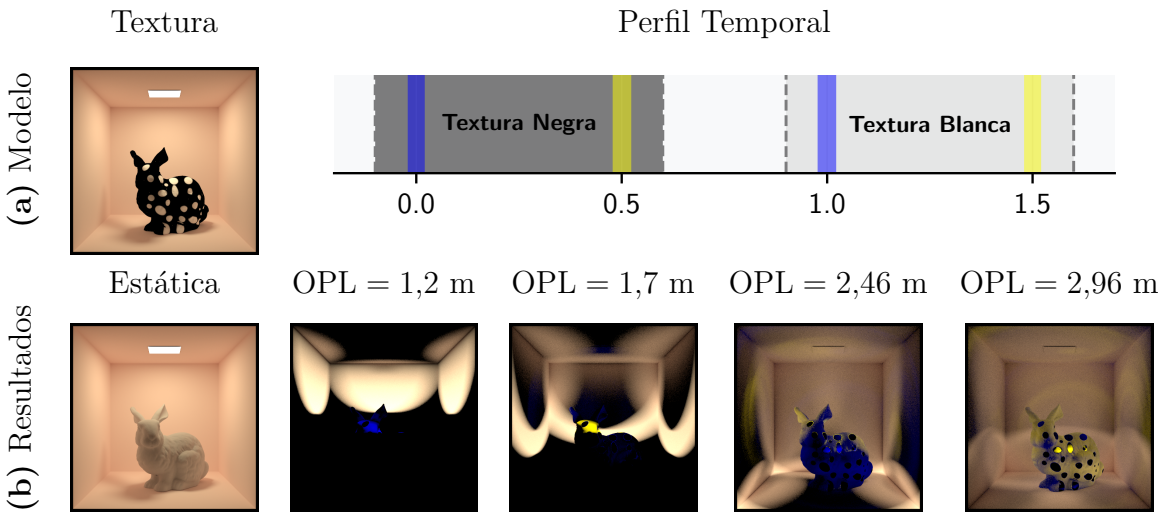


Figura 6.7: Modelo **(a)** y Resultados **(b)** para una composición de materiales transitorios con texturas. Video

6.2.4. Escenas Completas

En esta sección se expone una escena más compleja que representa un bosque con una serpiente, una mariposa y un escarabajo. Aquí, las partes verdes de las hojas presentes están compuestas por un material de clorofila con $t_0 = 2$ m. La imagen estática de la escena junto a diferentes fotogramas del video transitorio se puede ver en la Figura 6.8.

Aquí se aprecia que únicamente las hojas reemiten un pulso rojo tras OPL = 2 m, mientras que el resto de elementos de la escena, como la mariposa o el tronco, no.

