UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Codificación de algoritmos de trazado de rayos en Vulkan

Informe de Proyecto de Grado presentado por

Joaquin Bartaburu Chaine, Lucas Andrés Ubal Santana

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisor

Eduardo Fernández

Montevideo, 2 de junio de 2026

Resumen

El trazado de rayos es una técnica ampliamente utilizada para la generación de imágenes realistas en gráficos por computadora, debido a su capacidad para simular de forma precisa algunos fenómenos de iluminación. Por otra parte, su elevado costo computacional ha limitado históricamente su uso en aplicaciones interactivas o en tiempo real.

En los últimos años, la incorporación de hardware especializado para la aceleración del trazado de rayos en GPUs de uso general ha permitido ampliar el uso de estas técnicas, dando lugar a enfoques híbridos que combinan métodos tradicionales de rasterización con algoritmos de trazado de rayos. Para facilitar este tipo de aplicaciones, las principales APIs gráficas modernas, como Vulkan y DirectX, incorporan soporte específico para la ejecución de algoritmos de trazado de rayos sobre GPU.

En este contexto, el presente proyecto se centra en la implementación y evaluación de distintas técnicas de trazado de caminos ejecutadas íntegramente sobre GPU, utilizando extensiones especializadas de Vulkan. Se implementaron tres algoritmos representativos y se evaluaron sobre un conjunto de escenas de prueba y sobre dos GPU de distintos fabricantes y arquitecturas, analizando su comportamiento en términos de convergencia, ruido y desempeño computacional.

Los resultados obtenidos permiten concluir que no hay un algoritmo ganador, sino que depende del hardware disponible y de la configuración de la escena a renderizar, tanto en su geometría como en su iluminación. De este modo, el trabajo aporta una visión práctica sobre las ventajas y limitaciones de cada enfoque, y ejemplifica algunos aspectos relativos a la implementación de algoritmos de transporte de luz en diferentes GPUs.

El software generado se encuentra en: https://github.com/StrikerMF4/vkraytracing

Palabras clave: vulkan, ray tracing, path tracing, bidirectional path tracing, next event estimation, Disney BRDF, BSDF, global illumination, GPU, NVIDIA, AMD

Índice general

Capítulo 1

Introducción

1.1. Motivación y objetivos

El trazado de rayos (ray tracing) es una de las técnicas más utilizadas en la simulación del transporte de la luz, ya que permite generar imágenes con un alto grado de realismo físico. Sin embargo, su elevado costo computacional ha limitado históricamente su aplicación en entornos interactivos o en tiempo real. Con la aparición de hardware especializado para la aceleración del trazado de rayos y las extensiones específicas incluidas en APIs modernas como Vulkan, se abrió la posibilidad de desarrollar implementaciones más eficientes y flexibles, que permiten explorar y comparar diferentes técnicas de iluminación global.

Este proyecto surge con el objetivo de implementar y analizar distintas técnicas de trazado de rayos: trazado de caminos inverso (Backward Path Tracing), trazado de caminos inverso con estimador del siguiente evento (Backward Path Tracing con Next Event Estimation) y trazado de caminos bidireccional (Bidirectional Path Tracing) aprovechando las capacidades de cómputo paralelo de las GPU modernas.

Se busca evaluar los resultados obtenidos con cada técnica en términos de velocidad de convergencia, nivel de ruido y rendimiento, utilizando un modelo de materiales microfacetados con reflexión y transmisión de la luz.

El trabajo se apoya en la Pasantia de Iniciación a la Investigación (PEDECIBA Informática) de Joaquín Fontana (2024), extendiéndolo para cumplir con los objetivos específicos de este proyecto.

Además del análisis de las técnicas de renderizado, este Proyecto de Grado busca evaluar el uso de Vulkan como plataforma de desarrollo, identificando las principales dificultades y ventajas que presenta para la implementación de algoritmos de trazado de rayos complejos en GPU.

Los objetivos específicos del proyecto son:

- Implementar tres algoritmos de iluminación global utilizando las extensiones de trazado de rayos de Vulkan.

- Diseñar una estructura modular que permita intercambiar las técnicas en tiempo de ejecución, facilitando su comparación visual y de rendimiento.
- Desarrollar un modelo de materiales microfacetado, asegurando la compatibilidad con distintos tipos de superficie.
- Analizar la calidad visual, el tiempo de convergencia y la eficiencia de cada técnica mediante escenas de prueba controladas.
- Comparar los resultados obtenidos con implementaciones de referencia, como Mitsuba, para validar la corrección de las soluciones implementadas.
- Comparar los rendimientos de diferentes arquitecturas de GPU de diferentes empresas (NVIDIA y AMD) al ejecutar los algoritmos implementados.

1.2. Organización del documento

El resto de este documento se organiza en cinco capítulos, que describen las diferentes etapas del trabajo realizado.

En el Capítulo 2 se presenta la **Revisión de antecedentes**, que incluye los conceptos fundamentales necesarios para comprender el problema del transporte de luz. Se introducen las bases físicas de la radiometría, la Ecuación de Renderizado que cuantifica los principales aspectos de la iluminación global, los principales métodos de integración mediante Monte Carlo y las formulaciones que dieron origen a las técnicas implementadas. A su vez, se comenta el funcionamiento del ducto de trazado de rayos de Vulkan.

En el Capítulo 3 se describe el **Diseño** del sistema, detallando la estructura general de la aplicación, la organización de los *shaders*, y las decisiones de diseño adoptadas para permitir la comparación entre las diferentes técnicas de renderizado. También se presentan los mecanismos implementados para la gestión de escenas y el intercambio de técnicas en tiempo de ejecución.

En el Capítulo 4 se aborda la **Implementación**, explicando los aspectos técnicos más relevantes del desarrollo. Se detalla la integración de las extensiones de trazado de rayos de Vulkan, la estructura de los ductos, y las particularidades de cada técnica implementada.

El Capítulo 5 presenta la **Experimentación**, incluyendo los casos de prueba diseñados para evaluar las distintas técnicas. Se analizan los resultados obtenidos en términos de rendimiento, ruido y velocidad de convergencia, comparando las salidas visuales y los tiempos de renderizado de cada método. A su vez, se compara el rendimiento del algoritmo al ejecutarse en tarjetas gráficas de los proveedores NVIDIA y AMD.

Finalmente, en el Capítulo 6 se exponen las **Conclusiones y trabajo futuro**, donde se discuten los resultados alcanzados, las dificultades encontradas durante el desarrollo y las posibles líneas de mejora y extensión del proyecto.

Capítulo 2

Revisión de antecedentes

En este capítulo se presentan los fundamentos teóricos y técnicos necesarios para el desarrollo de este trabajo. Se comenzará describiendo el modelado físico de la luz y su interacción con los materiales, para luego profundizar en los modelos de iluminación global y la ecuación de transporte. Además, se detallarán las técnicas de trazado de rayos implementadas, específicamente trazado de caminos y trazado de caminos bidireccional. Por último, se explicara la arquitectura de Vulkan y su ducto para el trazado de rayos acelerado por hardware.

2.1. Modelado de la luz

Para comprender cómo interactúa la luz con las superficies, primero se deben definir algunos conceptos fundamentales.

2.1.1. Radiometría

Para simular el transporte de la luz, primero es necesario definir las magnitudes físicas que permiten cuantificarla. La radiometría es la rama de la óptica que se ocupa de la medición de la radiación electromagnética, incluida la luz visible. En el ámbito de la computación gráfica, dos de sus magnitudes fundamentales son la radiancia y la irradiancia, cuyas definiciones y funciones resultan esenciales para modelar con precisión el comportamiento de la luz.

- **Radiancia** $L(x, \omega)$: se define como la potencia por unidad de ángulo sólido, por unidad de área proyectada sobre el plano perpendicular a la dirección de la radiación (ver Figura 2.1) (Boyd, 1983).

  La propiedad más importante de la radiancia y la razón de su centralidad en los algoritmos de transporte es que es constante a lo largo de un rayo de luz en el vacío (Pharr y cols., 2023). Esto significa que, si se conoce la radiancia que sale de un punto en una superficie, se conoce la radiancia que llegará a otro punto si no hay nada en el camino.

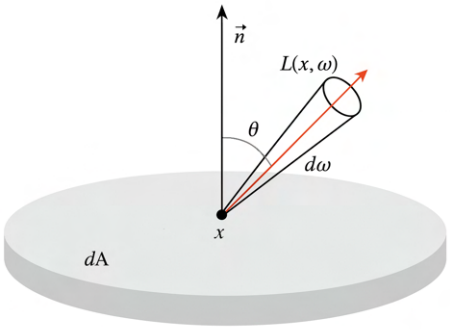


Figura 2.1: Radiancia saliente $L(x, \omega)$.

- **Irradiancia** $E(\mathbf{x})$: representa la potencia total de luz que incide sobre un punto x de una superficie por unidad de área (Boyd, 1983). La irradiancia es el efecto acumulativo de la radiancia que llega desde todas las direcciones (ver Figura 2.2).

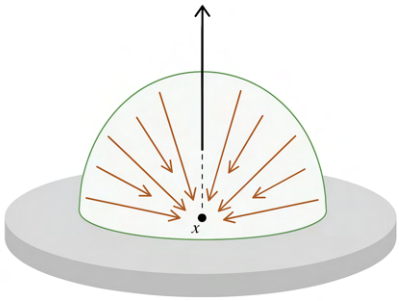


Figura 2.2: Irradiancia $E(x)$ como suma de la radiancia incidente desde todas las direcciones.

La relación entre estas dos magnitudes es crucial: la radiancia es la causa, una propiedad del campo de luz, mientras que la irradiancia es el efecto, lo que una superficie recibe en un punto (Pharr y cols., 2023). La interacción de la luz con un material puede, por tanto, ser conceptualizada como un proceso que transforma la irradiancia incidente en radiancia saliente.

2.1.2. Reflexión y Refracción

Cuando la luz incide sobre una superficie que separa dos medios ópticamente diferentes, su comportamiento está regido por principios físicos claramente

establecidos: la reflexión y la refracción.

La reflexión se produce cuando la radiación electromagnética (luz) incide sobre una superficie y es devuelta al medio original. Este fenómeno se rige por la Ley de la Reflexión (Hecht, 2002, p. 105-106), que establece que el ángulo de incidencia (θ_i) y el ángulo de reflexión (θ_r) son iguales cuando se miden respecto a la normal de la superficie:

$$\theta_i = \theta_r \tag{2.1}$$

Es importante destacar que la Ley de la Reflexión describe un comportamiento ideal que solo se cumple estrictamente en superficies perfectamente lisas. En materiales reales, incluso una superficie que aparenta ser lisa presenta irregularidades microscópicas (las denominadas microfacetas) cuya orientación depende del grado de rugosidad del material, como se ilustra en la Figura 2.3. La Ley de la Reflexión continúa siendo válida a nivel de cada microfaceta individual; sin embargo, la distribución de sus orientaciones provoca que, a escala macroscópica, la energía reflejada se disperse en un rango de direcciones en lugar de concentrarse únicamente en la dirección especular ideal.

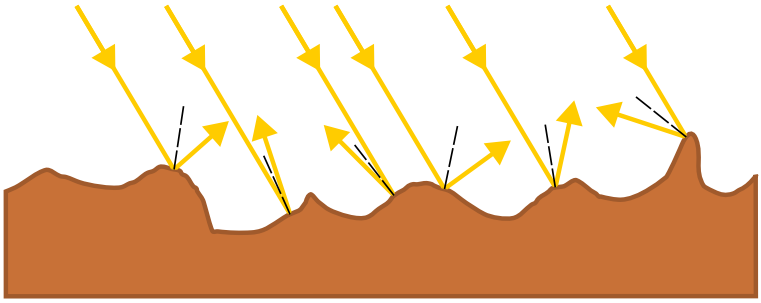


Figura 2.3: Efecto de la luz al incidir sobre una superficie rugosa. Aunque los rayos de luz ingresan paralelos entre sí, la presencia de microfacetas con distintas orientaciones provoca que la luz se refleje en múltiples direcciones.

La **refracción**, por otra parte, consiste en el cambio de dirección que experimenta la luz cuando atraviesa una superficie entre dos medios con diferentes índices de refracción (IOR, por sus siglas en inglés). Este fenómeno se describe mediante la Ley de Snell (Hecht, 2002, p. 109), expresada por la ecuación (ver Figura 2.4):

$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \tag{2.2}$$

donde n_1 y n_2 representan los índices de refracción del medio incidente y del medio transmisor respectivamente, mientras que θ_1 y θ_2 corresponden a los ángulos formados por el rayo incidente y refractado respecto a la normal de la superficie.

Cuando un rayo de luz se propaga por un medio con índice de refracción n_1 y llega a la superficie que lo separa de un segundo medio de índice n_2, con

$n_1 > n_2$, puede reflejarse por completo dentro del medio de mayor índice. Este fenómeno se conoce como reflexión interna total.

La reflexión interna total ocurre únicamente cuando el ángulo de incidencia θ_1 supera un valor umbral denominado ángulo crítico, definido por:

$$\theta_c = \arcsin \frac{n_2}{n_1}$$

- Si $\theta_1 < \theta_c$, parte de la luz se refracta y parte se refleja.
- Si $\theta_1 = \theta_c$, el rayo refractado se propaga paralelo a la superficie.
- Si $\theta_1 > \theta_c$, ocurre reflexión interna total y ningún rayo atraviesa el límite.

Estos comportamientos se pueden visualizar en la Figura 2.4, donde se muestran los diferentes casos de incidencia de un rayo de luz al pasar de un medio de índice n_1 a otro de índice n_2.

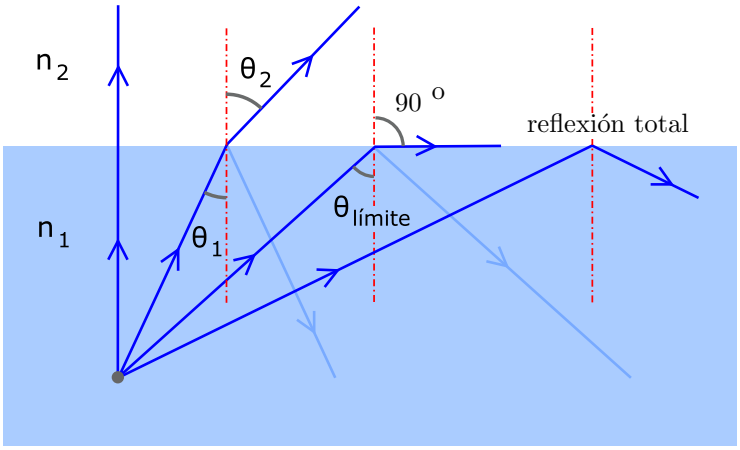


Figura 2.4: Comportamiento de la luz al pasar de un medio a otro para distintos ángulos de incidencia, ilustrando refracción y reflexión interna total (Melenero, 2009)

Ecuaciones de Fresnel

La cantidad de luz que se refleja versus la que se refracta en una interfaz no es constante, sino que depende del ángulo de incidencia y de los índices de refracción de los medios. Esta relación se describe mediante las Ecuaciones de Fresnel (Hecht, 2002).

Para una onda de luz no polarizada, la reflectancia total F_r se obtiene como el promedio de las reflectancias correspondientes a las polarizaciones perpendicular ($|r_s|^2$) y paralela ($|r_p|^2$). Los coeficientes de reflexión en amplitud se definen como:

$$r_s = \frac{n_1 \cos\theta_1 - n_2 \cos\theta_2}{n_1 \cos\theta_1 + n_2 \cos\theta_2} \tag{2.3}$$

$$r_p = \frac{n_2 \cos\theta_1 - n_1 \cos\theta_2}{n_2 \cos\theta_1 + n_1 \cos\theta_2} \tag{2.4}$$

y la reflectancia total se calcula como:

$$F_r = \frac{1}{2}(|r_s|^2 + |r_p|^2) \tag{2.5}$$

donde θ_1 es el ángulo de incidencia, n_1 y n_2 son los índices de refracción de los medios incidente y transmisor respectivamente, y θ_2 es el ángulo de refracción obtenido a partir de la ley de Snell.

Debido al costo computacional de evaluar estas ecuaciones completas en tiempo real (especialmente por la raíz cuadrada implícita en el cálculo de $\cos\theta_t$), es común utilizar la aproximación de Schlick (1994). Esta aproximación estima el factor de Fresnel $F(\theta)$ basándose en la reflectancia bajo incidencia normal (F_0):

$$F(\theta_1) \approx F_0 + (1 - F_0)(1 - \cos\theta_1)^5 \tag{2.6}$$

donde F_0 para materiales dieléctricos se calcula como:

$$F_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2 \tag{2.7}$$

Esta aproximación ofrece un buen equilibrio entre eficiencia computacional y fidelidad visual, por lo que se utiliza ampliamente en motores de renderizado modernos.

2.1.3. Funciones de Distribución Reflectante y Transmisiva (BSDF)

Las leyes de reflexión y refracción presentadas anteriormente son deterministas y válidas únicamente para superficies perfectamente lisas. Sin embargo, en la práctica, las superficies reales presentan características más complejas, como rugosidad, translucidez o irregularidades en su estructura. Para modelar adecuadamente el comportamiento de la luz en estos casos, se introduce un enfoque estadístico mediante la Función de Distribución de Dispersión Bidireccional (BSDF, por sus siglas en inglés).

La función de distribución BSDF, denotada como $f_s(\omega_i, \omega_o)$, se define como la relación entre la radiancia diferencial saliente ($dL_o(\omega_o)$) en una dirección ω_o y la irradiancia diferencial incidente ($dE_i(\omega_i)$) desde una dirección ω_i (Veach, 1997, eq. 3.11):

$$f_s(\omega_i, \omega_o) = \frac{dL_o(\omega_o)}{dE_i(\omega_i)} = \frac{dL_o(\omega_o)}{L_i(\omega_i)\cos\theta_i d\omega_i} \tag{2.8}$$

En otras palabras, el BSDF indica qué fracción de la luz incidente desde una dirección ω_i es dispersada hacia la dirección ω_o.

Comúnmente, el BSDF se separa en dos componentes:

- Función de Distribución de Reflectancia Bidireccional (BRDF): f_r, describe la distribución de luz reflejada, donde ω_i y ω_o pertenecen al mismo hemisferio respecto a la superficie.

- Función de Distribución de Transmitancia Bidireccional (BTDF): f_t, describe la distribución de luz transmitida, donde ω_i y ω_o se encuentran en hemisferios opuestos respecto a la superficie.

Al integrar esta definición sobre todas las posibles direcciones incidentes, se obtiene la ecuación de dispersión (*scattering equation*), que predice la radiancia total saliente desde un punto en la superficie (Veach, 1997, eq. 3.12):

$$L_o(\omega_o) = \int_{s^2} L_i(\omega_i) f_s(\omega_i \to \omega_o) \cos\theta_i d\omega_i$$

Esta fórmula es la base de todos los métodos de trazado de rayos y Monte Carlo en iluminación global.

Modelos de Microfacetas

La mayoría de las superficies del mundo real no son perfectamente lisas. La teoría de microfacetas es el modelo físico predominante para explicar la apariencia de superficies rugosas, como los metales cepillados o los plásticos mate. La idea central es que una superficie macroscópicamente rugosa está compuesta por una infinidad de pequeños espejos microscópicos, llamados microfacetas, donde cada uno de ellos refleja la luz de forma perfectamente especular. La apariencia global de la superficie es el resultado estadístico de las reflexiones de todas estas microfacetas.

La formulación clásica más conocida de estos modelos es la propuesta por Cook y Torrance, expresada mediante la siguiente ecuación (Walter y cols., 2007):

$$f_r(\omega_i, \omega_o) = \frac{D(h) \cdot F(\omega_i, h) \cdot G(\omega_i, \omega_o)}{4(\mathbf{n} \cdot \omega_i)(\mathbf{n} \cdot \omega_o)} \tag{2.9}$$

donde:

- $D(h)$ es la **Función de Distribución de Normales** (NDF), que describe la distribución estadística de las normales de las microfacetas. Una distribución concentrada produce un reflejo nítido (superficie lisa), mientras que una distribución amplia produce un reflejo borroso (superficie rugosa). El vector h (halfway vector) biseca el ángulo entre la dirección de incidencia ω_i y la dirección de observación ω_o. Solo las microfacetas cuya normal se alinea con h pueden reflejar la luz desde ω_i hacia ω_o.