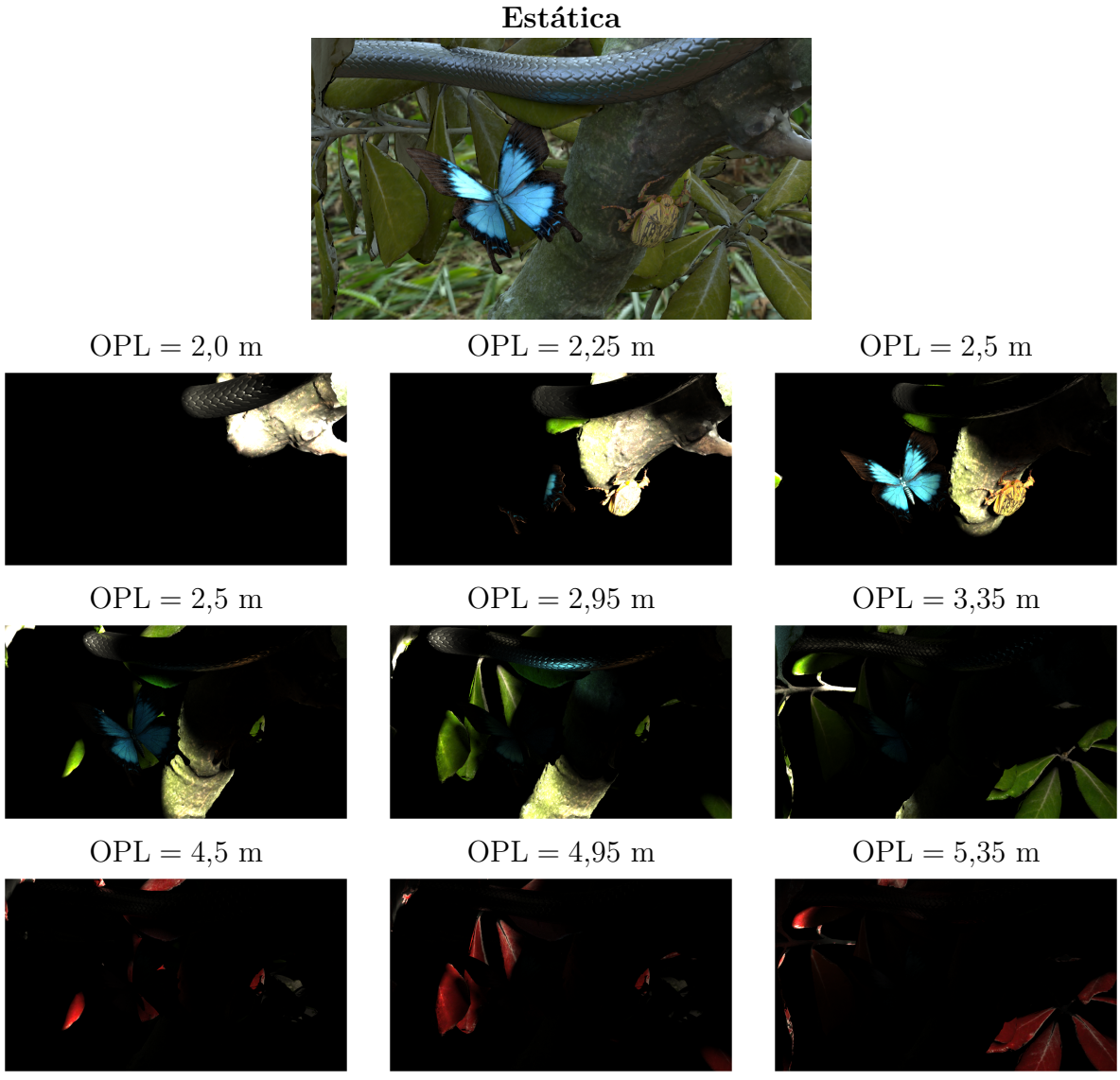


Figura 6.8: Arriba, escena de referencia utilizada para analizar los resultados de materiales transitorios. Esta presenta en las hojas un perfil de clorofila donde el pulso rojo se emite en $t_0 = 2$ m. Debajo, fotogramas obtenidos para distintos instantes de tiempo. Video

6.3. Análisis de rendimiento

Una vez implementados los materiales transitorios en el marco de Mitsuba y Mitransient, se realizó un análisis de rendimiento para comprobar que la implementación no había alterado significativamente la convergencia de los algoritmos.

Para ello, se analizó el tiempo requerido para el renderizado de una escena compleja estática para un número incremental de **muestras por píxel** o SPP [1]. En este análisis se compararon los resultados para los siguientes casos:

1. Utilizando la versión previa de Mitransient y Mitsuba, sin modificaciones.

2. Utilizando la versión nueva utilizando de forma independiente: la escena sin materiales transitorios (estática), añadiendo un material separable y añadiendo una composición de materiales transitorios.

El análisis se realizó en `MAKINON1`, un equipo de sobremesa con 2 procesadores Intel Xeon E5-2697 v4 (36 núcleos, 72 hilos a 3.6 GHz) y 256 GB de memoria RAM.

Los resultados se pueden ver en la Figura 6.9, donde se concluye que los tiempos antes y después de las modificaciones son prácticamente iguales, por lo que no ha aumentado la complejidad de Mitsuba.

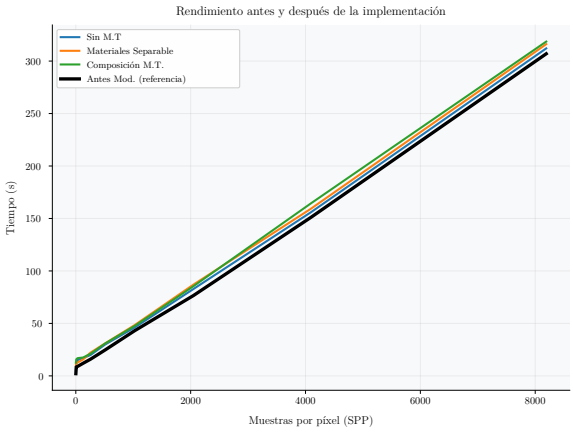


Figura 6.9: Gráfica para representar el tiempo requerido para renderizar una escena para diferentes valores de SPP. Aquí `Sin M.T` son los tiempos tras la modificación sin añadir ningún material transitorio, `Material Separable` y `Composición M.T` añadiendo un material separable y uno composición respectivamente (tras la modificación) y `Antes Mod.` el tiempo obtenido con una versión de Mitsuba y Mitransient previa.

[1] Las muestras por píxel o SPP es el número de caminos que se trazan para obtener la radiancia incidente en cada uno de los píxeles de la escena.

Capítulo 7

Conclusiones

En este trabajo se amplió el motor transitorio de Mitransient con el objetivo de definir e implementar materiales que exhiben retrasos temporales en la reemisión de la luz, como es el caso de los materiales fluorescentes. El primer paso consistió en establecer un marco teórico riguroso que sirviera como base para describir formalmente este tipo de materiales. A partir de esta base, se procedió a modificar tanto el código de Mitransient como el de Mitsuba 3, adaptando ambos sistemas para soportar estos nuevos materiales.

Una vez implementado el soporte necesario, se implementó un subconjunto de estos materiales cuyos retrasos temporales son parcialmente o totalmente independientes de la parte espacial del material. Finalmente, se analizaron los resultados obtenidos para los modelos implementados, comprobando la corrección de los mismos.

Además del trabajo puramente de implementación, una parte fundamental de este proyecto fue la definición de una base sólida y coherente para la definición de estos materiales, lo que resultó de especial interés debido a la formación en Matemáticas del autor. La implementación, por su parte, también representó un desafío considerable debido, principalmente, a la arquitectura interna de Mitsuba 3, que no está ampliamente documentada y presenta diversas particularidades. Esto exigió mucho tiempo para analizar el código fuente y experimentar cómo se podía modificar.

Una vez se comprendió el funcionamiento de esta herramienta, gran parte del trabajo restante se centró en la búsqueda de diferentes alternativas para la implementación de los nuevos materiales, evaluando cuidadosamente las ventajas y desventajas de cada método. El proceso finalizó con una validación exhaustiva de los resultados, verificando la correcta implementación de estos materiales y su comportamiento en distintos escenarios y condiciones.