- $G(\omega_i, \omega_o)$ es la **Función de Geometría**, que representa la probabilidad de que las microfacetas no se bloqueen mutuamente. Este término incluye efectos de ensombrecimiento (shadowing) y enmascaramiento (masking).

- $F(\omega_i, h)$ es la **Función de Fresnel**, que determina la cantidad de luz que es reflejada por cada microfaceta individual, basándose en las ecuaciones de Fresnel. Este término es crucial para diferenciar entre materiales metálicos y dieléctricos.
- El denominador $4(\mathbf{n} \cdot \omega_i)(\mathbf{n} \cdot \omega_o)$es un **factor de corrección** que relaciona el área efectiva de las microfacetas con el área macroscópica observada.

Para asegurar la coherencia con la física de la luz y el realismo en el sombreado de materiales, un BRDF debe adherirse a varias propiedades matemáticas fundamentales. Primero, el BRDF debe ser siempre no negativo, lo que significa que el valor de la reflectancia nunca puede ser inferior a cero, ya que la luz es una forma de energía que no puede ser negativa. Segundo, debe exhibir reciprocidad de Helmholtz, un principio que establece que el camino de la luz es reversible; es decir, la cantidad de luz reflejada es la misma si se intercambian las direcciones de la luz incidente y la de observación. Esto garantiza la coherencia visual de los materiales desde diferentes perspectivas.

Crucialmente, para la conservación de la energía, la integral de la BRDF ponderada por el coseno sobre todas las posibles direcciones de salida (representando la energía total reflejada) debe ser menor o igual a 1. La integral nunca debe ser mayor a 1, ya que esto implicaría la creación de energía en la superficie, conduciendo a artefactos poco realistas como materiales que emiten un brillo inorgánico o que son más brillantes que la fuente de luz.

El Modelo de Materiales de Disney

Continuando con los fundamentos de los BSDFs, el modelo de materiales implementado en este proyecto se basa en el modelo de BRDF de Disney (Burley, 2012). Este modelo empírico fue diseñado teniendo en cuenta los principios de coherencia con la física de la luz mencionados, pero orientado a lograr un equilibrio óptimo entre precisión física y flexibilidad artística, siendo capaz de describir la gran mayoría de los materiales utilizados en una producción con un único conjunto de parámetros intuitivos. La filosofía de este modelo se basa en la teoría de microfacetas, pero introduce modificaciones y adiciones empíricas para lograr un mayor control artístico y reproducir efectos observados en mediciones de materiales reales que los modelos simples no capturan.

Una característica fundamental de este modelo es su enfoque en la coherencia con la física de la luz más que en la exactitud estricta. Si bien se basa en principios físicos y datos medidos, su prioridad es la expresión artística y la facilidad de uso para los artistas gráficos en la producción de efectos visuales (Burley, 2012, p. 12). Esto representa una filosofía de diseño crucial en el renderizado de producción: aunque los modelos basados en la física son deseables para el realismo, las necesidades prácticas a menudo exigen ajustes empíricos para proporcionar controles intuitivos a los artistas. Esto significa que "basado en la física" no siempre implica "perfectamente preciso físicamente", sino más bien "coherente con la física de la luz" y amigable para el artista.

Sus características principales son:

- Utiliza la distribución **GGX (Trowbridge-Reitz**) para la Función de Distribución de Normales (NDF) en el término especular. Esta distribución proporciona reflejos especulares más realistas, especialmente en ángulos rasantes, en lugar de otras distribuciones como Beckmann.

- **Mezcla Metálico/Dieléctrico**: En lugar de tener shaders separados, el modelo se controla con un único parámetro `metallic`. Un valor de 0 lo hace puramente dieléctrico (no metálico, como plástico o cristal), mientras que un valor de 1 lo hace puramente metálico. Valores intermedios permiten mezclas. Esto simplifica enormemente el flujo de trabajo de los artistas.

- **Componente Difuso No-Lambertiano**: El modelo difuso Lambertiano es una simplificación que asume que una superficie dispersa la luz de manera uniforme en todas las direcciones, sin importar el ángulo de la luz o del observador. Es decir, una superficie puramente Lambertiana se vería con el mismo brillo desde cualquier punto de vista. El modelo difuso de Disney es no-Lambertiano para lograr un mayor realismo. Incorpora dos efectos que modulan cómo se dispersa la luz difusa:

  - **Retro-reflexión en ángulos rasantes:** Añade un término que hace que la superficie parezca más brillante cuando la luz incide y se observa en ángulos muy oblicuos, reflejándose hacia atrás. Esto simula el comportamiento de materiales rugosos donde la luz se dispersa más intensamente hacia la fuente.

  - **Oscurecimiento de Fresnel:** En ángulos rasantes, una mayor parte de la luz se refleja especularmente en la superficie (según las leyes de Fresnel), dejando menos luz para penetrar y dispersarse difusamente dentro del material. Esto resulta en una atenuación de la componente difusa en esos ángulos.

- **Componentes Adicionales**: Introduce términos opcionales para efectos específicos:

  - **Sheen**: Un brillo adicional en ángulos rasantes, útil para simular la apariencia de telas como el terciopelo. Se controla con el parámetro `sheen` y su tinte con `sheenTint`.
  - **Clearcoat**: Una segunda capa especular, isotrópica y siempre encima de las demás capas. Simula un barniz o una capa de laca sobre un material base. Se controla con los parámetros `clearcoat` y `clearcoatGloss`.

Lo bueno del modelo de Disney reside en su conjunto de parámetros intuitivos, que permiten a los artistas pensar en términos de propiedades físicas familiares en lugar de coeficientes abstractos o complejos. En la Figura 2.5 se muestra el efecto de estos parámetros.

Posteriormente, Burley (2015) extendió este modelo original (BRDF) hacia un modelo unificado de BSDF. La motivación principal fue integrar la dispersión subsuperficial y la refracción dentro de un mismo marco de trazado de caminos.

Esta extensión introduce el concepto de Transmisión Especular, que permite modelar superficies refractivas como el vidrio o el agua. En este modelo unificado, la energía se distribuye entre los lóbulos de reflexión (difusa y especular) y un lóbulo de refracción basado en la física, determinado por el índice de refracción (IOR). A diferencia de las aproximaciones anteriores, el modelo de 2015 gestiona explícitamente la probabilidad de que un rayo sea reflejado o transmitido a través de la superficie, permitiendo simular materiales complejos que poseen tanto características metálicas, dieléctricas y transmisivas de forma simultánea.

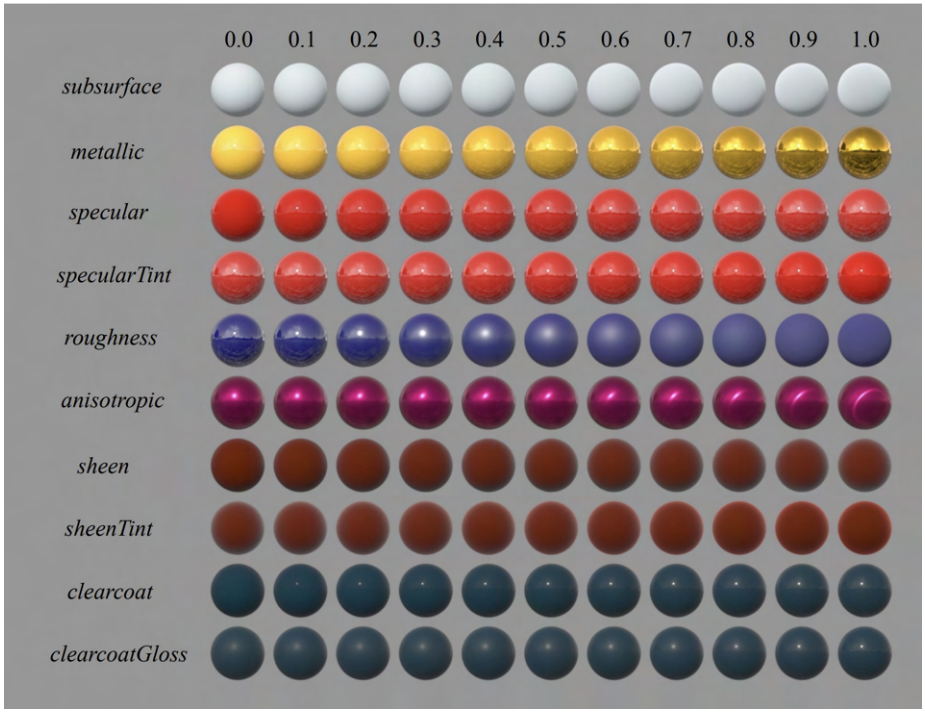


Figura 2.5: Ejemplos del efecto de los parámetros del BRDF de Disney. Cada parámetro varía de 0 a 1, manteniendo constantes el resto (Burley, 2012).

2.2. Modelos de iluminación global

Mucha de la luz que percibimos no proviene directamente de una fuente luminosa, sino que ha sido reflejada o transmitida múltiples veces por las superficies de la escena antes de llegar a nuestros ojos. Por ejemplo, en una habitación iluminada por el sol, solo algunas superficies son alcanzadas directamente por

los rayos solares, mientras que el resto se ilumina gracias a la luz reflejada por las paredes, el piso y los objetos. Este fenómeno se conoce como iluminación global.

La simulación de la iluminación global implica modelar cómo la luz se propaga y se intercambia entre las superficies de la escena, considerando tanto la emisión directa como la radiancia reflejada, refractada o dispersada. Para formalizar este proceso, se han propuesto diferentes ecuaciones, desde formulaciones físicas generales hasta versiones específicas adaptadas al renderizado computacional.

2.2.1. Ecuación de Transporte de la Luz

La Ecuación de Transporte de la Luz (LTE) es la formulación matemática fundamental que describe el flujo de energía luminosa en un entorno, y por lo tanto, es fundamental para la estimación de la iluminación global de una escena. Cuantifica la radiancia total saliente de un punto en una superficie en función de su propia emisión, las propiedades de dispersión del material (BSDF) y la iluminación que incide sobre él desde todas las demás direcciones de la escena.

$$L(x, \omega_o) = L_e(x, \omega_o) + \int_{S^2} L(x_{\mathcal{M}}(x, \omega_i), -\omega_i)\, f_s(x, \omega_i \to \omega_o)\, d\sigma_x^{\perp}(\omega_i) \quad (2.10)$$

donde:

- $x_{\mathcal{M}}(x, \omega_i)$ denota el punto de la escena obtenido al trazar un rayo desde x en la dirección ω_i, es decir, la primera intersección con la geometría en esa dirección.
- $L(x, \omega_o)$ es la **radiancia saliente** del punto x en dirección ω_o.
- $L_e(x, \omega_o)$ es la **radiancia emitida** directamente por el punto x en dirección ω_o.
- $L(x_{\mathcal{M}}(x, \omega_i), -\omega_i)$ es la **radiancia incidente** en el punto x proveniente de la dirección ω_i.
- $f_s(x, \omega_i \to \omega_o)$ es el BSDF del material en el punto x, que describe cómo la luz incidente desde ω_i se dispersa hacia ω_o.
- $d\sigma_x^{\perp}(\omega_i)$ es el término que ajusta la irradiancia incidente por el ángulo, mostrando cómo la "energía efectiva" de la luz depende de la orientación de la superficie respecto a la dirección de la luz.
- La integral sobre S^2 suma todas las contribuciones de la luz incidente que son reflejadas o transmitidas hacia la dirección ω_o.

La ecuación anterior proviene de la física del transporte de la luz. Pero su resolución directa es compleja debido a la recursividad de la integral, lo que motivó la búsqueda de formulaciones más prácticas en el contexto del renderizado por computadora.

James Kajiya (1986) presentó una versión simplificada y computacionalmente aplicable de la ecuación de transporte, conocida como la ecuación de renderizado. Esta formulación asume un medio no participante, es decir, que la luz interactúa únicamente con las superficies y no con el volumen del espacio que las separa, modelando el flujo de radiancia exclusivamente entre puntos sobre las superficies de la escena. Dado que la ecuación es recursiva e integra sobre un dominio de alta dimensión, Kajiya propuso resolverla mediante métodos Monte Carlo, dando origen al algoritmo de trazado de caminos inverso.

2.2.2. Formulación en tres puntos de la ecuación de transporte

En su tesis doctoral, Eric Veach (1997) propuso una reformulación global de la ecuación de transporte, conocida como la Formulación en Tres Puntos. Esta versión generaliza las formulaciones previas, incluyendo la de Kajiya, al eliminar las variables direccionales explícitas en las integrales y expresar el transporte de luz de forma simétrica y puramente espacial entre puntos de la escena. La radiancia se define, entonces, como una función del par de puntos conectados, $L(x \to x')$, lo que permite una representación más compacta y general del flujo de energía en equilibrio.

$$L(x' \to x'') = L_e(x' \to x'') + \int_{\mathcal{M}} L(x \to x') f_s(x \to x' \to x'') G(x \leftrightarrow x') dA(x) \tag{2.11}$$

En esta ecuación, los términos se interpretan de la siguiente manera:

- $L(x \to x')$ y $L_e(x' \to x'')$ representan la radiancia total y emitida (respectivamente) que viaja entre los puntos indicados.
- $f_s(x \to x' \to x'')$ es el BSDF evaluado en el punto central x'. A diferencia de la forma direccional $f_s(x', \omega_i \to \omega_o)$, aquí la dirección incidente ω_i corresponde al vector normalizado desde x hacia x', y la dirección saliente ω_o al vector desde x' hacia x''.

La integral se evalúa sobre el área $A(x)$ de la unión de todas las superficies $\mathcal{M}$ que componen la escena.

El factor $G(x \leftrightarrow x')$ es el componente geométrico y se define de la siguiente forma:

$$G(x \leftrightarrow x') = V(x \leftrightarrow x') \frac{|cos(\theta_o)\ cos(\theta_i')|}{\| x - x' \|^2} \tag{2.12}$$

donde θ_o y θ_i' son los ángulos entre el segmento $x \leftrightarrow x'$ y las normales de las superficies en los puntos x y x' respectivamente (como se ve en la Figura 2.6), mientras que $V(x \leftrightarrow x') = 1$ si x y x' son mutuamente visibles y cero en caso contrario.

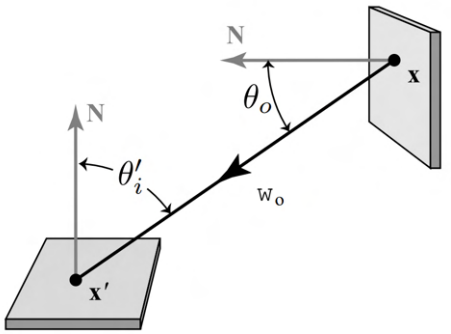


Figura 2.6: Representación de los ángulos y vectores relevantes para evaluar el término geométrico G entre dos puntos. (Veach, 1997)

2.2.3. Monte Carlo

El método de Monte Carlo permite estimar integrales a partir de muestras aleatorias. Sea

$$I = \int_\Omega f(x)\, d\mu(x), \tag{2.13}$$

la integral que se desea aproximar. Si $X_1, \ldots, X_N$ son variables aleatorias independientes e idénticamente distribuidas según una densidad $p(x)$ definida sobre Ω, entonces el estimador de Monte Carlo se define como

$$\hat{I}_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}. \tag{2.14}$$

Este estimador es una variable aleatoria, ya que su valor depende del conjunto de muestras generado. Su varianza está dada por

$$\mathrm{Var}(\hat{I}_N) = \frac{1}{N} \left(\int_\Omega \frac{f(x)^2}{p(x)}\, d\mu(x) - I^2 \right). \tag{2.15}$$

En consecuencia, al aumentar la cantidad de muestras, la dispersión del estimador respecto al valor verdadero de la integral disminuye. En particular,

$$\mathrm{Var}(\hat{I}_N) \to 0 \qquad \text{cuando } N \to \infty, \tag{2.16}$$

por lo que el estimador converge al valor de la integral.

En renderizado, esta propiedad es especialmente relevante, ya que el error de los métodos de Monte Carlo se manifiesta visualmente como ruido y está directamente relacionado con la varianza del estimador. Por este motivo, muchas técnicas de renderizado pueden interpretarse como métodos de reducción de varianza. Entre ellas se encuentran *importance sampling* y *multiple importance sampling* (MIS). El objetivo de estas técnicas no es modificar el valor esperado del estimador, sino reducir su varianza para obtener imágenes menos ruidosas con la misma cantidad de muestras.

2.3. Trazado de Rayos

El trazado de rayos o *ray tracing* es una técnica que consiste en la proyección de rayos para estudiar las intersecciones de estos con una escena. Los rayos equivalen a semirrectas y se suelen representar como un punto de origen y una dirección en la que se proyectan.

Un rayo puede expresarse en forma paramétrica según la siguiente ecuación:

$$r(t) = o + d\ t$$

donde o representa el origen del rayo y d la dirección en la que se proyecta. Al variar t obtenemos los distintos puntos incluidos en el recorrido del rayo. Las intersecciones del rayo con la escena se suelen reportar como la distancia t hasta el punto o. Los valores negativos de t suelen ser ignorados, ya que representan puntos en la dirección opuesta a la que se proyecta el rayo.

Este tipo de técnicas se utilizan en diferentes campos para simular múltiples fenómenos que involucran ondas o partículas, por ejemplo, para simular el comportamiento de las ondas de sonido o modelar el comportamiento de la luz.

En el área de la computación gráfica, las técnicas de trazado de rayos se utilizan para la generación de una representación bidimensional (una imagen) a partir de una escena normalmente tridimensional.

En la realidad, el color de cada píxel de la imagen generada por una cámara fotográfica digital se corresponde con la intensidad y la longitud de onda de la luz que alcanza una cierta zona de su sensor. Para generar una imagen utilizando trazado de rayos, se suele considerar un modelo de cámara, para determinar la distribución y ubicación de los píxeles dentro del espacio de la escena; luego, se determina el color de la luz que viaja por la escena hasta cada uno de los píxeles de la imagen. Para ello, se trabaja con los múltiples caminos que sigue la luz desde las fuentes luminosas hasta la cámara.

A continuación, presentaremos una serie de técnicas que utilizan trazado de rayos para la simulación del flujo de la luz que llega a la cámara.

2.3.1. Trazado de caminos hacia adelante (FPT)

Trazado de caminos hacia adelante (también conocido como trazado de luz) simula el viaje que siguen los fotones en la realidad, desde una fuente luminosa (por ejemplo, el sol) hasta el observador. Para esto, se trazan una serie de rayos desde cada una de las fuentes luminosas y se consideran aquellos caminos que alcanzan la cámara, como se ve en la Figura 2.7.

El FPT evalúa la misma integral de la Ecuación de Transporte de la Luz (2.10) pero "desde la emisión hacia el sensor": se generan caminos $x_0 \to \ldots \to x_k$ con x_0 en una luz y se acumula su contribución si el camino interseca la cámara.

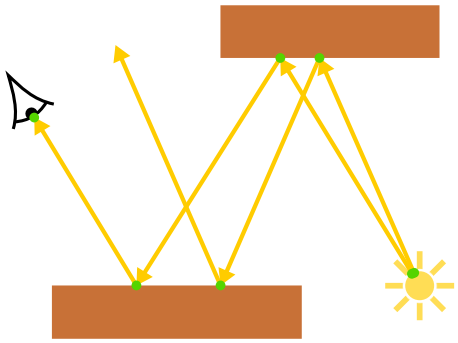


Figura 2.7: Diagrama de los rayos trazados desde la fuente luminosa

Sin embargo, dado que la cámara suele ser una entidad muy pequeña en comparación con el resto de los objetos de la escena, la probabilidad de que un rayo que rebota en una superficie impacte en la cámara es pequeña. A su vez, es posible que muchas fuentes ni siquiera contribuyan a la imagen final por oclusión o distancia.

2.3.2. Trazado de Caminos Inverso (BPT)

El trazado de caminos inverso, también conocido como *backward path tracing* (BPT) o simplemente *path tracing*, genera los rayos primarios (los primeros en ser trazados) desde la cámara y los propaga hacia la escena. Cada rayo interactúa con las superficies mediante reflexión, refracción o difusión, acumulando la contribución de luz en cada rebote. El proceso continúa hasta que el recorrido del rayo alcanza una fuente luminosa o se cumple algún criterio de terminación, como una ruleta rusa o un limite en la profundidad del camino.

La Figura 2.8 ilustra este proceso.

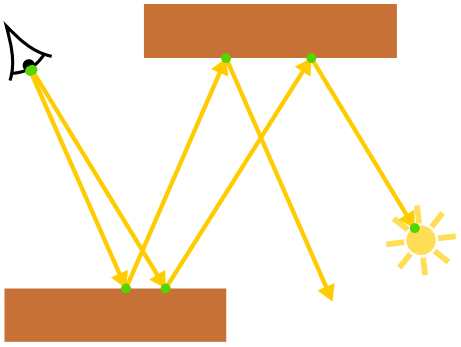


Figura 2.8: Diagrama de los rayos trazados desde la cámara

Este método estima por Monte Carlo la integral de la Ecuación de Transporte de la Luz (2.10), evaluando la radiancia saliente $L_o(x, \omega_o)$ a partir de muestras

direccionales ω_i^k elegidas según una densidad $p(\omega_i^k)$:

$$L_o(x,\omega_o) \;=\; L_e(x,\omega_o) \;+\; \frac{1}{N}\sum_{k=1}^{N} \frac{L_o(x_{\mathcal{M}}(x,\omega_i),-\omega_i^k)\, f_s(x,\omega_i^k \to \omega_o)\,(\omega_i^k \cdot n)}{p(\omega_i^k)}\,, \tag{2.17}$$

A diferencia del FPT, este enfoque concentra el muestreo en los caminos que efectivamente contribuyen a la imagen, ya que parte de los píxeles de la cámara. No obstante, puede presentar alta varianza cuando las fuentes son pequeñas o están ocluidas, debido a la baja probabilidad de que un camino aleatorio desde la cámara las interseque.

2.3.3. Trazado de Caminos Inverso con Estimador del Siguiente Evento (BPT-NEE)

En trazado de caminos inverso (BPT) la luz que llega a la cámara se calcula simulando los rebotes de la luz en las superficies hasta que se alcanza una fuente luminosa. Todos los caminos que no terminan en una fuente son descartados, lo que genera una alta varianza, especialmente cuando las luces son pequeñas o están ocluidas.

Una forma de reducir esta varianza es evaluar explícitamente la iluminación directa en cada punto del camino; para esto, se evalúa la luz que viaja al punto desde las fuentes de luz que son visibles desde el mismo. Esta técnica se conoce como Estimador del Siguiente Evento (Next Event Estimation o NEE). La Figura 2.9 representa este proceso.

Al incluir la iluminación directa en el calculo de la luz se pueden aprovechar todos los caminos generados, aún cuando estos no terminen en una fuente luminosa.

Siguiendo la formulación presentada por Szirmay-Kalos (1999), la radiancia saliente L_o en un punto x puede descomponerse en dos componentes: una correspondiente a la iluminación directa proveniente de las fuentes de luz, y otra a la iluminación indirecta, obtenida de manera recursiva siguiendo los rebotes sucesivos de la luz en las superficies:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_{\text{directa}}(x,\omega_o) + L_{\text{indirecta}}(x,\omega_o) \tag{2.18}$$

En el algoritmo de BPT, la iluminación directa se calcularía tomando un nuevo punto x' en la escena y evaluando $L_o(x',\omega_o')$, siendo ω_o' el vector que apunta a x desde x'.

El término de iluminación directa L_{directa} se obtiene al evaluar la contribución de las fuentes de luz visibles desde x. Siguiendo a Szirmay-Kalos (1999, p. 60), para un punto de la escena x y una muestra y en la superficie de una fuente luminosa, la contribución puede escribirse como:

$$L_{directa}(x,\omega) = \int_{S_e} L_e(y,\omega_{y\to x}) f_s(x,-\omega_{y\to x}\to\omega) G(x\leftrightarrow y) dy \tag{2.19}$$

donde:

- $L_e(y, \omega_i)$ es la radiancia emitida por la fuente en dirección al punto x,
- f_s es la función de dispersión del material en x,
- $G(x \leftrightarrow y)$ es el término geométrico (definido en (2.12)),
- y S_e representa la unión de las superficies de todas las luces en la escena.

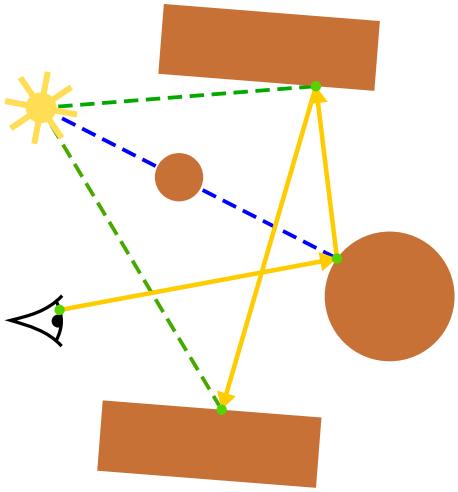


Figura 2.9: Diagrama del trazado de los rayos considerando los rayos de sombra. La luz indirecta está señalizada con la linea naranja, y la directa con la linea punteada verde. La linea punteada azul es un caso donde un objeto obstruye el trayecto de la luz directa a un punto del camino

2.3.4. Trazado de Caminos Bidireccional (BDPT)

El trazado de caminos bidireccional, también conocido como *Bidirectional Path Tracing* (BDPT), fue introducido por Lafortune y Willems (1993) y posteriormente formalizado en detalle por Eric Veach (1997). Esta técnica busca combinar los beneficios del trazado de caminos clásico y de la conexión explícita con las fuentes de luz.

En BDPT, se generan subcaminos tanto desde la cámara como desde las fuentes de luz. Luego, estos subcaminos se combinan de múltiples maneras posibles para formar caminos completos de transporte de luz. Esta estrategia permite mejorar la eficiencia del muestreo y reducir la varianza del estimador.

BDPT resulta especialmente efectivo en escenas con iluminación indirecta compleja o con fuentes de luz difíciles de alcanzar, ya que los subcaminos generados desde las luces permiten capturar contribuciones que un método basado únicamente en caminos iniciados desde la cámara difícilmente encontraría.

Al variar la cantidad de vértices tomados de cada subcamino (desde la luz y desde la cámara) se obtiene una familia de técnicas de muestreo. Fijamos la longitud del camino por número de aristas k (por lo tanto, el camino tiene $k+1$ vértices). Si el subcamino de luz contiene s vértices y el de cámara t, el camino completo tiene $s+t$ vértices y, por lo tanto $k = s+t-1$ aristas. Para una longitud de camino k fija existen $k+2$ técnicas posibles (variando $s = 0, \ldots, k+1$, $t = 0, \ldots, k+1$ siguiendo la ecuación). La Figura 2.10 ilustra las cuatro técnicas de muestreo para caminos de largo $k = 2$.

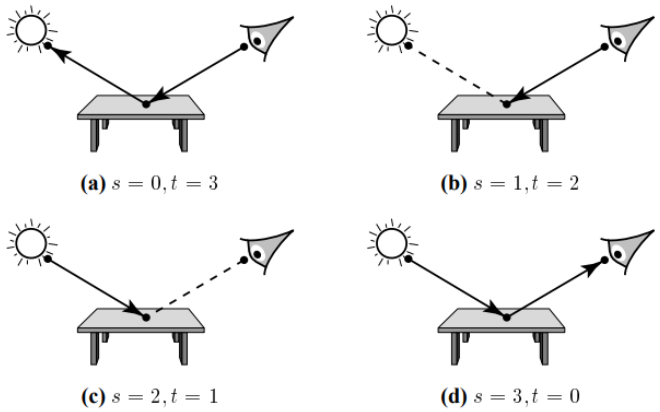


Figura 2.10: En el caso de caminos de longitud $k = 2$, Veach distingue cuatro técnicas de muestreo bidireccional: (a) el trazado de caminos sin un tratamiento especial de las luces (BPT), (b) el trazado con estimación explícita de iluminación directa (BPT con NEE), (c) el trazado de fotones que registra muestras al impactar superficies visibles, y (d) el trazado desde la luz que solo genera muestras al alcanzar la lente de la cámara (FPT).
Cabe señalar que la técnica (a) solo puede aplicarse cuando las luces son áreas emisoras, mientras que la técnica (d) requiere una lente con apertura finita.
Cada una produce una distribución de probabilidad diferente sobre el espacio de caminos, por lo que resultan más adecuadas para distintos tipos de configuraciones. En general, la técnica (b) ofrece buenos resultados en la mayoría de los casos; sin embargo, la (a) puede ser preferible con materiales muy especulares, mientras que las (c) y (d) tienden a presentar menor varianza cuando las fuentes de luz son altamente direccionales. Ver en la tesis de Veach (1997).

Cada una de estas técnicas considera diferentes aspectos y efectos de la iluminación sobre la escena. Para aprovechar este aspecto, BDPT genera muestras de todas las técnicas y las combina utilizando multiple importance sampling (MIS). En particular, se introduce el siguiente estimador (Veach, 1997, p. 300) para cada medida I_j:

$$F = \sum_{s \geq 0} \sum_{t \geq 0} w_{s,t}(\overline{x}_{s,t}) \frac{f(\overline{x}_{s,t})}{p_{s,t}} \tag{2.20}$$

donde $\overline{x}_{s,t}$ es el camino generado según la función de densidad $p_{s,t}$, y las funciones de pesos $w_{s,t}$ representan la estrategia para combinar las muestras de las distintas técnicas (en la Sección 2.3.4 introduciremos un grupo de estrategias a utilizar).

Generación de muestras

Se genera un subcamino desde la luz $y_0 \dots y_{n_{L-1}}$ compuesto por n_L vértices, y un subcamino desde la cámara $z_{n_{E-1}} \dots z_0$ compuesto por n_E vértices, donde y_0 es un punto en una fuente de luz, y z_0 un punto en la lente de la cámara.

Se define $\overline{x}_{s,t}$ como el camino que combina los primeros s vértices del subcamino de la luz con los primeros t del subcamino de la cámara:

$$\overline{x}_{s,t} = y_0, \dots, y_{s-1}, z_{t-1}, \dots, z_0, \quad \text{con} \quad 0 \leq s \leq n_L,\ 0 \leq t \leq n_E.$$

Ya que estos caminos están compuestos por dos subcaminos generados en forma independiente, es necesario comprobar la visibilidad entre los vértices y_{s-1} y z_{t-1}. Si el segmento que los conecta está ocluido, o si el flujo de luz no puede propagarse entre los puntos (por las propiedades de los materiales), la contribución de ese camino será nula.

Calculo de contribuciones

Para calcular la contribución de cada técnica, (Veach, 1997, p. 304-305) se puede derivar la siguiente fórmula a partir del estimador (2.20):

$$F = \sum_{s\geq 0}\sum_{t\geq 0} w_{s,t} C^*_{s,t} = \sum_{s\geq 0}\sum_{t\geq 0} w_{s,t}(\alpha^L_s c_{s,t} \alpha^E_t) \tag{2.21}$$

donde $C^*_{s,t}$ es la contribución no ponderada, que se descompone en tres términos, donde α^L_s depende únicamente del subcamino de la luz, α^E_s depende únicamente del subcamino de la cámara, y $c_{s,t}$ depende de los vértices y_{s-1} y z_{t-1}.

Los términos α^L_i y α^E_i se definen recursivamente como:

$$\begin{aligned}
\alpha^L_0 &= 1, \\
\alpha^L_1 &= \frac{L^{(0)}_e(y_0)}{P_A(y_0)}, \\
\alpha^L_i &= \frac{f_s(y_{i-3} \to y_{i-2} \to y_{i-1})}{P_{\sigma^\perp}(y_{i-2} \to y_{i-1})}\alpha^L_{i-1} \quad \text{para } i \geq 2
\end{aligned} \tag{2.22}$$

$$\begin{aligned}
\alpha^E_0 &= 1, \\
\alpha^E_1 &= \frac{W^{(0)}_e(z_0)}{P_A(z_0)}, \\
\alpha^E_i &= \frac{f_s(z_{i-1} \to z_{i-2} \to z_{i-3})}{P_{\sigma^\perp}(z_{i-2} \to z_{i-1})}\alpha^E_{i-1} \quad \text{para } i \geq 2
\end{aligned} \tag{2.23}$$

donde i es la cantidad de vértices considerados en el subcamino correspondiente. Estos pesos pueden calcularse de manera incremental al momento de generar los subcaminos, almacenando los valores parciales para cada longitud i.

$L_e^{(0)}(y_0)$ y $W_e^{(0)}(z_0)$ representan las componentes espaciales de la emisión y de la respuesta del sensor, respectivamente; es decir, la parte que depende únicamente de la posición inicial del rayo.

$P_{\sigma^\perp}(y_{i-2} \to y_{i-1})$ denota el PDF de muestreo de la dirección $y_{i-2} \to y_{i-1}$ desde el punto y_{i-2}, expresada en la medida de ángulo sólido proyectado.

Para el caso $i = 2$, se siguen las convenciones de Veach (1997, p. 304), definiendo $f_s(y_{-1} \to y_0 \to y_1) = L_e^{(1)}(y_0 \to y_1)$ y $f_s(z_1 \to z_0 \to z_{-1}) = W_e^{(1)}(z_1 \to z_0)$, que corresponden a las componentes direccionales de la emisión y de la respuesta del sensor.

El componente $c_{s,t}$ se define como:

$$\begin{aligned} c_{0,t} &= L_e(z_{t-1} \to z_{t-2}), \\ c_{s,0} &= W_e(y_{s-2} \to y_{s-1}), \\ c_{s,t} &= f_s(y_{s-2} \to y_{s-1} \to z_{t-1}) G(y_{s-1} \leftrightarrow z_{t-1}) f_s(y_{s-1} \to z_{t-1} \to z_{t-2}) \\ &\qquad\qquad \text{para } s, t > 0 \end{aligned} \tag{2.24}$$

Cabe destacar que el término $G(y_{s-1} \leftrightarrow z_{t-1})$ incluye una prueba de visibilidad entre ambos vértices. En el caso $s, t > 0$, esta verificación (que implica lanzar un rayo sombra) es el paso más costoso del cálculo (Veach, 1997, p. 305).

Densidad de probabilidad

Los términos p_i^L y p_i^E representan la probabilidad de generar los primeros i vértices del subcamino de luz y del subcamino de la cámara, respectivamente. Siguiendo a Veach (1997, p. 303):

$$\begin{aligned} p_0^L &= 1, \\ p_1^L &= P_A(y_0), \\ p_i^L &= p_{\sigma^\perp}(y_{i-2} \to y_{i-1}) G(y_{i-2} \leftrightarrow y_{i-1}) p_{i-1}^L \quad \text{para } i \geq 2 \end{aligned} \tag{2.25}$$

$$\begin{aligned} p_0^E &= 1, \\ p_1^E &= P_A(z_0), \\ p_i^E &= p_{\sigma^\perp}(z_{i-2} \to z_{i-1}) G(z_{i-2} \leftrightarrow z_{i-1}) p_{i-1}^E \quad \text{para } i \geq 2 \end{aligned} \tag{2.26}$$

donde $P_A(y_0)$ es la probabilidad de seleccionar el punto y_0 sobre la superficie de una fuente luminosa.

Usando estas definiciones, la densidad de probabilidad $p_{s,t}$ con la que se genera el camino $\overline{x}_{s,t}$ se define como:

$$p_{s,t}(\overline{x}_{s,t}) = p_s^L p_t^E \tag{2.27}$$

Esta densidad es fundamental al calcular los pesos de cada técnica.

Multiple Importance Sampling

Hasta este punto, como se aprecia en la Figura 2.10, hay distintas "técnicas" para la generación de un camino. Cada una de estas técnicas presenta más o menos varianza dependiendo de la zona de la escena sobre la que se estén generando las muestras. Por ejemplo, si se realiza el trazado de caminos desde la cámara, teniendo una única fuente de luz muy pequeña, la mayoría de los rayos no la impactarán; por tanto, la imagen generada a partir de las muestras tendrá mucho ruido. Si se disponen de otras técnicas que presenten menor varianza en la misma zona, se podrían priorizar las muestras generadas por estas para obtener un resultado más estable.

La idea de ponderar las muestras de distintas técnicas para reducir la varianza se conoce como *Multiple Importance Sampling* (MIS).

Recordando la Ecuación 2.20, que define el estimador general de la imagen:

$$F = \sum_{s\geq 0}\sum_{t\geq 0} w_{s,t}(\overline{x}_{s,t})\frac{f(\overline{x}_{s,t})}{p_{s,t}}\,, \tag{2.28}$$

cada par (s,t) identifica una técnica distinta de generación de caminos, $p_{s,t}$ es la probabilidad asociada a dicha técnica y $w_{s,t}$ es el peso MIS correspondiente.

Los pesos w_i pueden calcularse de múltiples formas; se utilizará la heurística de potencia (*power heuristic*), introducida por Veach (Veach, 1997, p. 273), definida como

$$w_i = \frac{(pdf_i(X_i))^2}{\sum_{k=1}^{n}(pdf_k(X_i))^2}\,, \tag{2.29}$$

donde $pdf_i(X_i)$ representa la densidad de probabilidad con que la técnica i genera la muestra X_i.

Optimización del cálculo

Utilizando la notación $k = s + t - 1$ y $p_i = P_A(\overline{x}_{i,s+t-i})$, el camino $\overline{x}_{s,\,t}$ se puede reescribir como: $\overline{x} = x_0 \ldots x_k$; lo que simplifica la ecuación de la heurística de potencia a la siguiente expresión:

$$w_{s,\,t} = \frac{p_s^2}{\sum_{i=0}^{k+1} p_i^2} \tag{2.30}$$

Se observa como p_s es la probabilidad de la estrategia con la que el camino actual fue efectivamente generado. El resto de probabilidades $p_0 \ldots p_{s-1}$ y $p_{s+1} \ldots p_{s+t}$ representan las distintas formas en que el mismo camino $\overline{x}_{s,t}$ podría haberse generado.

Usando esta notación, se puede reescribir la expresión de la siguiente forma:

$$w_{s,\,t} = \frac{p_s^2}{\sum_{i=0}^{k+1} p_i^2} = \frac{1}{\sum_{i=0}^{k+1}(p_i \,/\, p_s)^2} \tag{2.31}$$

Esta fórmula muestra que el peso $w_{s,t}$ de la técnica que genera el camino $\overline{x}_{s,t}$ depende únicamente de los cocientes $\frac{p_i}{p_s}$, es decir, de qué tan probable era generar el mismo camino mediante las demás técnicas en comparación con la técnica actual.

Utilizando la definición de las probabilidades (2.25) y (2.26) para cada subcamino, podemos expresar los p_i para los caminos de largo $k = 3$:

$$\begin{aligned}
p_0 &= P_A(x_3)P_\sigma(x_3 \to x_2)G(x_3 \leftrightarrow x_2)P_\sigma(x_2 \to x_1)G(x_2 \leftrightarrow x_1) \\
&\quad \cdot\ P_\sigma(x_1 \to x_0)G(x_1 \leftrightarrow x_0) \\
p_1 &= P_A(x_3)P_\sigma(x_3 \to x_2)G(x_3 \leftrightarrow x_2)P_\sigma(x_2 \to x_1)G(x_2 \leftrightarrow x_1)P_A(x_0) \\
p_2 &= P_A(x_3)P_\sigma(x_3 \to x_2)G(x_3 \leftrightarrow x_2)P_\sigma(x_0 \to x_1)G(x_0 \leftrightarrow x_1)P_A(x_0) \\
p_3 &= P_A(x_3)P_\sigma(x_1 \to x_2)G(x_1 \leftrightarrow x_2)P_\sigma(x_0 \to x_1)G(x_0 \leftrightarrow x_1)P_A(x_0) \\
p_4 &= P_A(x_3)P_\sigma(x_2 \to x_3)G(x_2 \leftrightarrow x_3)P_\sigma(x_1 \to x_2)G(x_1 \leftrightarrow x_2) \\
&\quad \cdot\ P_\sigma(x_0 \to x_1)G(x_0 \leftrightarrow x_1)P_A(x_0)
\end{aligned} \tag{2.32}$$

Cada uno de estos términos representa la probabilidad de generar el mismo camino $\overline{x}$ utilizando una técnica distinta.