Trabajo Futuro. A partir de este trabajo quedan frentes abiertos que explorar como trabajo a futuro. La definición de una base matemática para simular materiales

que presentan retrasos temporales abre la puerta a la exploración de materiales transitorios más complejos. Esto incluye materiales multicapa o con multifacetas, cuya implementación requerirá el desarrollo de modelos físicos más avanzados adaptados al dominio transitorio.

Además, la simulación de materiales transitorios complejos puede ser una herramienta fundamental para otros campos. En la medicina, podría usarse para modelar y estudiar cómo la luz se propaga a través de tejidos biológicos. En la ciencia de materiales, podría ayudar a predecir el comportamiento de los materiales a escalas de tiempo muy pequeñas, siendo útil para campos como la física de materiales, la óptica y la química.

Conclusiones Personales. Este trabajo me ha permitido consolidar los conocimientos adquiridos durante la carrera de **Ingeniería Informática**, sobre todo en programación orientada a objetos y en informática gráfica. Además, también he podido poner en valor mis conocimientos en **Matemáticas**, lo que ha resultado especialmente gratificante. Por otro lado, este trabajo me ha abierto la puerta al mundo de la investigación, donde he podido descubrir la gran cantidad de oportunidades y líneas de trabajo existentes, tanto en Informática Gráfica como en áreas relacionadas.

En resumen, considero que la experiencia ha contribuido significativamente a mi formación académica y me ha permitido sentar una base útil para futuros proyectos de investigación.

Capítulo 8

Bibliografía

[1] Andreas Velten, Di Wu, Adrian Jarabo, Belen Masia, Christopher Barsi, Chinmaya Joshi, Everett Lawson, Moungi Bawendi, Diego Gutierrez, and Ramesh Raskar. Femto-photography: capturing and visualizing the propagation of light. *ACM Trans. Graph.*, 32(4), July 2013.

[2] Adrián Jarabo, Belén Masiá, Julio Marco, and Diego Gutierrez. Recent advances in transient imaging: A computer graphics and vision perspective. *CoRR*, abs/1611.00939, 2016.

[3] Adrian Jarabo, Julio Marco, Adolfo Munoz, Raul Buisan, Wojciech Jarosz, and Diego Gutierrez. A framework for transient rendering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 33(6), November 2014.

[4] Julio Marco, Quercus Hernandez, Adolfo Muñoz, Yue Dong, Adrian Jarabo, Min H. Kim, Xin Tong, and Diego Gutierrez. Deeptof: Off-the-shelf real-time correction of multipath interference in time-of-flight imaging. *ACM Transactions on Graphics (SIGGRAPH Asia 2017)*, 36(6):1–12, 2017.

[5] Nikhil Naik, Shuang Zhao, Andreas Velten, Ramesh Raskar, and Kavita Bala. Single view reflectance capture using multiplexed scattering and time-of-flight imaging. *ACM Transactions on Graphics (SIGGRAPH Asia 2011)*, 30(6):1–10, 2011.

[6] Andreas Velten, Thomas Willwacher, Otkrist Gupta, Ashok Veeraraghavan, Moungi G. Bawendi, and Ramesh Raskar. Recovering three-dimensional shape around a corner using ultrafast time-of-flight imaging. *Nature Communications*, 3:745, 2012.

[7] Victor Arellano, Diego Gutierrez, and Adrian Jarabo. Fast back-projection for non-line of sight reconstruction. *Optics Express*, 25(10):11574–11583, 2017.

[8] Matthew O'Toole, David B. Lindell, and Gordon Wetzstein. Confocal non-line-of-sight imaging based on the light-cone transform. *Nature*, 555(7696):338–341, 2018.

[9] Xiaochun Liu, Ibón Guillén, Marco La Manna, Ji Hyun Nam, Syed Azer Reza, Toan Huu Le, Adrian Jarabo, Diego Gutierrez, and Andreas Velten. Non-line-of-sight imaging using phasor-field virtual wave optics. *Nature*, 572(7771):620–623, 2019.

[10] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022. https://mitsuba-renderer.org.

[11] Diego Royo, Miguel Crespo, and Jorge Garcia-Pueyo. mitransient. https://github.com/diegoroyo/mitransient, 2023.

[12] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2016.

[13] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[14] F O Bartell, E. L. Dereniak, and W. L Wolfe. The Theory And Measurement Of Bidirectional Reflectance Distribution Function (Brdf) And Bidirectional Transmittance Distribution Function (BTDF). In Gary H. Hunt, editor, *Radiation Scattering in Optical Systems*, volume 0257, pages 154 – 160. International Society for Optics and Photonics, SPIE, 1981.

[15] B. Duvenhage, K. Bouatouch, and D. G. Kourie. Numerical verification of bidirectional reflectance distribution functions for physical plausibility. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, SAICSIT '13, page 200–208, New York, NY, USA, 2013. Association for Computing Machinery.

[16] Jaroslav Křivánek. Monte carlo estimation and integration. In *Basic Monte Carlo Ray Tracing*, chapter 5. 2023. Accessed: 2025-05-18.

[17] Eric Veach. *Robust monte carlo methods for light transport simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[18] Laurent Belcour. Efficient rendering of layered materials using an atomic decomposition with statistical operators. *ACM Trans. Graph.*, 37(4), July 2018.

[19] 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*, EGSR'07, page 195–206, Goslar, DEU, 2007. Eurographics Association.

[20] Bo Sun, Kalyan Sunkavalli, Ravi Ramamoorthi, Peter N. Belhumeur, and Shree K. Nayar. Time-varying brdfs. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):595–609, 2007.

[21] Takuto Narumoto, Hiroaki Santo, and Fumio Okura. Synthesizing time-varying brdfs vianbsp;latent space. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part LXXI*, page 109–124, Berlin, Heidelberg, 2024. Springer-Verlag.

[22] A. Fichet, L. Belcour, and P. Barla. Non-orthogonal reduction for rendering fluorescent materials in non-spectral engines. *Computer Graphics Forum*, 43(4):e15150, 2024.

[23] O. Nalbach, H.-P. Seidel, and T. Ritschel. Practical capture and reproduction of phosphorescent appearance. *Computer Graphics Forum*, 36(2):409–420, 2017.

[24] Rupsa Datta, Tiffany M Heaster, Joe T Sharick, Amani A Gillette, and Melissa C Skala. Fluorescence lifetime imaging microscopy: fundamentals and advances in instrumentation, analysis, and applications. *J. Biomed. Opt.*, 25(7):1–43, May 2020.

[25] Xuchen Wang, Mohammad Sajjad Mirmoosa, Viktar S. Asadchy, Carsten Rockstuhl, Shanhui Fan, and Sergei A. Tretyakov. Metasurface-based realization of photonic time crystals. *Science Advances*, 9(14):eadg7541, 2023.

[26] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, NY, 1986.

[27] The Stanford 3D Scanning Repository. Stanford bunny, n.d. Available at: https://graphics.stanford.edu/data/3Dscanrep/.

[28] Adrian Jarabo and Victor Arellano. Bidirectional rendering of vector light transport. *Computer Graphics Forum*, 37(6):96–105, 2018.

Lista de Figuras

Lista de Tablas

Apéndices

Apéndice A

Contexto Matemático

Se introducen ahora los diferentes conceptos que soportan la teoría explicada durante la memoria.

En la Sección A.1 se presenta la teoría para la parametrización de texturas, necesaria para entender cómo se mapea una superficie 3D, como puede ser una esfera, a un espacio 2D y de ahí, a un fichero de texturas.

Por otro lado, en la Sección A.2 se presentan los teoremas y demostraciones que dan validez a ciertos resultados presentados en la memoria. En concreto, se presentan los teoremas para la formación de BSDFs transitorias así como el teorema para el muestreo por la inversa de la CDF.

A.1. Parametrización de Texturas

En informática gráfica, para poder mapear una textura 2D en una superficie 3D, se utiliza una parametrización:

$$\begin{aligned} \Phi : U \subset \mathbb{R}^2 &\longrightarrow \Omega \\ (u, v) &\longrightarrow \Phi(u, v) = \mathbf{x} \end{aligned}$$

que mapea parte de la geometría desde un conjunto abierto U de $\mathbb{R}^2$ a la geometría 3D. En este espacio 2D, se define una transformación

$$\begin{aligned} \Psi : U \subset \mathbb{R}^2 &\longrightarrow \text{Espacio Texturas} \\ (u, v) &\longrightarrow \Psi(u, v) \end{aligned}$$

al dominio de la imagen que define la textura. Así, se obtiene la función $\Psi \circ \Phi^{-1}(\mathbf{x})$ que asigna a cada punto de la geometría una posición en la imagen. Este esquema se puede ver en la Figura A.1.

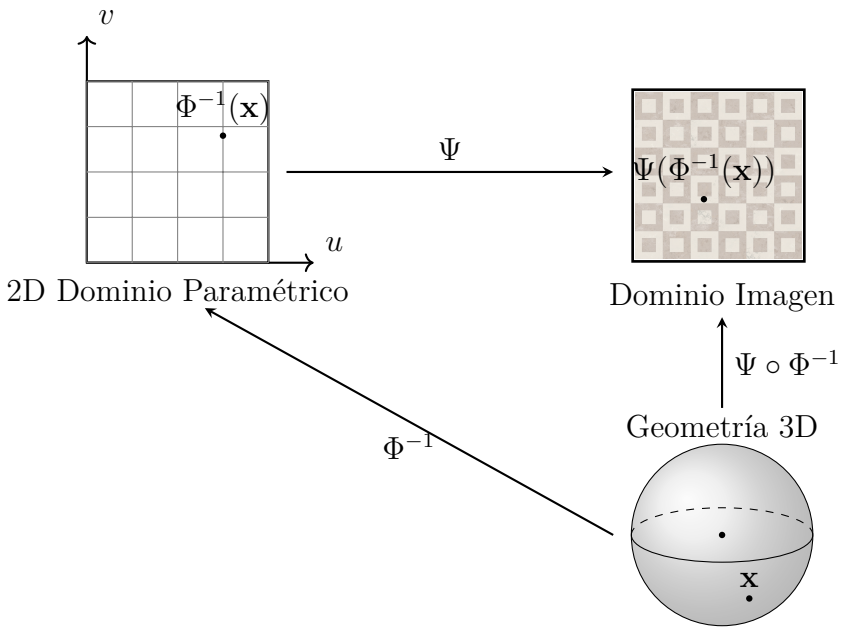


Figura A.1: Representación de transformaciones entre el dominio paramétrico de las coordenadas (u, v), el dominio de la imagen que define la textura y la geometría 3D que se quiere modelar. Aquí, cada punto $\mathbf{x}$ de la geometría, tiene su correspondiente $\Phi^{-1}(\mathbf{x})$ en el espacio paramétrico, que corresponde a su vez a un punto $\Psi(\Phi(\mathbf{x}))$ en la imagen.