Si se calcula, por ejemplo, el cociente entre dos de estas probabilidades, como p_2/p_1, obtenemos:

$$\begin{aligned}
\frac{p_2}{p_1} &= \frac{P_A(x_3)P_\sigma(x_3 \to x_2)G(x_3 \leftrightarrow x_2)P_\sigma(x_0 \to x_1)G(x_0 \leftrightarrow x_1)P_A(x_0)}{P_A(x_3)P_\sigma(x_3 \to x_2)G(x_3 \leftrightarrow x_2)P_\sigma(x_2 \to x_1)G(x_2 \leftrightarrow x_1)P_A(x_0)} \\
&= \frac{P_\sigma(x_0 \to x_1)\, G(x_0 \leftrightarrow x_1)}{P_\sigma(x_2 \to x_1)\, G(x_2 \leftrightarrow x_1)}.
\end{aligned} \tag{2.33}$$

Este cociente indica la relación entre la probabilidad de generar el mismo camino utilizando una técnica que conecta los vértices (x_0, x_1), frente a otra que conecta (x_1, x_2).

Esta expresión se puede generalizar para el resto de los cocientes:

$$\begin{aligned}
&\frac{p_1}{p_0} = \frac{P_A(x_0)}{P_\sigma(x_1 \to x_0)\, G(x_1 \leftrightarrow x_0)}, \\
&\frac{p_{i+1}}{p_i} = \frac{P_\sigma(x_{i-1} \to x_i)\, G(x_{i-1} \leftrightarrow x_i)}{P_\sigma(x_{i+1} \to x_i)\, G(x_{i+1} \leftrightarrow x_i)} \quad \text{para } 0 < i < k, \\
&\frac{p_{k+1}}{p_k} = \frac{P_\sigma(x_{k-1} \to x_k)\, G(x_{k-1} \leftrightarrow x_k)}{P_A(x_k)}.
\end{aligned} \tag{2.34}$$

Luego, para calcular p_i/p_s se puede realizar el siguiente cálculo:

$$\begin{aligned}
\frac{p_i}{p_s} &= \frac{p_i}{p_{i+1}}\frac{p_{i+1}}{p_{i+2}}\cdots\frac{p_{s-2}}{p_{s-1}}\frac{p_{s-1}}{p_s} = \prod_{j=i}^{s-1}\frac{p_j}{p_{j+1}} \quad \text{para} \quad i < s, \\
\frac{p_i}{p_s} &= \frac{p_{s+1}}{p_s}\frac{p_{s+2}}{p_{s+1}}\cdots\frac{p_{i-1}}{p_{i-2}}\frac{p_i}{p_{i-1}} = \prod_{j=s+1}^{i}\frac{p_j}{p_{j-1}} \quad \text{para} \quad i > s
\end{aligned} \tag{2.35}$$

Se observa cómo este producto se puede realizar de forma incremental (realizando la multiplicación de cada cociente) para obtener todos los cocientes p_i/p_s con $0 \leq i < s$, y de forma análoga para $s < i \leq k+1$. Sumando todos estos cocientes, se puede calcular el valor del peso siguiendo la Ecuación (2.31).

2.4. Introducción a Vulkan

Esta sección tiene como objetivo presentar el flujo de procesamiento de trazado de rayos de Vulkan, las etapas que lo componen y el intercambio de datos entre las mismas.

Vulkan es una API gráfica que permite comunicarse directamente con la tarjeta gráfica (GPU). Se considera de bajo nivel porque da un control muy detallado al programador: la aplicación debe encargarse por sí misma de manejar la memoria, crear los command buffers (listas de instrucciones para la GPU) y enviarlas a las colas de ejecución. Esto hace que Vulkan sea más compleja de usar, pero también más rápida y eficiente.

Para el trazado de rayos, Vulkan utiliza un conjunto de extensiones:

- `VK_KHR_ray_tracing_pipeline`: define las etapas programables (*raygen*, *miss*, *closest hit*, *any hit*, *intersection*, *callable*) y la instrucción de trazado.
- `VK_KHR_acceleration_structure`: define las estructuras de aceleración (TLAS/BLAS) y sus construcciones/actualizaciones.
- `VK_KHR_deferred_host_operations`: soporte para operaciones diferidas de construcción.

Estas extensiones de trazado de rayos fueron publicadas por *Khronos* como extensiones provisionales a principios de 2020 y se estandarizaron a finales de 2020.

Gracias a estas extensiones, el trazado de rayos se puede usar dentro de Vulkan como una estrategia de renderizado más, junto a la rasterización tradicional. Para que el trazado de rayos funcione con aceleración real, la GPU debe tener hardware especializado en trazado de rayos y los controladores deben exponer las extensiones de Vulkan correspondientes. Actualmente, las GPUs con hardware especializado para el trazado de rayos son:

- `NVIDIA`: Turing (2018, RTX 2000), Ampere (2020, RTX 3000), Ada Lovelace (2022, RTX 4000) y Blackwell (2025, RTX 5000).

- `AMD`: RDNA 2 (2020, Radeon RX 6000), RDNA 3 (2022, Radeon RX 7000) y RDNA 4 (2025, Radeon RX 9000).
- `Intel`: Arc Alchemist (2022) y Arc Battlemage (2024 - 2025). Además, Core Ultra con gráficos Arc integrados (2023).

2.4.1. Ducto de Trazado de Rayos

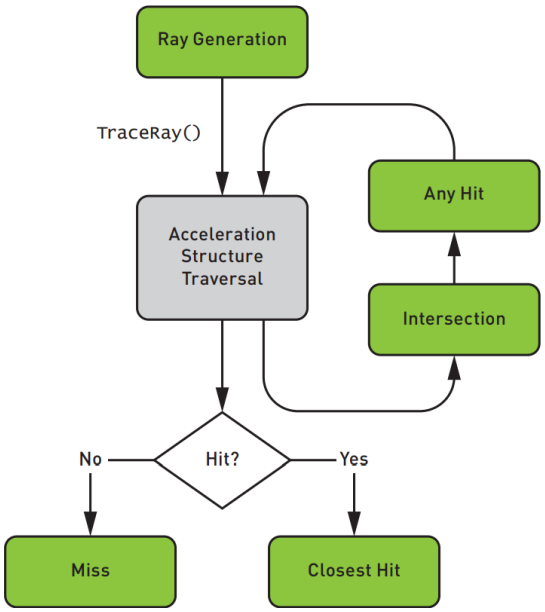


Figura 2.11: Diagrama de flujo del ducto de trazado de rayos
Fuente: (Marrs, Shirley, y Wald, 2021, fig. 16-2)

El ducto para el trazado de rayos de Vulkan se compone de una serie de etapas programables que interactúan entre sí, siguiendo el flujo representado en la Figura 2.11. Cada etapa es implementada en un shader, aunque es opcional implementar las etapas de *Any hit* e *Intersection*.

Ray Generation (`.rgen`).

El trazado de rayos comienza en la etapa de *Ray Generation*, responsable de emitir los rayos primarios (y, si corresponde, los secundarios) que recorrerán la escena. Esta etapa se ejecuta en forma paralela sobre una cuadrícula de una, dos o tres dimensiones, generalmente un hilo por píxel de la imagen a generar. En esta etapa, se invoca la instrucción de trazado (`traceRayEXT`), que consulta las estructuras de aceleración (TLAS/BLAS) y determina las intersecciones. Según el resultado, se ejecutarán las etapas que correspondan para procesar la

intersección (o la falta de esta) y retornar el resultado a la etapa de generación para componer la imagen.

Miss (`.rmiss`).

Cuando no se encuentra ninguna intersección válida, se ejecuta la etapa *Miss*. Esta etapa podría ser utilizada para tomar una muestra de un *skybox* o simplemente devolver un color por defecto. Tanto esta etapa como la de *Closest-hit* son capaces de trazar rayos de forma recursiva.

Closest Hit (`.rchit`).

La etapa de *Closest-hit* se ejecuta para la intersección más cercana al origen del rayo y se utiliza comúnmente para realizar cálculos de iluminación y evaluación de materiales. Es posible tener más de un shader para esta etapa; el shader a utilizar para una intersección dada depende del *shader binding table* y de la instancia dentro de la estructura de aceleración en la que se haya encontrado la intersección. Se profundizará en este aspecto más adelante. También puede emitir rayos secundarios usando `traceRayEXT`.

Intersection (`.rint`).

La etapa *Intersection* se encarga de determinar si existe una intersección entre un rayo y una geometría dentro de la escena, y en caso afirmativo, informa la distancia a la intersección desde el origen del rayo. Esto es útil para implementar geometrías definidas por el usuario, como puede ser una esfera. Las intersecciones rayo-triángulo tienen soporte por defecto; si no se especifica un shader de intersección, se utilizará este.

Any Hit (`.rahit`).

La etapa *Any Hit* se ejecuta para cada intersección encontrada. Esta etapa puede ser utilizada para descartar intersecciones; por ejemplo, para realizar *alpha testing*, descartando las intersecciones que ocurren en puntos donde la textura no cumple con cierto criterio (como tener una transparencia inferior a cierto umbral).

2.4.2. Estructuras de aceleración

Para optimizar el rendimiento en escenas complejas, Vulkan utiliza estructuras de datos especializadas para acelerar las pruebas de intersección entre rayos y geometría. Estas estructuras, llamadas *Acceleration Structures* (AS), se construyen a partir de la información de la escena y organizan la geometría de manera jerárquica. La jerarquía se divide en dos niveles: el *Bottom-Level Acceleration Structure* (BLAS), que contiene los triángulos o los AABBs (para el caso de las primitivas personalizadas que defina la aplicación) que componen la escena; y el *Top-Level Acceleration Structure* (TLAS), que almacena referencias

a uno o más BLAS, junto con información adicional, como las transformaciones o el grupo de shaders a utilizar. La relación entre estas estructuras puede observarse en la Figura 2.12.

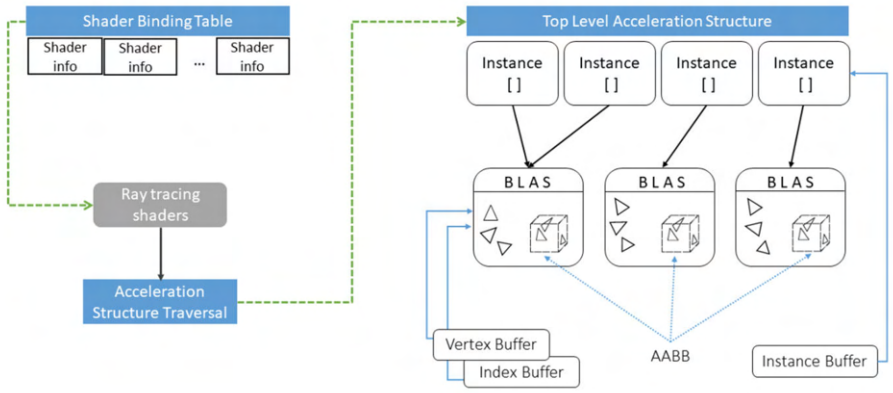


Figura 2.12: Diagrama de la organización y la interacción de las estructuras de aceleración con el ducto de trazado de rayos. Fuente: (Khronos Group, 2023)

2.4.3. Shader Binding Table (SBT)

La **Shader Binding Table (SBT)** es una estructura en memoria que asocia los grupos de shaders con las diferentes etapas del ducto de trazado de rayos y con cada instancia o primitiva de la escena. La tabla contiene registros que definen los shaders a ejecutar en cada tipo de grupo:

- *Ray Generation Group*: etapa general de generación de rayos.
- *Miss Groups*: uno o varios shaders que se ejecutan cuando un rayo no interseca ninguna geometría.
- *Hit Groups*: grupos asociados a intersecciones que combinan shaders de *closest hit*, *any hit* y, en el caso de primitivas personalizadas, también *intersection*.
- *Callable Groups* (opcional): shaders que pueden invocarse explícitamente desde otros shaders.

Cuando un rayo interseca una instancia del TLAS, el *record* correspondiente en la SBT determina exactamente qué shaders del *hit group* se ejecutan para esa primitiva. De esta forma, es posible asignar materiales o comportamientos diferentes por instancia o por geometría sin necesidad de cambiar el ducto.

2.4.4. Comunicación entre etapas

Cada etapa tiene acceso a ciertas variables internas de Vulkan para referenciar elementos como las matrices de transformación, el ID de la primitiva impactada, etc; Sin embargo, la comunicación entre las etapas se realiza a través de un payload compartido.

El tipo que utiliza el payload es definido por el usuario en los shaders, comúnmente se usa un struct con todos los datos o flags que sean necesarios para mantener el estado del rayo entre las distintas etapas o para comunicar datos de la intersección al resto de etapas. Por ejemplo, al terminar la ejecución del *closest-hit shader*, se podría retornar el color en el punto impactado, la normal o la nueva dirección en la que debería proyectarse el rayo en el siguiente paso.

2.4.5. Primitivas Personalizadas

La aplicación tiene la posibilidad de definir primitivas geométricas arbitrarias. Para ello se utiliza el shader de intersección (`.rint`), donde se evalúa si un rayo dado interseca la geometría correspondiente. En caso de que exista intersección, esta se notifica mediante una llamada a la función *reportIntersectionEXT*.

Un ejemplo en pseudocódigo para el caso de una esfera se presenta en el Algoritmo 1.

Algoritmo 1 Intersección Rayo-Esfera

$ray\_origin \leftarrow gl\_WorldRayOriginEXT$
$ray\_direction \leftarrow gl\_WorldRayDirectionEXT$

$oc \leftarrow ray\_origin - sphere\_center$
$a \leftarrow dot(ray\_direction, ray\_direction)$
$b \leftarrow 2{,}0 \ * dot(oc, ray\_direction)$
$c \leftarrow dot(oc, oc) - radius * radius$
$discriminant \leftarrow b * b - 4 * a * c$
if $discriminant \geq 0$ **then**
    $t \leftarrow (-b - sqrt(discriminant))/(2{,}0 * a)$
    $reportIntersectionEXT(t)$

Las primitivas personalizadas requieren definir un AABB al momento de crear la BLAS. De esta forma, solo se ejecutará el *intersection shader* si el rayo impacta el AABB.

2.5. GPU y Procesamiento Paralelo

La ejecución de algoritmos de trazado de rayos sobre GPU no depende únicamente del costo algorítmico de cada técnica, sino también de cómo el hardware explota el paralelismo y de cómo se comporta el flujo de control de los shaders.

En particular, el modelo de ejecución de las GPU modernas favorece cargas de trabajo uniformes y penaliza aquellas con alta variabilidad lógica entre hilos.

2.5.1. Paralelismo masivo y modelo de cómputo en GPU

Las GPU están orientadas a maximizar el *throughput* ejecutando una gran cantidad de hilos en paralelo. En cargas de trabajo de renderizado, es habitual asociar el cómputo a una grilla de ejecución (por ejemplo, por píxel o por muestra), de forma que muchas invocaciones del mismo *shader* procesan datos distintos de manera concurrente.

Este paralelismo se ve condicionado por dos factores: (i) la capacidad del hardware de mantener muchos hilos activos, y (ii) la uniformidad del trabajo realizado por dichos hilos. Cuando el trabajo por hilo varía de forma significativa, el paralelismo efectivo disminuye.

2.5.2. Ejecución SIMT

Las arquitecturas modernas implementan un modelo SIMT (*Single Instruction, Multiple Threads*). En este modelo, el hardware agrupa los hilos en unidades de ejecución que comparten el flujo de control. En NVIDIA estos grupos se denominan *warps*, mientras que en AMD se conocen como *wavefronts*.

Mientras todos los hilos del grupo siguen el mismo camino de ejecución, el hardware puede ejecutar instrucciones de manera eficiente. Cuando los hilos requieren caminos de ejecución distintos, el grupo debe resolver esa diferencia mediante serialización parcial, reduciendo el paralelismo efectivo.

2.5.3. Divergencia de control (branching)

La divergencia de control ocurre cuando hilos de un mismo grupo toman ramas diferentes de una condición (`if/else`, `switch`, bucles con distinta cantidad de iteraciones, etc.). En ese caso, la GPU ejecuta las ramas de forma serializada, habilitando y deshabilitando hilos según corresponda. El costo no proviene solo de la evaluación de la condición, sino de que una fracción de los hilos queda inactiva mientras se ejecuta la rama opuesta.

En trazado de caminos, la divergencia es frecuente debido a la naturaleza del algoritmo: rayos cercanos pueden (i) intersectar o no geometría, (ii) interactuar con materiales distintos, (iii) terminar en iteraciones diferentes por criterios de terminación, o (iv) disparar rayos secundarios distintos según el evento muestreado. Esta variabilidad produce diferencias de flujo de control entre hilos y reduce la eficiencia de ejecución.

2.5.4. Coherencia de rayos y acceso a memoria

Además del flujo de control, el rendimiento también se ve afectado por la coherencia espacial de los rayos y sus accesos a memoria. Rayos con direcciones similares tienden a recorrer regiones cercanas de las estructuras de aceleración

y a reutilizar datos, mientras que rayos incoherentes generan recorridos más dispersos, reducen la localidad y aumentan fallos de caché.

En escenas complejas, pequeñas variaciones en la geometría o en la distribución de luz pueden modificar la coherencia del conjunto de rayos (primarios o secundarios) y, por lo tanto, alterar el costo práctico del renderizado incluso manteniendo constantes parámetros como resolución o profundidad máxima.

Capítulo 3

Diseño

En este capítulo se presenta el alcance del proyecto, la estructura general del sistema y las principales decisiones de diseño adoptadas. Se describe la arquitectura del programa, los módulos que la componen y su interacción, junto con la organización de los *shaders* utilizados para el trazado de rayos.

También se detallan los mecanismos implementados para la carga de escenas, la interfaz gráfica y el intercambio de técnicas en tiempo real, que permiten evaluar y comparar las distintas estrategias de renderizado desarrolladas.

3.1. Alcance

El objetivo principal de este proyecto es la implementación y comparación de distintas técnicas de trazado de rayos para el renderizado fotorrealista, utilizando la API Vulkan y, en específico, sus extensiones para trazado de rayos. Con este fin, se desarrolla un programa que genera de forma incremental una imagen basada en una escena seleccionada por el usuario, permitiendo que los parámetros del programa se ajusten en tiempo real desde una interfaz gráfica, con el fin de comparar las diferencias entre las técnicas en términos de ruido, convergencia y rendimiento.

El programa permite generar imágenes con tres algoritmos distintos:

- **Trazado de Caminos Inverso (BPT)**, descrito en la Sección 2.3.2.
- **Trazado de Caminos Inverso con Estimador del Siguiente Evento (BPT-NEE)**, descrito en la Sección 2.3.3.
- **Trazado de Caminos Bidireccional (BDPT)**, descrito en la Sección 2.3.4.

Las tres técnicas definen los rayos que se trazan sobre la escena y realizan los cálculos necesarios para determinar el color de cada píxel de la imagen. Todas ellas comparten un mismo modelo de materiales, basado en el BSDF de Disney (2.1.3).

Se partió del trabajo de Fontana (2024), consistente en una serie de ejemplos de uso de la extensión de trazado de rayos de Vulkan.

A partir de esta base, se realizaron modificaciones para adaptarla a los requerimientos del proyecto. Se mantuvieron conceptos clave de la implementación original, como la organización modular de shaders y el uso de bibliotecas auxiliares (*NVVK Helper*, *GLM*, *ImGui*), pero se reestructuró la aplicación para incorporar nuevas funcionalidades orientadas a la investigación y comparación de los algoritmos a implementar.

A continuación, se mencionan los principales cambios que sufrió el proyecto base:

- Se unificó la aplicación en un único proyecto que permite el cambio de técnica en tiempo de ejecución, en lugar de contar con una aplicación independiente para cada una.
- Se centralizó el cálculo del BSDF en un único archivo común (*material.glsl*), reemplazando los cálculos específicos en los *.rchit*. A su vez, el cálculo del BSDF se reemplazo por una implementación del modelo BSDF de Disney (2.1.3).
- Se incorporó un sistema de carga de escenas general donde se definen los modelos y los parámetros a utilizar en un mismo archivo, sustituyendo la carga de modelos *.obj* independientes.
- Se extendieron los parámetros configurables desde la interfaz gráfica y se agregaron estadísticas de la ejecución del algoritmo.
- Se añadió el calculo de intersección con esferas mediante *intersection shader* personalizado.

3.1.1. Arquitectura del Sistema

Se adoptó una arquitectura modular orientada a objetos para abstraer la complejidad de Vulkan y facilitar la comparación de algoritmos. Este diseño desacopla la gestión de recursos gráficos de bajo nivel de la definición de la escena y la implementación de las técnicas de renderizado.

La Figura 3.1 presenta el diagrama de clases simplificado del sistema, destacando los módulos principales y sus relaciones de dependencia.