Nótese que, como el número de píxeles de la imagen es finito, el espacio paramétrico U (y, por ende, la geometría Ω) se discretiza en pequeños parches al proyectarse sobre el mismo píxel. Esto induce una partición del dominio paramétrico en un recubrimiento finito de subconjuntos $\mathcal{I} = \{I_1, \ldots, I_n\}$, con $U = \bigcup_{i=1}^{n} I_i$, donde cada subconjunto I_i está asociado a un píxel específico y, por ende, a un valor igual de la textura.

A.2. Teoremas

Se presentan a continuación los diferentes teoremas necesarios para validar los resultados presentados en la memoria.

A.2.1. Muestreo por inversa de CDF

Se comienza demostrando el teorema que permite obtener muestras aleatorias independientes a partir de muestras uniformes $U[0, 1]$ utilizando la inversa de la función de densidad acumulativa.

Teorema 1 (Obtención de muestras utilizando la CDF)**.** *Sea $p : \Delta t \longrightarrow \mathbb{R}$ la función de densidad de una variable aleatoria continua $X \sim D_x$ para una distribución concreta D_x. Si F_x^{-1} es la función acumulativa inversa de X y $Y \sim U[0, 1]$ es una variable*

continua uniforme en $[0, 1]$*, entonces:*

$$Z = F_x^{-1}(Y) \sim D_x$$

Demostración. Basta calcular la CDF de Z, es decir $F_z(z) = P(Z \leq z)$:

$$\begin{aligned} P(Z \leq z) &= P\left(F_x^{-1}(Y) \leq z\right) \\ &= P\left(Y \leq F_x(z)\right) \\ &= F_y\left(F_x(z)\right) \\ &= F_x(z) \end{aligned}$$

Como la CDF completamente determina la distribución de una variable aleatoria y la CDF de Z coincide con la de X, concluimos que tienen la misma distribución, es decir, $Z \sim D_x$. □

A.2.2. Composición Materiales Separables

En esta sección, se exponen los teoremas que permiten definir BSDFs transitorias a partir de una función de probabilidad y una BSDF estacionaria. Primero se demuestra para el caso separable, es decir, con independencia total del punto de intersección $\mathbf{x}$ y luego se demuestra para el caso modulado espacialmente, donde hay dependencia con $\mathbf{x}$.

Teorema 2 (Composición de materiales transitorios separables)**.** *Sea* $f_r^S(\mathbf{x}, \omega_o, \omega_i)$ *una BSDF estacionaria y* $p : \Delta T \subset \mathbb{R}^{\geq 0} \longrightarrow \mathbb{R}$ *una función de densidad con soporte no negativo. Entonces:*

$$f_r(\mathbf{x}, \omega_o, \omega_i, \Delta t) = f_r^S(\mathbf{x}, \omega_o, \omega_i) \cdot p(\Delta t)$$

es una BSDF válida.

Demostración. Basta comprobar las tres propiedades básicas planteadas en las Ecuaciones 2.4, 2.5 y 2.6. La propiedad de positividad es inmediata ya que tanto f_r^S como p son positivas. Por otra parte, como f_r^S es una BSDF, cumple la propiedad de simetría. Así, f_r también lo cumple:

$$f_r(\mathbf{x}, \omega_i, \omega_o, \Delta t) = f_r^S(\mathbf{x}, \omega_i, \omega_o) \cdot p(\Delta t) = f_r^S(\mathbf{x}, \omega_o, \omega_i) \cdot p(\Delta t) = f_r(\mathbf{x}, \omega_o, \omega_i, \Delta t)$$

Finalmente, como $p(\Delta t)$ es una función de densidad, cumple que:

$$\int_{\Delta T} p(\Delta t) d\Delta t = 1.$$

Por lo tanto, la propiedad de conservación de energía también se cumple:

$$\begin{aligned}\int_{\Delta T}\int_{\Omega} f_r(\mathbf{x},\omega_o,\omega_i,\Delta t)\; d\Delta t\; d\omega_i &= \int_{\Delta T}\int_{\Omega} f_r^S(\mathbf{x},\omega_o,\omega_i)\cdot p(\Delta t)\; d\Delta t\; d\omega_i \\ &= \int_{\Delta T} p(\Delta t)\;\left(\int_{\Omega} f_r^S(\mathbf{x},\omega_o,\omega_i)\; d\omega_i\right) d\Delta t \\ &\leq \int_{\Delta T} p(\Delta t) d\Delta t \\ &\leq\; 1\end{aligned}$$

□

Teorema 3 (Composición de materiales transitorios separables modulados espacialmente)**.** *Sea $f_r^S(\mathbf{x},\omega_o,\omega_i)$ una BSDF estacionaria y sea $p(\Delta t \mid \mathbf{x})$ una función de densidad con soporte $\mathbb{R}^{\geq 0}$ para todo $\mathbf{x}\in\Omega$. Entonces*

$$f_r(\mathbf{x},\omega_o,\omega_i,\Delta t) = f_r^S(\mathbf{x},\omega_o,\omega_i)\cdot p(\Delta t \mid \mathbf{x})$$

define una BSDF válida.

Demostración. La demostración es idéntica a la del Teorema 2, ya que f_r^S conserva simetría y energía por hipótesis, y $p(\Delta t \mid \mathbf{x})$ es una función de densidad válida con soporte no negativo. □

A.2.3. Composición de BSDFs transitorias separable

Se continúa con la demostración que permite construir una BSDF válida como una suma ponderada de BSDFs separables.