A continuación, se describen los roles y responsabilidades de los componentes esenciales:

Gestión Principal y API Gráfica

La clase `VulkanHandler` constituye el núcleo de la aplicación y actúa como una capa de abstracción sobre la API de Vulkan. Su responsabilidad principal es encapsular la complejidad asociada a la inicialización del contexto gráfico (instancia, dispositivos, *swapchain* y colas de comandos) y coordinar el ciclo de renderizado.

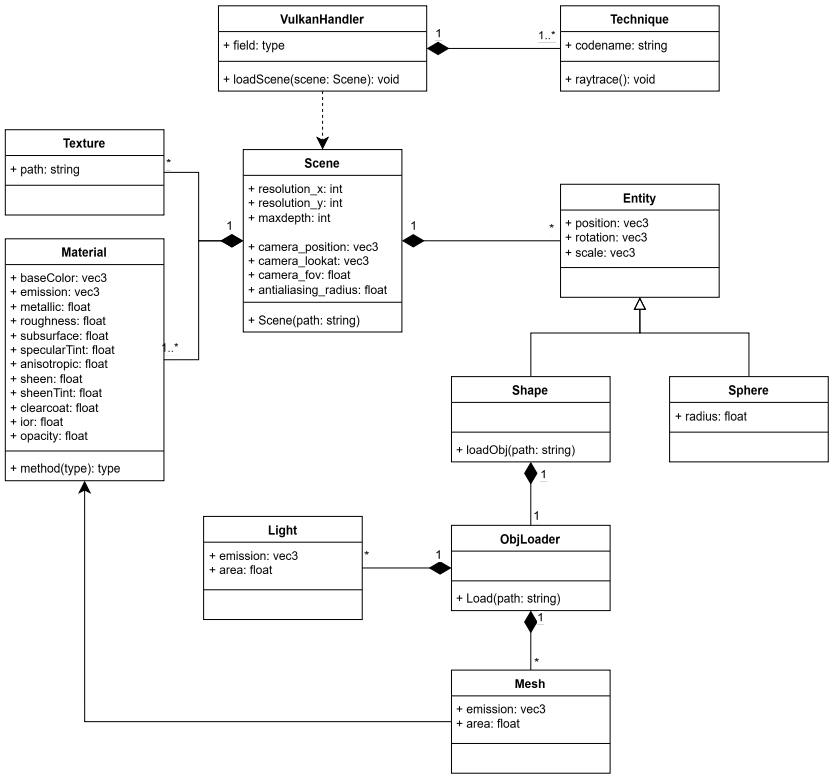


Figura 3.1: Diagrama de clases del sistema. La clase `VulkanHandler` actúa como coordinador central, gestionando la interacción entre la representación de la escena (`Scene`) y las distintas estrategias de trazado (`RayTracingPipeline`).

Esta clase es la encargada de gestionar la carga de los recursos de la escena en la GPU, despachar los comandos de dibujo y presentar la imagen final en pantalla.

Abstracción de Técnicas de Trazado

Para cumplir con el requisito de intercambiar algoritmos en tiempo real, se diseñó la abstracción `RayTracingPipeline`. Esta clase encapsula todo el estado necesario para ejecutar una técnica específica de trazado de rayos. Sus responsabilidades incluyen:

- La gestión de los grupos de shaders específicos para la técnica (*ray generation*, *miss*, *closest hit*).
- La configuración del *Pipeline Layout* y los *Descriptor Sets* necesarios para acceder a los recursos de la escena.
- La construcción de la *Shader Binding Table* (SBT) asociada al algoritmo.

La clase `VulkanHandler` gestiona una colección de estas técnicas (BPT, BPT-NEE, BDPT) y permite alternar el puntero del ducto activo dinámicamente, evitando la necesidad de reconstruir el contexto gráfico completo ante un cambio de algoritmo.

Representación de la Escena

La escena se modela a través de la clase `Scene`, la cual actúa como un contenedor de alto nivel independiente de la API gráfica. Esta clase es responsable de la carga de archivos de definición de escena (formato JSON/SCN):

- **Entidades:** La escena contiene una lista de entidades polimórficas (`Entity` y sus derivadas `Shape`/`Sphere`). La clase `Entity` representa estas entidades renderizables, agrupando la información de sus transformaciones espaciales (posición, rotación, escala) y su geometría.
- **Geometría y Materiales:** Para las mallas poligonales, se utiliza la clase auxiliar `ObjLoader`, encargada de leer archivos *.obj*, procesar vértices e índices, y organizar las propiedades de los materiales (color, texturas, parámetros físicos) en estructuras de datos que posteriormente serán subidas a la GPU.

El formato completo del archivo de escena y el modelo de materiales se describen en el Anexo A.

3.2. Shaders

El ducto de trazado de rayos se compone de una serie de *shaders* que interactúan entre sí para realizar las distintas etapas del proceso de renderizado. Cada uno cumple un rol específico dentro del flujo de ejecución y se comunica con los demás mediante estructuras compartidas en GPU.

Los shaders que componen el programa se dividen en tres grupos:

- **Shaders específicos a cada técnica:** Cada técnica implementa los shaders {*técnica*}*.rgen*, {*técnica*}*.rchit*, {*técnica*}*_primitives.rchit*, {*técnica*}*.rahit*, {*técnica*}*.rmiss*.
- **Shaders comunes a todas las técnicas:** *pathtracer.rint*, *pathtracer.frag*, *pathtracer.vert*.
- **Shaders adicionales:** *lights.glsl*, *material.glsl*, *random.glsl*, *raycommon.glsl*, *host_device.h*.

3.2.1. Shaders específicos a cada técnica

Cada técnica implementa cinco shaders que cumplen distintas funciones:

- **Ray Generation Shader (.rgen)**

  Contiene el algoritmo principal de la técnica a implementar. Se encarga de invocar el trazado de rayos y de realizar los cálculos de iluminación en función del resultado de las intersecciones (la normal, el valor del BSDF, etc).

- **Ray Closest Hit (.rchit)**

  Determina las propiedades de la superficie en el punto de intersección. Además, calcula la próxima dirección del rayo y el resultado de la interacción de la luz con la superficie, invocando el BSDF definido en el shader *material.glsl*.

  Existen dos versiones de este shader: una para las intersecciones con triángulos (en el caso de modelos 3D) y otra para las primitivas personalizadas.

- **Ray Any Hit (.rahit)**

  Se utiliza en los casos en los que no es necesario calcular la iluminación. Un ejemplo de esto son los test de visibilidad (o rayos de sombra), donde se proyecta un rayo entre dos puntos de la escena para comprobar si existe alguna superficie intermedia; por lo que sólo nos interesa saber si ocurrió una intersección o no.

- **Ray Miss (.rmiss)**

  Define el resultado cuando un rayo no impacta con ninguna superficie de la escena.

3.2.2. Shaders comunes a todas las técnicas

Los siguientes shaders cumplen funciones generales del ducto y son utilizados por todas las técnicas por igual.

- *pathtracer.rint*

  Determina si ocurre una intersección entre un rayo y una primitiva geométrica.

- *pathtracer.frag*

  Aplica una corrección gamma a la imagen generada por la técnica activa antes de presentarla en la interfaz gráfica. En el caso de BDPT, donde se generan dos imágenes como resultado (como se explicará en la Sección 4.11.3), también se encarga de combinarlas en una sola imagen.

- *pathtracer.vert*

  Aplica transformaciones a los vértices del quad al que se le aplica como textura la salida del algoritmo.

3.2.3. Shaders adicionales

El resto de los shaders se encuentra dividido por funciones, con el objetivo de mantener un diseño modular y favorecer la reutilización del código entre las distintas técnicas implementadas.

- *lights.glsl*

  Contiene las funciones relacionadas con el muestreo de direcciones y las distribuciones de probabilidad (PDF) de las fuentes de luz.

- *material.glsl*

  Contiene las funciones relacionadas con el cálculo del BSDF.

- *random.glsl*

  Contiene funciones para el muestreo de variables aleatorias.

- *raycommon.glsl*

  Define los objetos compartidos entre los diferentes shaders, además de otros objetos específicos a cada técnica.

- *host_device.h*

  Define las estructuras de datos que se comparten entre el programa principal y los shaders.

3.3. Interfaz Gráfica

La interfaz gráfica (GUI) debe ofrecer un conjunto de controles que permitan modificar, de forma sencilla e intuitiva, tanto los parámetros de la cámara como los de los algoritmos durante la ejecución. Asimismo, debe proporcionar estadísticas e información relevante sobre el estado del renderizado. A continuación, se detallan las principales funcionalidades requeridas:

- Selección de escena mediante un menú inicial.
- Información en pantalla que permita monitorear el estado del algoritmo, incluyendo la técnica utilizada, el tiempo de ejecución, la cantidad de iteraciones, los cuadros por segundo (FPS) y el tiempo entre fotogramas.
- Pausar la ejecución del algoritmo, tanto manualmente como de forma automática, al alcanzar un límite de tiempo o de iteraciones.
- Reiniciar la ejecución del algoritmo.
- Ajuste de parámetros de la cámara, tales como la distancia focal, la apertura y el campo de visión. Además, debe permitir restablecer la posición de la cámara a su configuración inicial.

- Ajuste de parámetros de los algoritmos, por ejemplo, la profundidad máxima. En el caso de trazado de caminos bidireccional, se debe poder seleccionar una técnica específica para su análisis, así como habilitar o deshabilitar el uso de MIS.

3.4. Carga de escenas

Para la definición de escenas de prueba, se diseñó un formato de escenas que permite declarar los objetos que las componen (mallas de triángulos y/o esferas) y los materiales presentes en ellas.

Los materiales de los modelos *.obj* se definen en el archivo de escena, en lugar de en un archivo *.mtl* adicional, preservando las referencias a los materiales presentes en el *.obj*. Esto permite centralizar los parámetros de la escena en un solo archivo, además de permitir sobrescribir los materiales de los modelos.

3.5. Intercambio de técnicas en tiempo real

Para facilitar la comparación entre las distintas técnicas, se implementó el manejo de múltiples ductos de trazado de rayos. Cada ducto compila sus propios shaders y mantiene su propia *shader binding table*.

Los datos de la escena son compartidos entre todos los ductos, al igual que otros *buffers* con información extra, que pueden o no ser utilizados por las distintas técnicas.

Capítulo 4

Implementación

En este capítulo se describen los aspectos prácticos de la implementación del sistema, abordando los detalles técnicos y las soluciones concretas adoptadas para la construcción del proyecto. Se abordan los componentes más relevantes para el funcionamiento del renderizador y para la comparación de las técnicas implementadas.

En primer lugar, se presenta la implementación de la interfaz gráfica, junto con la generación de números pseudoaleatorios en la GPU y la creación de rayos primarios. A continuación, se detallan los mecanismos para la gestión de la escena, incluyendo el acceso a las fuentes de luz, el muestreo aleatorio sobre superficies emisivas y el manejo de primitivas personalizadas. Luego, se especifica la implementación del modelo de materiales, explicando la adaptación del modelo BSDF de Disney al proyecto. Finalmente, se profundiza en la implementación de los tres algoritmos de iluminación global desarrollados (BPT, BPT-NEE y BDPT), mencionando las optimizaciones y limitaciones observadas durante el desarrollo.

4.1. Interfaz Gráfica

La interfaz gráfica fue implementada utilizando la librería *ImGui* (Cornut, 2025), la cual permite superponer elementos de control sobre la imagen renderizada sin interferir con el ducto de renderizado. A través de esta interfaz es posible visualizar estadísticas de ejecución, ajustar parámetros del algoritmo utilizado y modificar propiedades de la cámara en tiempo real.

4.1.1. Estadísticas

Durante la ejecución, el sistema muestra un panel con información relevante para el análisis del desempeño del algoritmo (ver Figura 4.1). Dicho panel incluye el algoritmo seleccionado, el número total de iteraciones realizadas, el tiempo acumulado de ejecución, los valores de FPS y tiempo por fotograma, así como

el estado actual del proceso (por ejemplo, ejecutando o pausado). Además, se listan los atajos de teclado disponibles.

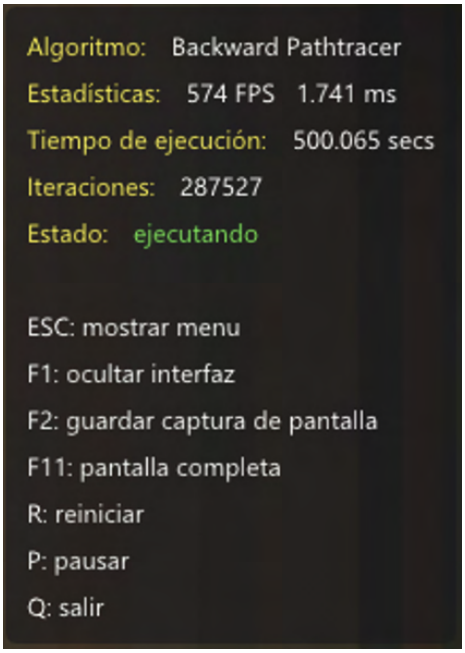


Figura 4.1: Panel de estadísticas mostrado durante la ejecución, incluyendo FPS, tiempo por fotograma, número de iteraciones y controles disponibles.

4.1.2. Configuración del Algoritmo

La interfaz permite modificar en tiempo real los parámetros de las distintas técnicas implementadas (ver Figura 4.2). Entre los valores configurables se encuentran:

- Selección del algoritmo (BPT, BPT-NEE o BDPT).
- Profundidad máxima del camino (o de ambos caminos en el caso del BDPT).
- Parámetros específicos del BDPT:
  - Posibilidad de fijar una técnica particular (s, t).
  - Activar o desactivar *Multiple Importance Sampling* (MIS).
  - Activar o desactivar la contribución del camino seleccionado.
- Límites opcionales de ejecución, ya sea por tiempo o por número de iteraciones.

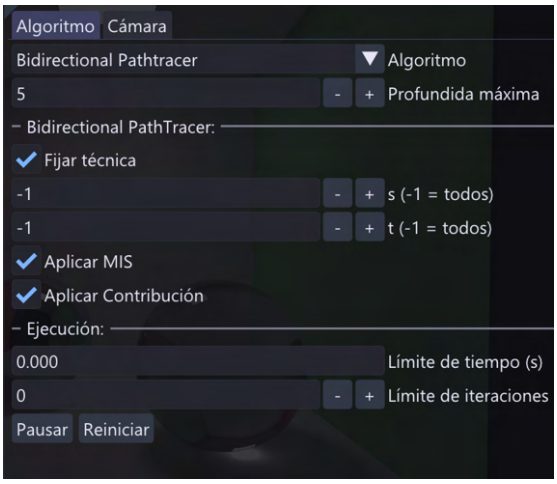


Figura 4.2: Panel de configuración del algoritmo, incluyendo selección de técnica, parámetros específicos de cada método y límites de ejecución.

4.1.3. Configuración de la Cámara

Además de los parámetros del algoritmo, la interfaz permite modificar las propiedades de la cámara utilizada para generar la imagen (ver Figura 4.3). Las opciones disponibles incluyen:

- Apertura del lente (relacionada con la profundidad de campo).
- Distancia focal.
- Campo de visión.
- Radio utilizado para el antialiasing.

Asimismo, la interfaz proporciona un botón para reiniciar la posición de la cámara a su estado inicial.

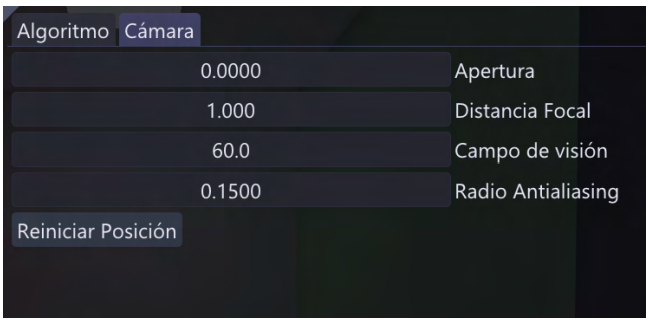


Figura 4.3: Panel de configuración de la cámara, donde es posible ajustar apertura, distancia focal, campo de visión y radio para antialiasing.

4.1.4. Selección de la Escena

Para la selección de la escena a renderizar se empleó el proyecto *imgui-filebrowser* (AirGuanZ, 2025). Esta herramienta proporciona una interfaz interactiva que permite navegar por el sistema de archivos, ingresar y salir de directorios, y filtrar los tipos de archivo visibles (ver Figura 4.4). En nuestro caso, se configuró para restringir la selección únicamente a archivos con extensión `.scn`, correspondientes al formato utilizado por nuestras escenas.

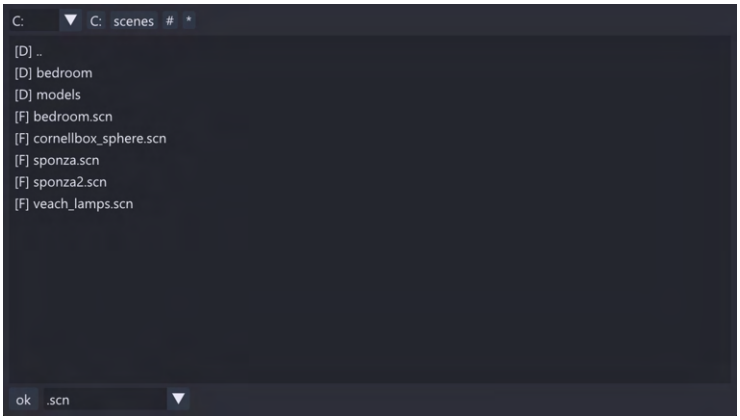


Figura 4.4: Interfaz de selección de escenas implementada con *imgui-filebrowser*, mostrando los archivos disponibles en formato `.scn`.

4.2. Guardado de salidas

Este proceso consiste en copiar la imagen producida por el algoritmo desde la memoria de la GPU hacia la memoria principal (CPU) y almacenarla utilizando el formato *.exr*, mediante la librería *TinyEXR* (Fujita, 2025).

El formato *OpenEXR* permite almacenar imágenes en alto rango dinámico (HDR), preservando los valores lineales de cada píxel, incluidos aquellos que exceden el rango de representación de la pantalla. Esto resulta útil para el análisis, ya que dichos valores pueden ser posteriormente reescalados o ajustados en exposición, permitiendo visualizar detalles que, en la imagen mostrada directamente en pantalla, quedarían saturados, o comparar algoritmos que producen imágenes con distintos niveles de luminosidad. En particular, al aplicarse una corrección gamma con fines de visualización, las regiones de alta luminancia se manifiestan como áreas completamente blancas al superar el rango $[0, 1]$ (o $[0, 255]$ en representación entera), ocultando información presente en la imagen original.

4.3. Generación de números pseudoaleatorios

Para el correcto funcionamiento de los algoritmos, es fundamental contar con un método eficiente para generar números pseudoaleatorios dentro de los *shaders* ejecutados en la GPU.

El generador debe realizar la menor cantidad posible de operaciones (y en el menor número de ciclos), además de poder ejecutarse de forma completamente paralela, sin requerir comunicación o sincronización entre distintos píxeles o iteraciones.

En este proyecto se utiliza un generador perteneciente a la familia de algoritmos PCG (O'Neill, 2014). En particular, se adopta la implementación mostrada en el Algoritmo 2, que ofrece un buen equilibrio entre la calidad de imagen y el rendimiento computacional (Jarzynski y Olano, 2020).

Algoritmo 2 Hash PCG (Jarzynski y Olano, 2020)

```
uint pcg_hash(inout uint seed) {
  seed = seed*747796405u+2891336453u;
  uint word = ((seed>>((seed>>28u)+4u))^seed)*277803737u;
  return (word>>22u)^word;
}
```

4.4. Generación de rayos primarios

El proceso de generación de rayos primarios consiste en calcular, para cada píxel de la imagen, un rayo que parte desde la posición de la cámara y se dirige hacia la escena. Este rayo define la dirección en la que se evaluará la visibilidad y la contribución luminosa correspondiente a ese píxel.