Teorema 4 (Composición de BSDFs transitorias separables)**.** *Sea $\{f_k(\mathbf{x},\omega_o,\omega_i,\Delta t)\}_{k=1}^n$ un conjunto finito de BSDF separables, es decir,*

$$f_k(\mathbf{x},\omega_o,\omega_i,\Delta t) = f_k^S(\mathbf{x},\omega_i,\omega_o)\cdot p_k(\Delta t \mid \mathbf{x})$$

donde f_k^S son un conjunto de BSDFs estáticas y $p_k(\Delta t \mid \mathbf{x})$ son funciones de densidad temporales con dominio positivo $\Delta t > 0$. Además, sean $\{c_k\}_{k=1}^n$ coeficientes tales que $c_k > 0\ \forall k$ y $\sum_{k=1}^n c_k = 1$. Entonces:

$$f_r(\mathbf{x},\omega_o,\omega_i,\Delta t) = \sum_{k=1}^{n} c_k\cdot f_k(\mathbf{x},\omega_o,\omega_i,\Delta t) = \sum_{k=1}^{n} c_k\cdot f_k^S(\mathbf{x},\omega_o,\omega_i)\cdot p_k(\Delta t \mid \mathbf{x})$$

es una BSDF válida.

Demostración. Al igual que en las demostraciones previas, veamos que las tres condiciones de una BSDF se cumplen. La propiedad de positividad es inmediata ya

que es una combinación con pesos positivos de funciones positivas. La propiedad de simetría es inmediata ya que cada una de las f_k^S es simétrica. Por otro lado, la propiedad de conservación de energía:

$$\begin{aligned}
\int_{\Delta T}\int_{\Omega} f_r(\mathbf{x},\omega_o,\omega_i,\Delta t)\ d\omega_i\ d\Delta t &= \int_{\Delta T}\int_{\Omega}\sum_{k=1}^{n} c_k\cdot f_k^S(\mathbf{x},\omega_o,\omega_i)\cdot p_k(\Delta t\mid \mathbf{x})\ d\omega_i\ d\Delta t \\
&\overset{(1)}{=} \sum_{k=1}^{n}\int_{\Delta T}\int_{\Omega} c_k\cdot f_k^S(\mathbf{x},\omega_o,\omega_i)\cdot p_k(\Delta t\mid \mathbf{x})\ d\omega_i\ d\Delta t \\
&= \sum_{k=1}^{n} c_k\cdot\int_{\Delta T} p_k(\Delta t\mid \mathbf{x})\left(\int_{\Omega} f_k^S(\mathbf{x},\omega_o,\omega_i)\ d\omega_i\right)\ d\Delta t \\
&\leq \sum_{k=1}^{n} c_k\cdot\int_{\Delta T} p_k(\Delta t\mid \mathbf{x})\ d\Delta t \\
&= \sum_{k=1}^{n} c_k = 1.
\end{aligned}$$

Nótese que en (1) se puede mover el sumatorio y la integral ya que se trata de un sumatorio finito. □

Apéndice B

Extensión Implementación

Se introducen a continuación ciertos conceptos para ayudar al lector a entender diferentes conceptos relacionados con la implementación.

En la Sección B.1 se introducen ciertos conceptos extra de Mitsuba que justifican, en menor medida, ciertas decisiones de implementación, mientras que en la sección B.2 se exponen en `C++` los algoritmos de Path-Tracing diseñados para la implementación de los materiales transitorios.

B.1. Extensión Mitsuba

En esta sección se introducen ciertos conceptos relativos a Mitsuba que ayudan a entender ciertas decisiones de diseño.

B.1.1. Escenas en Mitsuba 3

Para definir escenas en Mitsuba 3 desde `Python` existen dos enfoques diferentes:

1. **Mediante diccionarios**: Se definen las diferentes componentes de la escena (sensores, emisores, geometrías, materiales, etc.) directamente en esta estructura de `Python`.

2. **Desde ficheros XML**: Permite una definición más intuitiva utilizando etiquetas `XML` específicas para cada componente. Además, se definen en ficheros independientes de `Python`, haciéndolo más reutilizable.

En la Figura B.1 se muestra una comparativa para la definición de una escena sencilla utilizando un diccionario y un fichero `XML`.

Definir escenas de forma intuitiva es fundamental para la experiencia de usuario al utilizar el código implementado. Por ello, durante la implementación de los materiales transitorios, se buscaron formas de simplificar la definición de este tipo de materiales.

```python
{
    type: 'scene',
    # Cámara
    sensor: {
        type: 'perspective',
        ...
    },
    # Fuente de luz
    emitter: {
        ...
    },
    # Geometría Esfera
    sphere: {
        type: 'sphere',
        ...
    }
}
```

(a)

```xml
<scene>

    <!-- Cámara -->
    <sensor type="perspective">
        ...
    </sensor>

    <!-- Fuente de luz -->
    <emitter type="constant">
        ...
    </emitter>

    <!-- Geometría Esfera -->
    <shape type="sphere">
        ...
    </shape>
</scene>
```

(b)

Figura B.1: Comparación entre la definición de escenas en Mitsuba 3 usando diccionarios en `Python` (a) y archivos `XML` (b).

Como ya se ha comentado en la Sección 5.1.1 una decisión crucial fue decidir dónde implementar los perfiles temporales: directamente desde `Python` o integrarlos en `C++`. En una primera instancia, realizarlo todo directamente en `Python` era más conveniente. Sin embargo, el lector de escenas en formato `XML` solo reconoce componentes nativos de `C++`. Por lo tanto, una implementación en `C++` permite que los perfiles temporales puedan declararse directamente como etiquetas dentro del archivo `XML` de la escena, lo que no sería posible con una implementación en `Python`, donde cada perfil tendría que definirse como un conjunto de atributos separados y sin conexión estructural.