Sea (i, j) el identificador del píxel actual dentro de todos los hilos que estan ejecutando el shader, y (W, H) las dimensiones de la imagen. Las coordenadas normalizadas del píxel se obtienen como:

$$d = 2\left(\tfrac{i}{W},\ \tfrac{j}{H}\right) - \left(1,\ 1\right) \tag{4.1}$$

donde $d = (d_x, d_y)$ representa la posición del píxel en el plano de proyección en el rango $[-1, 1]^2$. El punto correspondiente en el espacio de la cámara se obtiene aplicando la matriz de proyección inversa P^{-1}:

$$p_c = P^{-1}\left(d_x,\ d_y,\ 1,\ 1\right) \tag{4.2}$$

Posteriormente, se utiliza la matriz de vista inversa V^{-1} para transformar este punto al espacio del mundo:

$$p_w = V^{-1}\ p_c \tag{4.3}$$

El origen del rayo corresponde al origen de la cámara en el espacio del mundo:

$$o = V^{-1} \begin{pmatrix} 0, & 0, & 0, & 1 \end{pmatrix} \tag{4.4}$$

y la dirección del rayo se define como el vector p_w normalizado:

$$d_r = \frac{p_w}{|p_w|} \tag{4.5}$$

De esta forma, el rayo primario queda definido por el par (o, d_r), donde o representa la posición de la cámara y d_r la dirección en la que se evalúa la visibilidad y la contribución luminosa asociada al píxel.

4.4.1. Profundidad de campo (Depth of Field)

El efecto de profundidad de campo se implementa desplazando el origen de los rayos dentro de un disco que representa la apertura de la cámara. Cada rayo se emite desde una posición distinta dentro de ese disco, lo que provoca que únicamente los puntos ubicados a la distancia de enfoque permanezcan nítidos.

Sea o el origen de la cámara, V^{-1} la matriz de vista inversa, f_d la distancia de enfoque y a el diámetro de la apertura. El punto de enfoque x_f se define como:

$$x_f = d \times f_d \tag{4.6}$$

donde d es la dirección del rayo sin aplicar la profundidad de campo.

A continuación, se generan dos valores aleatorios $r_1, r_2 \in [0, 1]$ para muestrear una posición dentro del disco de diámetro a:

$$\omega = r_1 \times 2\pi \tag{4.7}$$

$$l = r_2 \times \frac{a}{2} \tag{4.8}$$

Las direcciones horizontal y vertical de la cámara en el espacio del mundo se obtienen como:

$$c_r = V^{-1} \times (1, 0, 0, 0) \tag{4.9}$$

$$c_u = V^{-1} \times (0, 1, 0, 0) \tag{4.10}$$

Con ellas se calcula una posición aleatoria sobre el disco de apertura:

$$x_a = l \times \big(\cos(\omega) c_r + \sin(\omega) c_u \big) \tag{4.11}$$

El nuevo origen y la nueva dirección del rayo se definen como:

$$o_f = o + x_a \tag{4.12}$$

$$d_f = \frac{x_f - x_a}{|x_f - x_a|} \tag{4.13}$$

De esta manera, cada rayo se origina en un punto distinto del disco de apertura y se dirige hacia el punto de enfoque. Los rayos que convergen en el plano focal producen una imagen nítida en esa región, mientras que aquellos que intersecan planos más cercanos o más lejanos generan el desenfoque característico del efecto de profundidad de campo.

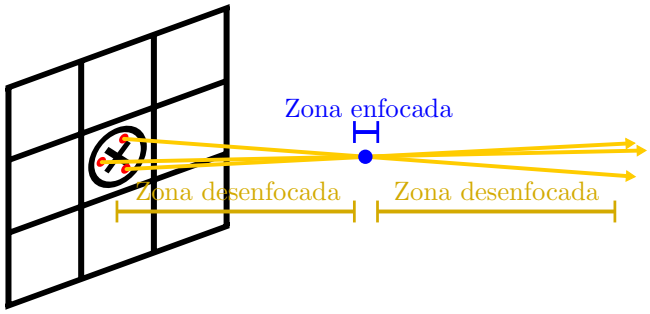


Figura 4.5: Rayos trazados desde el disco de apertura centrado en el píxel con dirección hacia el punto de enfoque.

En la Figura 4.5 se ilustra cómo los rayos generados desde diferentes puntos del disco convergen sobre el punto de enfoque y se separan al alejarse de él. Por su parte, la Figura 4.6 muestra una comparación entre una escena renderizada con y sin el efecto de profundidad de campo, donde puede observarse el desenfoque progresivo de las superficies que no se encuentran en el plano de enfoque.

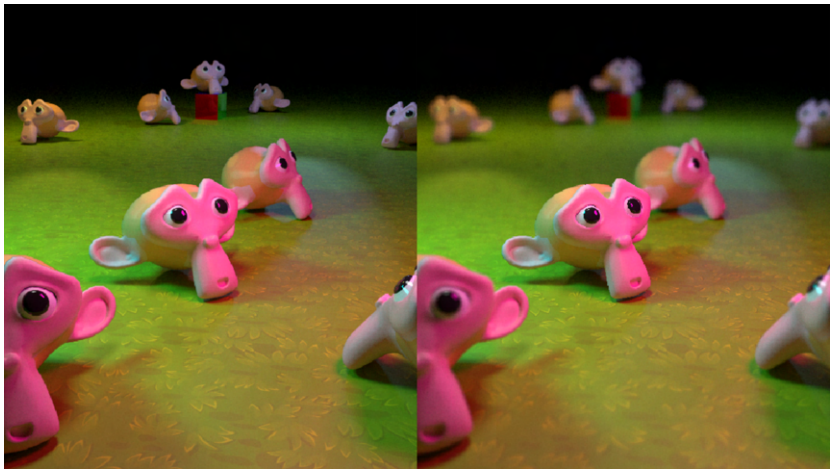


Figura 4.6: Comparación de una escena renderizada sin (izquierda) y con (derecha) el efecto de profundidad de campo. En la mitad izquierda se observa una imagen completamente nítida, mientras que en la derecha los objetos fuera del plano de enfoque presentan el desenfoque característico del efecto.

4.4.2. Antialiasing

En resoluciones bajas, la dirección inicial de los rayos primarios (cuando no se utiliza profundidad de campo) puede variar significativamente entre píxeles adyacentes. Esto provoca que los puntos de intersección sobre la escena difieran más de lo deseado, generando variaciones visibles en la contribución final del color. Cuando este efecto ocurre en los bordes de los objetos o en geometrías delgadas, se producen los llamados bordes en sierra (aliasing), como se muestra en la Figura 4.7.

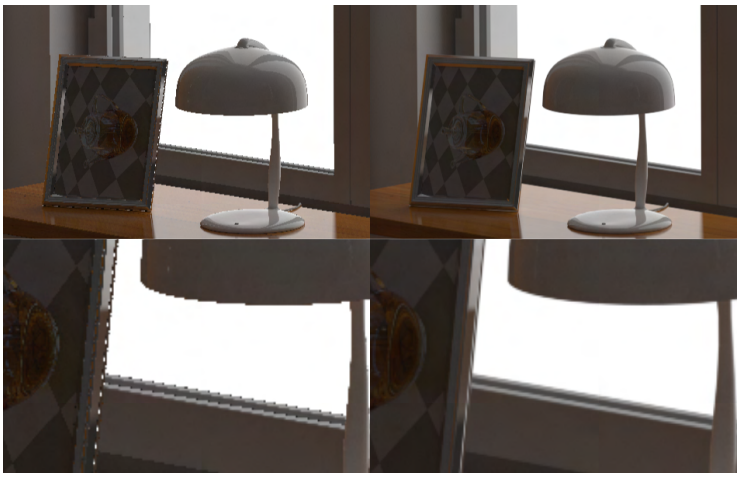


Figura 4.7: Comparación entre una imagen renderizada sin (izquierda) y con antialiasing (derecha).

Para reducir este problema, se introduce una ligera perturbación aleatoria en la posición efectiva del píxel. En lugar de muestrear siempre el centro del píxel, se generan múltiples muestras alrededor de él y se promedian sus contribuciones. De esta forma, el valor final del píxel representa mejor el área que cubre, suavizando así los contornos pronunciados.

La perturbación se obtiene mediante un vector

$$G = (G_x, G_y)$$

muestreado siguiendo una distribución normal (gaussiana) dentro de un círculo de radio configurable. Un radio mayor produce un muestreo más disperso alrededor del píxel, lo que incrementa el suavizado (aunque también puede introducir una pérdida de nitidez si se usa en exceso). En la implementación, el vector gaussiano se genera mediante el método de Box–Muller.

La perturbación G se aplica directamente al cálculo de d:

$$d = 2\left(\frac{i+G_x}{W},\ \frac{j+G_y}{H}\right) - \left(1,\ 1\right) \tag{4.14}$$

Finalmente, es importante remarcar que este método de antialiasing se basa en el mismo principio que la profundidad de campo: ambos introducen una perturbación en la posición desde donde se genera el rayo. Por lo tanto, si se activan simultáneamente, el efecto combinado puede generar una imagen mas borrosa de lo esperado.

4.5. Acceso a la información de las fuentes de luz

Dado que algunas de las técnicas implementadas requieren trazar rayos desde o hacia las fuentes de luz, fue necesario incorporar un *buffer* con la información de cada superficie capaz de emitir luz.

Las superficies emisivas se determinan durante el procesamiento de las geometrías y los materiales que componen la escena. Si el material asociado a una superficie presenta un valor de emisión distinto de cero, dicho valor se registra junto con la información necesaria para acceder a la geometría correspondiente.

En el caso de una malla de triángulos con material emisivo, se almacena el identificador del modelo que contiene la malla, así como los índices inicial y final de los triángulos que la componen. Con estos datos, es posible acceder a la información geométrica de la fuente de luz desde los *shaders*, de manera análoga a cómo, en el *closest-hit shader*, se accede al triángulo intersecado para obtener su normal y demás atributos.

Para las esferas emisivas, se almacena su identificador dentro del *buffer* de esferas; a partir de este dato, se puede acceder a su posición y a su radio.

4.6. Elección aleatoria de un punto en una luz

En los algoritmos de BDPT y BPT con Estimador del Siguiente Evento (NEE) es necesario seleccionar al azar una fuente de luz y, posteriormente, muestrear un punto aleatorio sobre su superficie. Para garantizar un muestreo uniforme en superficie, la elección de la fuente se realiza con una probabilidad proporcional a su área con respecto al área total de todas las fuentes luminosas. Luego, la selección de un triángulo dentro de la fuente se pondera según el área de cada triángulo.

La Figura 4.8 muestra dos distribuciones de puntos sobre dos triángulos. En la Figura superior, ambos triángulos reciben la misma cantidad de puntos, lo que provoca una mayor densidad en el triángulo más pequeño. En cambio, en la Figura inferior, la cantidad de puntos se distribuye de acuerdo con el área de cada triángulo, resultando en una distribución visualmente uniforme.

En nuestra implementación se asume que cada fuente luminosa está compuesta por triángulos de igual área. Esta suposición simplifica el proceso de muestreo, ya que permite seleccionar triángulos al azar sin necesidad de utilizar distribuciones acumulativas de probabilidad que garanticen un muestreo uniforme en superficie. En caso contrario, sería necesario construir y consultar dichas

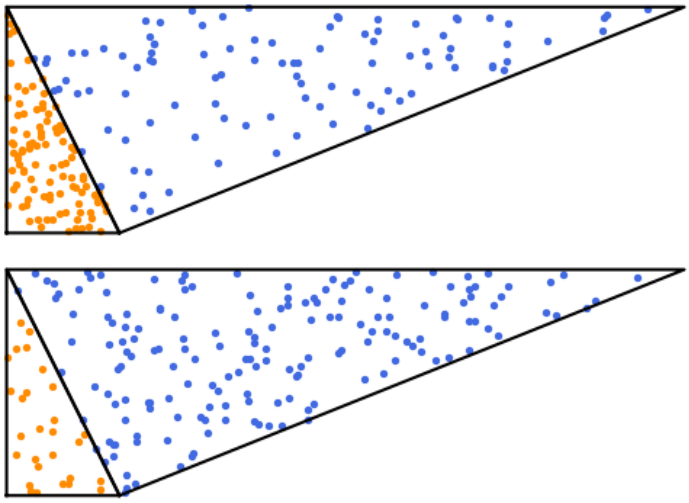


Figura 4.8: Distribución de 100 puntos aleatorios sobre dos triángulos. En la Figura superior no se considera el área de cada triángulo; en la inferior, la cantidad de puntos se pondera según el área.

distribuciones (por ejemplo, mediante una búsqueda binaria sobre un arreglo de áreas acumuladas), lo cual introduce *branching* y acceso irregular a memoria. Ambos factores implican un costo considerable, especialmente en la *GPU*, donde este tipo de operaciones afecta significativamente el rendimiento.

4.7. Primitivas Personalizadas

En Vulkan, el soporte nativo para colisiones rayo-escena está limitado a triángulos, ya que el hardware acelera automáticamente las pruebas rayo-triángulo. Para cualquier otra forma geométrica, la aplicación debe proporcionar un shader de intersección que defina la prueba de intersección matemática correspondiente.

Con el objetivo de representar esferas de manera más precisa, se incorporó soporte para otras primitivas geométricas.

El shader de intersección (*pathtracer_primitives.rint*) implementa las pruebas de intersección de todas las primitivas personalizadas. La implementación actual incluye únicamente esferas, aunque la estructura del código está diseñada para facilitar la incorporación de otros tipos de primitivas en el futuro.

4.7.1. Esferas

En las escenas, las esferas se definen especificando su centro y su radio. Esta información es utilizada por el shader de intersección para determinar si un rayo interseca o no la superficie de la esfera.

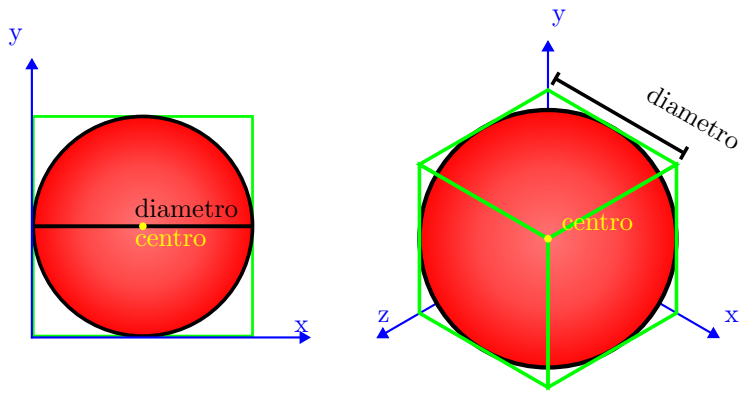


Figura 4.9: Esfera y su volumen contenedor AABB. El contenedor, de tamaño $2r$ en cada eje, encapsula la esfera y se utiliza para acelerar las pruebas de intersección.

Para cada esfera, se calcula un volumen contenedor AABB (*Axis-Aligned Bounding Box*), que se utiliza durante la construcción del BLAS. Este volumen define los límites espaciales de la esfera y permite que las pruebas de intersección iniciales sean aceleradas por hardware. Cuando un rayo impacta el AABB, se invoca al shader de intersección, que realiza la prueba de intersección exacta con la esfera.

La Figura 4.9 ilustra este proceso. A la izquierda se muestra una proyección bidimensional de una esfera inscrita en su volumen AABB, donde puede observarse que el tamaño del contenedor es igual al diámetro de la esfera ($2r$) en cada eje. A la derecha se presenta una visualización tridimensional de la esfera junto con su AABB, destacando cómo el volumen contenedor encapsula completamente la geometría y sirve como aproximación inicial para las pruebas de intersección.

4.8. Modelo de materiales

El modelo de materiales implementado es una adaptación del BSDF de Disney (Burley, 2015). Se decidió hacer ciertas simplificaciones y modificaciones específicas priorizando la facilidad de implementación y la claridad técnica para la comparación de algoritmos, sin perder la estructura general del modelo de referencia. Como simplificación inicial, y dado que la complejidad extra no aportaba valor significativo a la evaluación de las técnicas de trazado, se decidió omitir el lóbulo de `clearcoat`.

Toda la lógica de evaluación y muestreo del material se centraliza en un único archivo de shader (`material.glsl`). Este módulo es invocado por todas las técnicas de trazado desarrolladas (BPT, NEE y BDPT), garantizando una consistencia absoluta en la interpretación física de los materiales entre los distintos algoritmos.

La Figura 4.10 ilustra el comportamiento visual de los principales parámetros del modelo implementado, mostrando cómo afecta cada atributo a la apariencia de una esfera bajo condiciones de iluminación idénticas.

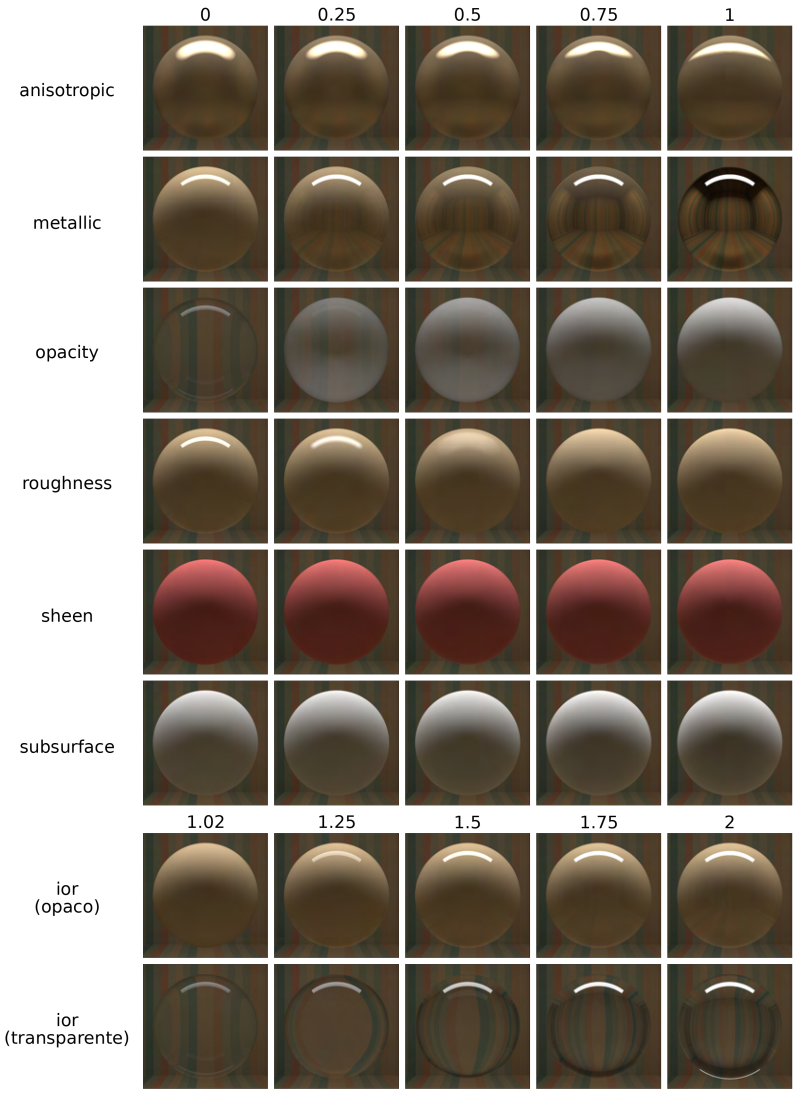


Figura 4.10: Variación de parámetros del modelo de materiales implementado con una esfera ejecutado con BPT.

A continuación, se detallan los componentes del modelo implementado y las

diferencias respecto al BSDF de Disney tomado como referencia:

4.8.1. Mezcla de Materiales

La respuesta global del material se determina a partir de una combinación de comportamientos metálico, dieléctrico y refractivo, controlada mediante los parámetros `metallic` y `opacity`. Como se ilustra en la Figura 4.11, primero se interpola entre el modelo dieléctrico y el refractivo según la opacidad, y luego se mezcla el resultado con el componente metálico.

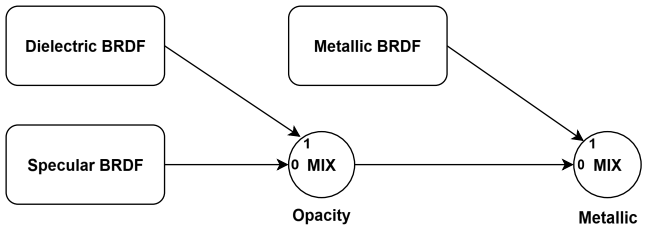


Figura 4.11: Diagrama de flujo de la mezcla de materiales. La opacidad pondera entre el modelo dieléctrico y el refractivo, y el resultado se mezcla con el modelo metálico.