Esta diferencia en la forma de definir los perfiles se muestra con un ejemplo en la Figura B.2.

```xml
<bsdf>
    ...
    <string name="TempProf" value="type" />
    <float name="Prop1" value="x" />
    <float name="Prop2" value="y" />
    ...

</bsdf>
```

(a)

```xml
<bsdf>
    ...
    <TempProf type="type">
        <float name="Prop1" value="x">
        <float name="Prop2" value="y">
        ...
    </TempProf>
</bsdf>
```

(b)

Figura B.2: Comparación entre la definición de Perfiles temporales (`TempProf`) con la implementación de estos en `Python` (a) y en `C++` (b).

Por lo tanto, la implementación en `C++` permite una definición más clara y sencilla

para el usuario, lo que condicionó parcialmente a la decisión final de implementar los perfiles temporales en C++.

Ejemplo definición Clorofila

Un ejemplo de cómo se definiría un material transitorio más complejo como es el caso de la Clorofila (Sección 3.4) se puede ver a continuación. Esta definición estructurada es posible gracias a la definición de los perfiles en C++, ya que estos materiales que tienen varios perfiles serían especialmente difíciles de definir con una implementación en Python.

```
<bsdf type="Composed" id="Chlorophyll">
    <float name="weight" value="0.5"/>
    <!-- Green -->
    <bsdf type="Separable" id="bsdf-1">
        <bsdf type="diffuse" id="chlor-green">
            ...
        </bsdf>
        <TemporalProfile type="ConstantTP" name="tp">
            <float name="delay" value="0"/>
        </TemporalProfile>
    </bsdf>
    <!-- Red -->
    <bsdf type="Separable" id="bsdf-2">
        <bsdf type="diffuse" id="chlor-red">
            ...
        </bsdf>
        <TemporalProfile type="ConstantTP" name="tp">
            <float name="delay" value="3"/>
        </TemporalProfile>
    </bsdf>
</bsdf>
```

Figura B.3: Esquema XML para definir la Clorofila utilizando ComposedBSDF. En $t = 0$, hay un pulso de color difuso verde y en el tiempo $t = t_0 = 3$ otro de color difuso rojo.

B.2. Algoritmo de Path-Tracing en C++

En esta sección se introducen de forma esquemática y en C++ las aportaciones introducidas al conocido algoritmo de Path-Tracing [13] para implementar un integrador transitorio sin retrasos temporales (el implementado en Mitransient [11]) y las modificaciones realizadas en el presente trabajo a este para poder soportar materiales transitorios.

B.2.1. Integrador transitorio sin retrasos temporales

En comparación con un algoritmo de Path-Tracing normal, en el integrador transitorio implementado por Mitransient se requiere calcular el tiempo de vuelo de la luz (en distancia óptica). Para ello, por cada interacción, se suma al tiempo total el tiempo de esa interacción, calculado como el índice de refracción actual multiplicado por la distancia recorrida en esa interacción. Estos cambios se pueden ver como líneas rojas en el Código B.1.

Listing B.1: Contribuciones (lineas rojas) necesarias para implementar un integrado transitorio sin retrasos temporales a partir de un integrador estacionario en C++.

```
#include <tuple>
#include <cmath>
// Interfaz con todas las clases necesarias
#include "Scene.h"

std::tuple<Radiance, Time> createPath(Scene& scene, Ray original_ray) {
    // Radiance is discretized to 3 values (RGB)
    Radiance throughput = {1.0f, 1.0f, 1.0f};
    Radiance result = {0.0f, 0.0f, 0.0f};
    Ray ray = original_ray;

    // Variables para Transient Rendering
    Time time = 0.0f;
    float ior = scene.defaultIOR();

    // Primera intersección
    SurfaceInteraction si = scene.intersect(ray);

    while (true) {
        // Añadir distancia óptica al tiempo total.
        time += si.time_of_flight(ray) * ior;

        // Si es emisor, evaluar la radiancia saliente
        // emitida y terminar el camino.
        if (si.material.type == EMITTER) {
            result = throughput * si.material.emitter->eval(ray);
            break;
        }

        // Si el camino tiene longitud máxima, acabar
        if (ray.length() > scene.maxPathLength()) {
            break;
        }

        // Muestrear nueva dirección y evaluar la BSDF
        auto [outgoing_ray, bsdf_val] =
                si.material.bsdf->sample(ray);

        // Escalar el throughput por la BSDF de la BSDF actual
        throughput *= bsdf_val.value;


        // Actualizar IOR con el del nuevo medio
        ior = bsdf_val.ior_medium_outgoing_ray;
```

```
45
46          // Continuar con la nueva dirección
47          ray = outgoing_ray;
48          si = scene.intersect(ray);
49      }
50
51      return {result, time};
52  }
```

B.2.2. Integrador transitorio con retrasos temporales

A el algoritmo presentado en la sección anterior, para poder implementar los materiales transitorios tal y como se explicó en el Capítulo 5, es necesario, en cada intersección, también muestrear un retraso temporal asociado, utilizando el nuevo método `sample_t`. Una vez obtenido, simplemente se suma al tiempo total. Los cambios se pueden ver como líneas rojas en el Código B.2.