4.8.2. Componente Difuso

Se implementó el componente difuso no-Lambertiano de Burley (2012), que introduce correcciones en ángulos rasantes (oscurecimiento tipo Fresnel) y retro-reflexión dependiente de la rugosidad. El término difuso base se expresa como:

$$\begin{aligned} f_d &= \frac{\texttt{baseColor}}{\pi}\left(1+(F_{D90}-1)(1-\cos\theta_l)^5\right)\left(1+(F_{D90}-1)(1-\cos\theta_v)^5\right) \\ F_{D90} &= 0{,}5+2\,\texttt{roughness}\,\cos^2\theta_d \end{aligned} \tag{4.15}$$

Para aproximar *subsurface scattering* (modelo 2012), se utiliza el término difuso alternativo:

$$\begin{aligned} f_{ss} &= \frac{\texttt{baseColor}}{\pi}\,1{,}25\left[F_{ss}\left(\frac{1}{\cos\theta_l+\cos\theta_v}-\frac{1}{2}\right)+\frac{1}{2}\right], \\ F_{ss} &= \left(1+(F_{ss90}-1)(1-\cos\theta_l)^5\right)\left(1+(F_{ss90}-1)(1-\cos\theta_v)^5\right), \\ F_{ss90} &= \texttt{roughness}\,\cos^2\theta_d \end{aligned} \tag{4.16}$$

Finalmente, el componente difuso se interpola mediante el parámetro `subsurface`:

$$f_{\text{diff}} = (1-\texttt{subsurface})\,f_d + \texttt{subsurface}\,f_{ss}. \tag{4.17}$$

Además, se suma el término *sheen*, controlado por `sheen` y `sheenTint`, para modelar brillo rasante adicional en materiales tipo tela.

4.8.3. Lóbulo Especular (Reflexión y Refracción)

El componente especular se basa en el modelo de microfacetas GGX (Trowbridge-Reitz), tanto para reflexión como para refracción:

- **Distribución (D):** Se utiliza la formulación anisotrópica de GGX. La función de distribución de normales (NDF) depende de las rugosidades independientes α_x, α_y alineadas con los ejes del plano tangente:

$$D(h) = \frac{1}{\pi \alpha_x \alpha_y \left(\left(\frac{h \cdot T}{\alpha_x} \right)^2 + \left(\frac{h \cdot B}{\alpha_y} \right)^2 + (n \cdot h)^2 \right)^2} \tag{4.18}$$

  donde T y B son los vectores tangente y bitangente, n es la normal macroscópica y h es el vector medio (half-vector).

- **Componente Geométrico (G):** Se implementa la función de ensombrecimiento y enmascaramiento de Smith separable, adaptada para la distribución GGX anisotrópica.

- **Fresnel (F):**

  - **Dieléctricos**: Se utiliza la ecuación completa de Fresnel 2.5 parametrizada directamente por el índice de refracción (`ior`). A diferencia del modelo artístico de Disney, que remapea el parámetro `specular` a un rango de IOR acotado, se optó por utilizar el `ior` físico directamente en lugar del parámetro `specular` (Burley, 2012).
  - **Metales**: Se utiliza la aproximación de Schlick 2.6, tintada con `baseColor`.

4.8.4. Anisotropía

La anisotropía permite modelar superficies con microestructuras orientadas, como el metal cepillado, estirando el reflejo especular en una dirección. Este comportamiento se controla mediante el parámetro `anisotropic`, el cual distorsiona la rugosidad isotrópica base ($\alpha = \text{roughness}^2$) separándola en dos componentes ortogonales:

$$\begin{aligned} \alpha &= \text{roughness}^2 \\ \text{aspect} &= \sqrt{1 - 0{,}9 \times \text{anisotropic}} \\ \alpha_x &= \alpha / \text{aspect} \\ \alpha_y &= \alpha \times \text{aspect} \end{aligned}$$

Para aplicar esta distorsión, es necesario definir un sistema de coordenadas tangente en cada punto de la superficie. Nuestra implementación permite especificar la dirección de la anisotropía mediante texturas o, en el caso de las esferas con primitivas personalizadas, también se puede hacer mediante un vector global. El shader construye el marco de referencia TBN (Tangente, Binormal, Normal) en el punto de impacto, alineando el eje Tangente con la dirección T_{aniso} definida en el material.

4.8.5. Mapeo de Texturas

Nuestra implementación permite controlar la mayoría de los parámetros del modelo BSDF mediante texturas. Además del color base (*Albedo*), se implementó soporte para mapas de `Metallic`, `Roughness`, `Opacity`, `Anisotropic` y `Masking`.

La gestión de estas texturas presenta particularidades dependiendo del tipo de geometría y del algoritmo de trazado, especialmente en lo que respecta a la definición de la dirección anisotrópica y la optimización del recorte (masking).

Mientras que el parámetro escalar `anisotropic` define la magnitud del estiramiento del reflejo, la orientación del mismo puede ser controlada píxel a píxel:

- En las **mallas**, se muestrea la textura utilizando las coordenadas UV interpoladas de los vértices.
- En las **esferas con primitivas personalizadas**, al no poseer vértices, el shader calcula analíticamente las coordenadas UV basándose en la posición del impacto relativa al centro de la esfera.

El *masking* permite recortar la geometría mediante una textura de transparencia (o canal alfa), descartando fragmentos de la superficie para generar siluetas complejas sobre geometría simple. Esta técnica se utiliza comúnmente para representar detalles finos —como hojas o cadenas— sin necesidad de modelar explícitamente su geometría. En la figura 4.12 se muestra un ejemplo donde el *masking* se emplea para definir los contornos de hojas y eslabones de una cadena.

En nuestra implementación, la evaluación del *masking* se realiza en el shader de *Any Hit*. Si el valor de la textura es inferior a un umbral, se ignora la intersección actual para continuar trazando el rayo a través de la escena.

4.9. Trazado de Caminos Inverso

El Trazado de Caminos Inverso (BPT) constituye la forma más básica de trazado de caminos y sirve como punto de partida para las implementaciones presentadas posteriormente. Este algoritmo simula el transporte de la luz generando caminos que se propagan desde la cámara hacia la escena, acumulando las contribuciones de las superficies que intersecan hasta llegar a una fuente emisora o abandonar el entorno.

Figura 4.12: Ejemplo de uso de *masking* en la escena *Sponza*. La textura de transparencia permite representar la forma detallada de las cadenas y las hojas de las plantas utilizando polígonos simples, descartando durante el trazado los fragmentos correspondientes a las regiones transparentes.

Algoritmo 3 Trazado de Caminos Inverso (BPT)

for cada píxel **do**
  $\alpha = 1$
  Generar rayo primario desde la cámara
  for profundidad $= 0 \ldots$ profundidad_máxima **do**
    Trazar rayo
    if impacta una luz **then**
      $color\_pixel = color\_pixel + L_d \cdot \alpha$ ▷ Acumular emisión
      break
    if no hay intersección **then**
      break
    Evaluar BSDF en el punto de intersección x:
    $\alpha = \alpha \cdot BSDF$ ▷ Actualizar contribución del camino

La implementación sigue directamente el modelo teórico descrito en la Sección 2.3.2, pero se desarrolla de forma iterativa para adaptarse al modelo de ejecución en GPU. Esta estructura evita el uso de recursión y permite mantener una única invocación del ducto por fotograma, aprovechando el paralelismo de las GPU.

El Pseudocódigo 3 resume la versión implementada del algoritmo.

4.10. Trazado de Caminos Inverso con Estimador del Siguiente Evento

El Trazado de Caminos Inverso con Estimador del Siguiente Evento (BPT-NEE) extiende el algoritmo anterior incorporando el cálculo explícito de la iluminación directa en cada rebote difuso. Como se menciona en la Sección 2.3.3, esta técnica introduce una segunda estrategia de muestreo que conecta los puntos del camino con las fuentes de luz visibles mediante rayos de sombra.

La implementación mantiene la misma estructura iterativa del BPT, agregando una función para el cálculo de la iluminación directa que se ejecuta únicamente en los rebotes difusos. El Pseudocódigo 4 resume la versión implementada del algoritmo.

Algoritmo 4 Trazado de Caminos Inverso con Estimación del Siguiente Evento (BPT-NEE)

```
for cada píxel do
    α = 1
    Generar rayo primario desde la cámara
    for profundidad = 0 ... profundidad_máxima do
        Trazar rayo
        if impacta una luz then
            color_pixel = color_pixel + L_e · α          ▷ Acumular emisión
            break
        if no hay intersección then
            break
        if x es rebote difuso then
            L_d = ILUMINACION_DIRECTA(x)
            color_pixel = color_pixel + L_d · α
        Evaluar BSDF en el punto de intersección x:
        α = α · BSDF                                     ▷ Actualizar contribución del camino
```

El cálculo de la iluminación directa, detallado en el Pseudocódigo 5, evalúa la contribución de todas las luces presentes en la escena. Para cada luz, se muestrea un punto y aleatorio sobre su superficie y se lanza un rayo de sombra desde el punto de intersección x hacia el punto muestreado y. Si el camino no está ocluido, se evalúan el BSDF en x, la radiancia emitida por la luz $L_e(y)$ y el término geométrico $G(x, y)$. La contribución resultante se acumula ponderada por la probabilidad de muestreo del punto $p_A(y)$, siguiendo a Szirmay-Kalos (1999).

Nuestra implementación sigue el enfoque de Szirmay-Kalos: la iluminación directa se estima mediante *light source sampling* como complemento al muestreo de la BSDF realizado en cada iteración. Una alternativa más robusta consiste en combinar los subcaminos generados por BPT con un punto muestreado en alguna fuente de luz utilizando *Multiple Importance Sampling* (MIS), tal como

se discute en Szirmay-Kalos y en implementaciones de referencia como PBRT. Este enfoque es equivalente a considerar las contribuciones correspondientes a los casos $s = 1$ y $s = 0$ dentro del marco de BDPT. La incorporación de esta técnica se deja como trabajo futuro.

Algoritmo 5 Iluminacion directa por NEE en un punto x

function ILUMINACION_DIRECTA(x)
  $L_d = 0$
  for cada luz en la escena **do**
    Elegir un punto aleatorio y en la luz
    Lanzar rayo de sombra $x \to y$
    if visible **then**
      Evaluar BSDF en x
      $L_d = L_d + BSDF \cdot L_e \cdot G(x, y) \cdot \dfrac{1}{p_A(y)}$
  return L_d

4.11. Trazado de Caminos Bidireccional

El Trazado de Caminos Bidireccional (BDPT) combina caminos generados desde la cámara y desde las luces para estimar la Ecuación de Rendering mediante la unión de subcaminos. Su principal ventaja radica en que permite cubrir regiones del espacio de caminos que serían difíciles de alcanzar si se muestreara únicamente desde uno de los extremos.

En esta sección se describe la implementación práctica del algoritmo, adaptada para su ejecución en GPU mediante Vulkan, y se detallan los principales aspectos del cálculo del peso de *Multiple Importance Sampling* (MIS).

4.11.1. Generación de subcaminos

El algoritmo construye dos subcaminos de manera independiente:

- **Subcamino de la cámara:** comienza en el lente y se propaga hacia la escena, acumulando vértices en las superficies que interseca. (Trazado de Caminos Inverso)
- **Subcamino de la luz:** comienza en una fuente de luz seleccionada aleatoriamente y se propaga hacia la escena siguiendo las reflexiones del material. (Trazado de caminos hacia adelante)

Cada vértice almacena la información necesaria para conectar ambos subcaminos: posición, normal, PDFs de muestreo hacia adelante y hacia atrás, throughput acumulado y un indicador que determina si el evento es totalmente especular. Los vértices especulares se generan cuando la rugosidad del material es menor que un umbral $\delta = 1 \times 10^{-4}$.

Figura 4.13: Técnicas generadas por BDPT para caminos de largo máximo $k = 6$ sin aplicar el peso MIS.

Cada combinación válida de longitudes (s, t) representa una técnica de muestreo distinta, según la cantidad de vértices de la luz y la cantidad de vértices de la cámara que se utilicen, en la Figura 4.13 se puede ver la salida de cada técnica para un camino de largo máximo $k = 6$. Una vez generadas las contribuciones, se combinan utilizando Multiple Importance Sampling (MIS), en la Figura 4.14 se puede ver la salida de cada técnica luego de aplicar MIS.

El Pseudocódigo 6 resume la versión implementada del algoritmo.

Algoritmo 6 Trazado de Caminos Bidireccional (BDPT)

for cada píxel **do**
  Generar subcamino de la cámara $\{y_0, \ldots, y_{t-1}\}$
  Generar subcamino de la luz $\{x_0, \ldots, x_{s-1}\}$
  for cada $1 \leq k \leq$ profundidad_maxima **do**
    numero_vertices $= k + 1$
    for cada $0 \leq s \leq$ numero_vertices **do**
      $t =$ numero_vertices $- s$
      if la conexión entre x_{s-1} y y_{t-1} es visible **then**
        $C_{s,t} = \alpha_s^L \, c_{s,t} \, \alpha_t^E$ ▷ Contribución de la técnica (s,t) 2.3.4
        $w_{s,t} = MIS(s, t)$ ▷ Peso de Multiple Importance Sampling
        $color\_pixel = color\_pixel + w_{s,t} \cdot C_{s,t}$

Figura 4.14: Técnicas generadas por BDPT para caminos de largo máximo $k = 6$ luego de aplicar el peso MIS.

4.11.2. Cálculo del peso de Multiple Importance Sampling

A continuación, se detalla cómo se computa el peso $w_{s,t}$ en nuestra implementación, siguiendo la heurística de potencia presentada en la Sección 2.3.4.

Para una conexión de longitudes (s, t), definimos $k = s + t - 1$. El peso se evalúa como

$$w_{s,t} = \frac{1}{\sum_{i=0}^{k} \left(\frac{p_i}{p_s}\right)^2}, \tag{4.19}$$

donde p_s es la probabilidad de la técnica activa (la que conecta x_{s-1} con y_{t-1}), y p_i son las probabilidades de las técnicas alternativas que podrían generar el mismo camino.

La sumatoria se divide en dos barridos: uno hacia el vértice de la luz (para $i < s$) y otro hacia el vértice de la cámara (para $i > s$). Finalmente, se suman ambos resultados y el término $p_s/p_s = 1$.

Barrido hacia la luz

Para calcular los cocientes p_0/p_s, p_1/p_s, $\ldots$, p_{s-1}/p_s, se recorre de forma iterativa la Ecuación 4.20 desde $i = s - 1$ hasta $i = 0$. En cada paso de la recursión, se obtiene un nuevo cociente a partir del siguiente. El Algoritmo 7 muestra cómo calcular la suma de la Ecuación 4.19, utilizando en cada paso la Ecuación 2.34.

$$\frac{p_i}{p_s} = \frac{p_i}{p_{i+1}} \frac{p_{i+1}}{p_s} \tag{4.20}$$

Algoritmo 7 Cálculo de la suma de los cocientes del MIS cuando $i < s$

function MIS(s, t)
  $cociente = 1$
  $suma = 0$
  for $i = s - 1$ **hasta** 0 **do**
    if $i = 0$ **then**
      $cociente = cociente \cdot \frac{P_\sigma(x_1 \to x_0)\, G(x_1 \leftrightarrow x_0)}{P_A(x_0)}$ ▷ p_0/p_1
      if x_0 *es rebote especular* **then**
        continuar
    else if $i < k$ **then**
      $cociente = cociente \cdot \frac{P_\sigma(x_{i+1} \to x_i)\, G(x_{i+1} \leftrightarrow x_i)}{P_\sigma(x_{i-1} \to x_i)\, G(x_{i-1} \leftrightarrow x_i)}$ ▷ p_i/p_{i+1}
      if x_{i-1} y/o x_i *son rebotes especular* **then**
        continuar
    else
      $cociente = cociente \cdot \frac{P_A(x_k)}{P_\sigma(x_{k-1} \to x_k)\, G(x_{k-1} \leftrightarrow x_k)}$
      ▷ p_k/p_{k+1}, cuando $t = 0$
      if x_{s-2} y/o x_{s-1} *son rebotes especular* **then**
        continuar
    $suma = suma +$ cociente^2

Barrido hacia la cámara

El cálculo de los cocientes p_{s+1}/p_s, $\ldots$, p_{k-1}/p_s, p_k/p_s es análogo al barrido hacia la luz, con la excepción de que, en este caso, se recorre de forma iterativa la Ecuación 4.21 desde $i = s + 1$ hasta $i = k + 1$

$$\frac{p_i}{p_s} = \frac{p_i}{p_{i-1}} \frac{p_{i-1}}{p_s} \tag{4.21}$$

Tratamiento de vértices especulares.

Siguiendo el tratamiento propuesto por Veach (1997, p. 314-316), cuando un vértice x_j es especular, las funciones de probabilidad asociadas a su BSDF,

$$P_\sigma(x_j \to x_{j+1}) \quad \text{y} \quad P_\sigma(x_j \to x_{j-1}),$$

se concentran en una única dirección. En estos casos, el vértice no representa un evento de muestreo con probabilidad finita, sino una reflexión o refracción perfecta; por lo tanto, las probabilidades valen cero en todo el dominio, excepto en esa dirección.

Cuando un cociente local involucra un vértice marcado como `especular`, el término correspondiente en la suma se descarta, ya que no representa una técnica de muestreo válida, como se puede ver en el Algoritmo 7.

Sin embargo, el factor sí se acumula en `cociente`, porque las probabilidades en el numerador y el denominador se cancelan entre sí.

4.11.3. Casos especiales para subcaminos cortos

Al considerar caminos que utilizan menos de dos vértices de uno de los subcaminos, surgen situaciones que requieren un manejo especial.

Caminos que no utilizan vértices del subcamino de la luz ($s = 0$)

Estas muestras ocurren cuando el último vértice del subcamino de la cámara se encuentra en la superficie de una de las luces.

Para poder modelar estos casos, las luces deben formar parte de la escena, permitiendo así la intersección de rayos con las fuentes luminosas.

Caminos que no utilizan vértices del subcamino de la cámara ($t = 0$)

Estas muestras ocurren cuando el último vértice del subcamino de la luz se encuentra en el lente de la cámara. Para ello, la cámara debe estar representada en la escena (por ejemplo, mediante un *quad*), permitiendo a los rayos intersecarse con su superficie.

Según Veach (1997, p. 340), estos casos tienen una contribución poco significativa en la mayoría de las escenas, por lo que se decidió omitir su implementación para simplificar el algoritmo.

Caminos que utilizan un sólo vértice del subcamino de la cámara ($t = 1$)

En general, para $t > 1$, la contribución de un camino que conecta la luz con la cámara puede asociarse de forma directa a un único píxel. En cambio, cuando $t = 1$, el subcamino de la cámara está formado únicamente por el vértice y_0, correspondiente a un punto en el lente. En esta situación, el píxel afectado no queda determinado de manera inmediata, sino que depende de la proyección del punto x_{s-1} a través de la cámara. Por ello, distintas conexiones entre x_{s-1} y y_0 pueden contribuir a píxeles diferentes, como se ilustra en la Figura 4.15.

El algoritmo se ejecuta de forma independiente en múltiples hilos, uno por cada píxel. En el caso $t = 1$, el píxel afectado casi siempre será distinto del píxel correspondiente al hilo. Debido a esto, es necesario acumular las contribuciones de estas muestras en un espacio de memoria auxiliar (en nuestro caso, en una imagen adicional).

Luego de terminar una iteración del algoritmo, es necesario unir los dos resultados para obtener la imagen final. Como cada píxel de la imagen generada por el algoritmo se corresponde con el promedio de los resultados de cada iteración, se pueden sumar las contribuciones de la imagen auxiliar a este promedio, dividiéndolas entre el total de iteraciones. El color que se obtiene luego de unir las dos imágenes se escribe en la imagen principal y los valores de la imagen auxiliar se reinician a cero.

Es importante notar que, durante un mismo paso, múltiples puntos pueden afectar el mismo píxel de la imagen auxiliar. Para evitar condiciones de carrera al actualizar estos valores, se utilizan operaciones atómicas.