Listing B.2: Contribuciones (lineas rojas) necesarias para implementar un integrado transitorio **con retrasos temporales** a partir del presentado en B.1.

```
1   #include <tuple>
2   #include <cmath>
3   // Interfaz con todas las clases necesarias
4   #include "Scene.h"
5
6   std::tuple<Radiance, Time> createPath(Scene& scene, Ray original_ray) {
7       // Radiance is discretized to 3 values (RGB)
8       Radiance throughput = {1.0f, 1.0f, 1.0f};
9       Radiance result = {0.0f, 0.0f, 0.0f};
10      Ray ray = original_ray;
11
12      // Variables para Transient Rendering
13      Time time = 0.0f;
14      float ior = scene.defaultIOR();
15
16      // Primera intersección
17      SurfaceInteraction si = scene.intersect(ray);
18
19      while (true) {
20          // Añadir distancia óptica al tiempo total.
21          time += si.time_of_flight(ray) * ior;
22
23          // Si es emisor, evaluar la radiancia saliente
24          // emitida y terminar el camino.
25          if (si.material.type == EMITTER) {
26              result = throughput * si.material.emitter->eval(ray);
27              break;
28          }
29
30          // Si el camino tiene longitud máxima, acabar
31          if (ray.length() > scene.maxPathLength()) {
32              break;
33          }
34
```

```
        // Muestrear nueva dirección, retraso temporal y evaluar la BSDF
        auto [outgoing_ray, bsdf_val, delay] =
                si.material.bsdf->sample_t(ray);

        // Escalar el throughput por la BSDF de la BSDF actual
        throughput *= bsdf_val.value;

        // Añadir retraso temporal al tiempo total
        time += delay;

        // Actualizar IOR con el del nuevo medio
        ior = bsdf_val.ior_medium_outgoing_ray;

        // Continuar con la nueva dirección
        ray = outgoing_ray;
        si = scene.intersect(ray);
    }

    return {result, time};
}
```

Apéndice C

Planificación Temporal

Este proyecto fue planificado originalmente para comenzar al inicio del segundo semestre del año académico 2024/25, es decir, a principios de 2025. Debido a la carga de trabajo del autor durante ese semestre, se estableció como fecha de entrega el mes de septiembre del mismo año. Desde el inicio se definió una secuencia de pasos necesarios para su desarrollo, los cuales se detallan a continuación:

- Investigación sobre renderizado transitorio.
- Estudio de las tecnologías base: Mitsuba y Mitransient.
- Desarrollo de una teoría inicial para implementar materiales transitorios.
- Implementación de dichos materiales.
- Validación visual y analítica de los resultados.

Para organizar el trabajo, se programó una reunión semanal de una hora para discutir los avances, ajustando la frecuencia según las necesidades. El proyecto comenzó con una fase de documentación intensiva sobre los conceptos clave y las tecnologías implicadas, planeada para durar algunas semanas. No obstante, como ocurre habitualmente en proyectos de software, la documentación fue necesaria durante buena parte del desarrollo, según lo iban requiriendo las circunstancias.

Una vez comprendidas las bases teóricas y técnicas, se formuló una teoría inicial que permitió abordar el problema. Sin embargo, esta teoría no se desarrolló por completo desde el principio, sino que fue avanzando para definir materiales más complejos.

A partir de ahí, se trabajó en la implementación desde distintos enfoques con el objetivo de cumplir los requisitos definidos al inicio. Este proceso fue iterativo: a medida que se profundizaba en la documentación, surgían alternativas que mejoraban las soluciones previas. Esto implicó no solo trabajo técnico, sino también un análisis constante de ventajas y desventajas de cada enfoque.

Durante las distintas etapas de implementación, se realizó una validación continua tanto visual como analítica de los resultados obtenidos.

Finalmente, se comenzó a redactar la memoria, una tarea que resultó más exigente de lo esperado. Esto se debió a la gran cantidad de conceptos previos necesarios para comprender la implementación realizada y a la complejidad que esta puede representar para lectores sin formación en informática gráfica. Por este motivo, se iteró múltiples veces la memoria para hacerla intuitiva y entendible, realizando múltiples figuras para simplificar conceptos que pueden resultar complejos.

Cabe señalar que durante el desarrollo se tomaron dos pausas: una por la realización de exámenes finales y otra por vacaciones personales. Esto se representa como franjas grises en el cronograma presentado.

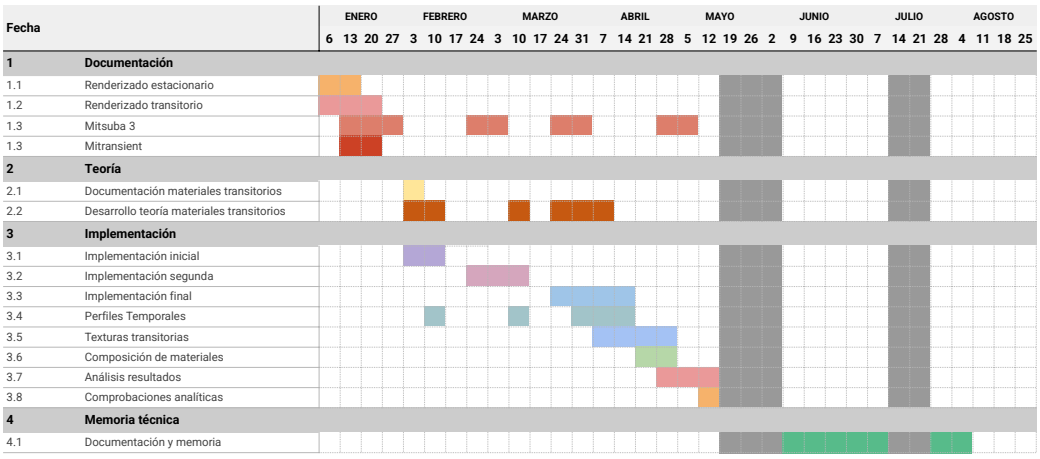


Figura C.1: Cronograma de las fases del proyecto.