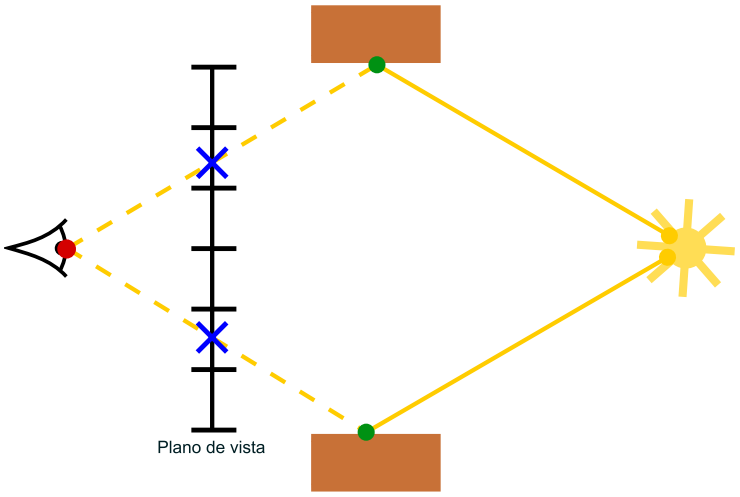


Figura 4.15: Diagrama bidimensional de la proyección de dos puntos distintos en el camino de la luz hacia la cámara para el caso $t = 1$ (punto rojo en la cámara) y $s = 2$ (puntos verdes de cada camino). La pantalla está representada por la línea vertical negra, y cada división representa un píxel. Se observa como cada uno de los caminos afecta a un píxel distinto de la pantalla.

Existen varias extensiones de Vulkan que permiten operaciones atómicas, dependiendo del tipo de estructura de datos (imagen, *buffer*, etc.) y del tipo de dato. Para operaciones atómicas sobre imágenes en punto flotante, se emplea la extensión *VK_EXT_shader_atomic_float*, compatible con tarjetas de todos los proveedores.

Dado que esta extensión solo permite imágenes de un canal, se adaptó la imagen para que fuera cuatro veces más ancha, colocando los cuatro componentes *RGBA* de cada píxel en posiciones contiguas. El acceso a cada componente se realiza mediante un *offset* horizontal:

$R = (pixel_x \times 4, pixel_y)$

$G = (pixel_x \times 4 + 1, pixel_y)$

$B = (pixel_x \times 4 + 2, pixel_y)$

$A = (pixel_x \times 4 + 3, pixel_y)$

Entre las operaciones atómicas disponibles (*AtomicAdd*, *AtomicExchange*, *AtomicLoad* y *AtomicStore*), se utiliza *AtomicAdd* para acumular los colores de todas las contribuciones que afectan a un mismo píxel. El canal *alpha* registra la cantidad de escrituras, aunque se utiliza únicamente para descartar píxeles sin contribuciones.

La sintaxis de la operación es:

```
imageAtomicAdd(imagen, pixel, valor);
```

4.11.4. Problema con tarjetas gráficas AMD

Al trabajar con tarjetas gráficas AMD, se presentaron problemas con el uso de la extensión *VK_EXT_shader_atomic_float* que no ocurrían en tarjetas NVIDIA.

La compilación de cualquier *shader* que contuviera una operación atómica sobre una imagen en punto flotante generaba un error, debido a que el controlador de AMD no implementa la función *imageAtomicAdd* para este tipo de datos.

Para resolver este inconveniente, se optó por utilizar una imagen con formato entero, ya que las operaciones atómicas sobre enteros no presentan este problema. Los valores en punto flotante se convierten a enteros multiplicándolos por una constante relativamente grande al momento de escribirlos, y se dividen por la misma constante al leerlos.

El valor de esta constante determina la precisión de la representación: si es lo suficientemente grande, la pérdida de precisión es despreciable. No obstante, dado que los resultados pueden superar el valor 1 (por ejemplo, en regiones cercanas a fuentes de alta intensidad), la constante debe ser menor que el rango máximo representable por el tipo entero utilizado, para evitar errores de desbordamiento. En nuestro caso, se utilizó un factor de escala igual a 2^{20}.

Capítulo 5

Experimentación

5.1. Metodología de Pruebas

5.1.1. Entorno de Pruebas

El análisis experimental se llevó a cabo utilizando dos tarjetas gráficas de gama media representativas de las arquitecturas de los fabricantes más utilizados actualmente: NVIDIA GeForce RTX 4070 (Ada Lovelace) con un procesador Intel Core i9-11900F, y AMD Radeon RX 7600 (RDNA 3) con un procesador Intel Core i7-7700. La elección de estos dispositivos permite contrastar diferentes enfoques de diseño para la aceleración de trazado de rayos.

NVIDIA GeForce RTX 4070

Basada en la arquitectura *Ada Lovelace* (NVIDIA, 2023), esta tarjeta utiliza un diseño de ejecución SIMT (*Single Instruction, Multiple Threads*) donde los hilos se agrupan en bloques de 32 denominados Warps.

- **Cómputo General:** Cuenta con 5888 núcleos CUDA. Estos núcleos gestionan la lógica de los shaders y el cálculo de iluminación.
- **Aceleración de Rayos:** Dispone de 46 RT Cores de 3ª generación. Estos son unidades de función fija independientes que asumen la carga completa del recorrido de la jerarquía de volúmenes envolventes (BVH) y las pruebas de intersección rayo-triángulo. Al ser unidades dedicadas, permiten que el sombreador (CUDA Core) continúe con otras tareas o lance nuevos rayos mientras el hardware resuelve la intersección de forma asíncrona.

AMD Radeon RX 7600

Basada en la arquitectura *RDNA 3* (Kramer, 2023), utiliza un modelo de ejecución que agrupa los hilos en Wavefronts (típicamente de 32 o 64 hilos). Su

enfoque para el trazado de rayos es híbrido, integrando la aceleración dentro de las unidades de cómputo generales.

- **Cómputo General:** Dispone de 2048 Stream Processors. A diferencia de NVIDIA, la arquitectura RDNA 3 permite la emisión dual (*Dual Issue*): pudiendo ejecutar dos operaciones del shader en el mismo ciclo, en lugar de una sola, siempre que no haya dependencias entre ellas. Esto mejora el aprovechamiento de las unidades de cómputo, pero solo se activa bajo condiciones específicas.
- **Aceleración de Rayos:** Cuenta con 32 Ray Accelerators (uno por unidad de cómputo). A diferencia de los RT Cores, estos aceleradores gestionan eficientemente la intersección de cajas y triángulos, pero delegan la lógica de control y el recorrido del BVH a los *Stream Processors*. Esto implica que el trazado de rayos comparte recursos con la ejecución de los shaders.

Las especificaciones técnicas relevantes de los dispositivos utilizados se detallan a continuación:

Característica	**NVIDIA RTX 4070**	**AMD RX 7600**
Arquitectura	Ada Lovelace	RDNA 3
VRAM	12 GB GDDR6X	8 GB GDDR6
Unidades de Sombreado	5888 (CUDA Cores)	2048 (Stream Proc.)
Aceleradores de Rayos	46 (RT Cores)	32 (Ray Accel.)
Potencia de Cómputo (FP32)	$\approx$ 29 TFLOPS	$\approx$ 21.7 TFLOPS
Ancho de Banda de Memoria	504 GB/s	288 GB/s

Tabla 5.1: Comparación de especificaciones técnicas entre las tarjetas gráficas utilizadas.

Dada la "superioridad" en las especificaciones técnicas de la RTX 4070, se espera un rendimiento mayor respecto a la RX 7600. Por este motivo, se utilizara la tarjeta de NVIDIA como la plataforma principal para el análisis detallado de la convergencia de los algoritmos. Sin embargo, se contrastarán los resultados con la RX 7600 en cada prueba para analizar las diferencias de comportamiento de cada arquitectura en diferentes contextos.

5.1.2. Configuración de Renderizado

Para aislar el desempeño de los algoritmos de transporte de luz, se fijaron los parámetros de renderizado en valores constantes para todas las pruebas, salvo que se indique explícitamente lo contrario:

- **Resolución:** 1280×720 píxeles (HD). Esta resolución permite visualizar correctamente los fenómenos presentes en la escena, sin perjudicar en gran medida el rendimiento de los algoritmos.

- **Profundidad Máxima del Camino:** 5 rebotes. Se eligió este valor como compromiso entre costo y calidad visual: permite capturar iluminación indirecta relevante y, en particular, caminos especulares que permiten visualizar cáusticas simples en las escenas evaluadas. En la sección 5.3.1 se analizará el impacto de la profundidad en el resultado de los distintos algoritmos, y se fundamentará esta elección.

- **Materiales:** Se utilizó el modelo *Disney BSDF* descrito en el capítulo anterior.

Todas las imágenes de referencia fueron generadas ejecutando los respectivos algoritmos durante 50 minutos. En el caso de Mitsuba, la imagen de referencia se generó bajo las mismas condiciones.

5.1.3. Protocolo de Medición y Métricas

Dado que las técnicas evaluadas presentan costos computacionales por muestra diferentes, una comparación basada únicamente en el número de iteraciones no resultaría representativa. Por este motivo, los resultados se analizaron principalmente en función de la evolución del error a lo largo del tiempo, lo que permite una comparación más adecuada del comportamiento práctico de cada algoritmo.

Durante la ejecución de cada prueba, el sistema capturó automáticamente el estado de la imagen y calculó las métricas de error en intervalos fijos de 0,5 segundos. Para cuantificar la calidad de la convergencia, cada imagen generada I se comparó con una imagen de referencia R, obtenida ejecutando el algoritmo con un número elevado de muestras hasta eliminar el ruido visible. Cada técnica posee su propia imagen de referencia R, ya que los distintos algoritmos no generan exactamente la misma imagen. En consecuencia, todas las métricas convergerán eventualmente hacia su valor ideal.

Se utilizaron las siguientes métricas:

- **MSE (Mean Squared Error):** Calcula el promedio de los errores cuadrados entre los píxeles de la imagen generada y la referencia. Para una imagen con N píxeles:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (I_i - R_i)^2 \tag{5.1}$$

  Esta métrica penaliza fuertemente los errores grandes debido a la elevación al cuadrado. En el contexto de Monte Carlo Ray Tracing, el MSE es muy sensible a los "fireflies" (píxeles con energía extremadamente alta), lo que puede generar picos de errores locales aunque la imagen general parezca converger.

- **RMSE (Root Mean Squared Error):** Es la raíz cuadrada del MSE, lo que devuelve el error a la misma escala de unidades que los valores de

los píxeles (radiancia).

$$RMSE = \sqrt{MSE} \tag{5.2}$$

Es útil para interpretar la magnitud promedio del error de manera más intuitiva que el MSE.

- **SSIM (Structural Similarity Index):** A diferencia de las métricas anteriores que se basan en diferencias de píxeles absolutos, el SSIM es una métrica de percepción que evalúa la degradación de la calidad estructural, considerando cambios en la luminancia, el contraste y la estructura.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.3}$$

donde μ representa el promedio, σ^2 la varianza y σ_{xy} la covarianza de las ventanas locales x (imagen generada) e y (referencia). Las constantes c_1 y c_2 se introducen para evitar inestabilidades numéricas cuando los denominadores se aproximan a cero, y se definen como:

$$c_1 = (k_1 L)^2, \quad c_2 = (k_2 L)^2,$$

donde L es el rango dinámico máximo de la imagen (por ejemplo, 1 si está normalizada o 255 en imágenes de 8 bits), y típicamente $k_1 = 0{,}01$ y $k_2 = 0{,}03$.

El SSIM varía entre -1 y 1, donde 1 indica identidad perfecta. Esta métrica es especialmente valiosa para evaluar la convergencia visual, ya que es menos sensible al ruido de alta frecuencia, que el ojo humano tiende a ignorar más fácilmente que los errores estructurales.

- **ꟻLIP**:

ꟻLIP es una métrica de comparación de imágenes basada en un modelo del sistema visual humano descrita en Ray Tracing Gems (2021, p. 301–320). A diferencia de una comparación estrictamente píxel a píxel, la métrica aplica transformaciones que permiten identificar diferencias visualmente relevantes entre una imagen de referencia y una imagen evaluada. Su salida consiste en un mapa de error en el cual cada píxel almacena la magnitud del error ꟻLIP entre ambas imágenes.

El mapa de error se visualiza mediante una escala de colores, en la cual los colores fríos (azul y morado) representan errores de baja magnitud, mientras que los colores cálidos (amarillo y rojo) indican errores de mayor magnitud. Esta representación permite identificar de forma inmediata las regiones con mayores discrepancias entre las imágenes comparadas.

Un ejemplo de la aplicación de la métrica ꟻLIP se muestra en la Figura 5.1, donde se presentan la imagen de referencia, la imagen evaluada y el mapa de error correspondiente (Marrs y cols., 2021, p. 309). Esta Figura permite observar cómo el mapa de error destaca las regiones con mayores

discrepancias entre las imágenes, facilitando la localización y cuantificación de las diferencias visuales.

Figura 5.1: Imagen de referencia, imagen evaluada y mapa de error ꟻLIP, extraídas de Ray Tracing Gems (2021, p. 303). El mapa de error permite identificar la magnitud y localización de las diferencias visuales entre ambas imágenes.

Los valores del mapa de error representan diferencias visuales, no métricas físicas. Por lo tanto, áreas con valores elevados no necesariamente indican una degradación de la calidad del renderizado, sino diferencias más notorias según el modelo utilizado.

5.2. Escenas de prueba

Para evaluar el desempeño de los algoritmos implementados, se seleccionó un conjunto de cuatro escenas que presentan desafíos geométricos y de iluminación variados. Estas escenas permiten aislar y analizar comportamientos específicos, desde la convergencia básica en entornos controlados hasta la resolución de caminos de luz complejos en escenarios arquitectónicos realistas.

5.2.1. CornellBox

Esta escena es una adaptación de la clásica *Cornell Box* (ver Figura 5.2). Se compone de una única fuente de luz de área en el techo y dos esferas (utilizando la primitiva geométrica que se presentó en 4.7.1): una con un material reflectante y la otra con un material refractivo.

La escena *CornellBox* se utiliza como caso de prueba base debido a la ausencia de complejidades geométricas o materiales avanzados. Esto permite aislar el comportamiento de los algoritmos de transporte de luz y analizar su convergencia y nivel de ruido en un entorno controlado. La inclusión de una esfera

reflectante y una esfera refractiva permite además observar el comportamiento de estos materiales bajo iluminación indirecta en una configuración sencilla.

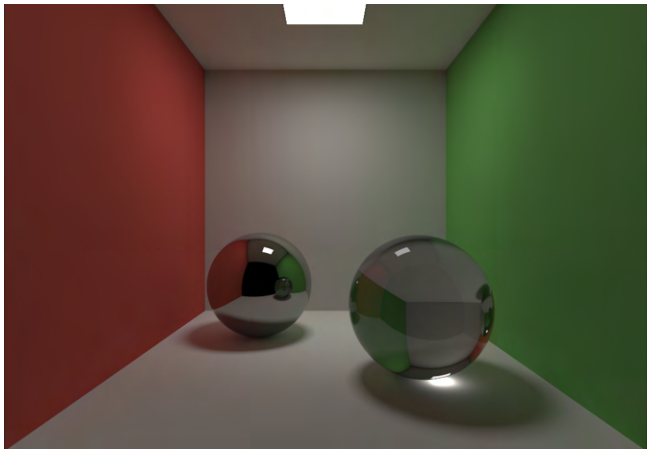


Figura 5.2: Escena *CornellBox*, ejecutada usando BDPT durante 1000 segundos.

5.2.2. Lámparas de Veach

Esta escena es una recreación del escenario de prueba propuesto por Eric Veach en su tesis doctoral (Veach, 1997). La composición presenta dos lámparas que contienen fuentes de luz de área muy pequeñas, simulando bombillas, y una mesa sobre la cual se ubica un objeto transparente con forma de huevo.

Figura 5.3: Escena VeachLamps, ejecutada usando BPT con NEE durante 1000 segundos.

El principal desafío de esta escena radica en la dificultad para encontrar

caminos válidos que conecten la cámara con las fuentes de luz, debido a su tamaño reducido y a que se encuentran parcialmente ocluidas por la estructura de las lámparas. Estas características hacen que los métodos basados únicamente en el trazado de caminos desde la cámara presenten una baja probabilidad de muestrear contribuciones relevantes de forma eficiente.

Por este motivo, esta escena resulta adecuada para evaluar distintas estrategias de muestreo del transporte de luz en escenarios con fuentes pequeñas y ocluidas. En particular, permite analizar el efecto de técnicas que incorporan muestreo explícito de las luces o la generación de caminos desde las fuentes luminosas.

Adicionalmente, tener un objeto transparente permite analizar la capacidad de los algoritmos para resolver efectos de iluminación complejos, como las cáusticas generadas por la refracción de la luz emitida por ambas lámparas.

5.2.3. Dormitorio

Esta escena representa un dormitorio iluminado por luz natural proveniente de dos ventanas laterales, donde se sitúan grandes luces de área. El entorno, creado originalmente por Rui Teixeira (*SlykDrako*), presenta una geometría significativamente más compleja que las escenas anteriores, con múltiples superficies que interactúan mediante reflexión especular y rugosa, como suelos de madera, espejos y mobiliario.

Figura 5.4: Escena *Dormitorio*, ejecutada usando BDPT durante 10000 segundos.

El gran tamaño de las fuentes de luz facilita su muestreo desde distintos puntos de la escena, reduciendo la dificultad asociada a la estimación de la iluminación directa. En este contexto, la escena resulta adecuada para analizar el comportamiento de los algoritmos en presencia de geometría compleja y múltiples interacciones de iluminación indirecta, sin que la localización de las fuentes luminosas represente el principal desafío del problema.

5.2.4. Sponza

La representación del atrio del Palacio Sponza (ubicado en Dubrovnik, Croacia) es una de las escenas más utilizadas en la investigación en gráficos por computadora para la evaluación de técnicas de iluminación global. La versión utilizada corresponde al modelo actualizado por Frank Meinl (Crytek), sobre el cual se modificaron las propiedades de los materiales con el objetivo de incrementar su variedad y complejidad. Para el análisis se definieron dos puntos de vista distintos: una toma general desde el atrio principal y una toma lateral desde una galería interior.

Figura 5.5: Escena *Sponza*, ejecutada usando BDPT durante 10000 segundos, desde dos puntos de vista distintos.

Esta escena introduce múltiples desafíos técnicos de forma simultánea. Por un lado, presenta una alta complejidad de materiales, incluyendo superficies difusas, materiales con mapas de rugosidad y metalicidad, así como geometría detallada mediante técnicas de enmascaramiento en elementos como cadenas y vegetación. Por otro lado, la estructura del atrio introduce una gran cantidad de superficies, oclusiones y detalles arquitectónicos que generan interacciones complejas de iluminación indirecta.

Con el objetivo de analizar el comportamiento de los algoritmos bajo distintas condiciones de iluminación y encuadre, se consideraron dos configuraciones de iluminación combinadas con ambos puntos de vista definidos. En una primera configuración, la escena se iluminó mediante un skybox, que actúa como una fuente de iluminación ambiental envolvente. En una segunda configuración, se utilizaron múltiples lámparas pequeñas distribuidas en el entorno, generando una iluminación más localizada y con mayores niveles de oclusión. Para cada una de estas configuraciones se evaluaron tanto la vista general del atrio principal como la vista lateral desde la galería interior.

Estas configuraciones permiten evaluar el desempeño de los algoritmos tanto en escenarios dominados por iluminación ambiental como en situaciones donde la iluminación es más localizada y presenta mayores niveles de oclusión, manteniendo constante la complejidad geométrica y de materiales de la escena.

5.3. Análisis de Resultados

En esta sección se mostrarán los resultados obtenidos al ejecutar las técnicas de trazado de caminos inverso (BPT), trazado de caminos inverso con estimación del siguiente evento (BPT-NEE) y trazado de caminos bidireccional (BDPT) sobre diferentes tarjetas gráficas, para cada una de las escenas mencionadas en la Sección 5.2.

Para facilitar la comparación de la convergencia, las gráficas se presentan en escala logarítmica. En SSIM, esta representación resalta diferencias cuando el valor se aproxima a 1, mientras que en RMSE resalta diferencias cuando el valor se aproxima a 0.

Además, para la visualización de los resultados se aplica una corrección gamma a las imágenes de salida de la forma $V_{\text{out}} = V_{\text{in}}^{\gamma}$, donde V_{in} es la salida del algoritmo y V_{out} el valor luego de aplicar la transformación. En particular, se utiliza ($\gamma = 1/(2{,}2)$), un valor común en algoritmos de renderizado. Esta corrección se aplica únicamente a las imágenes presentadas en el informe y no afecta el cálculo de las métricas.

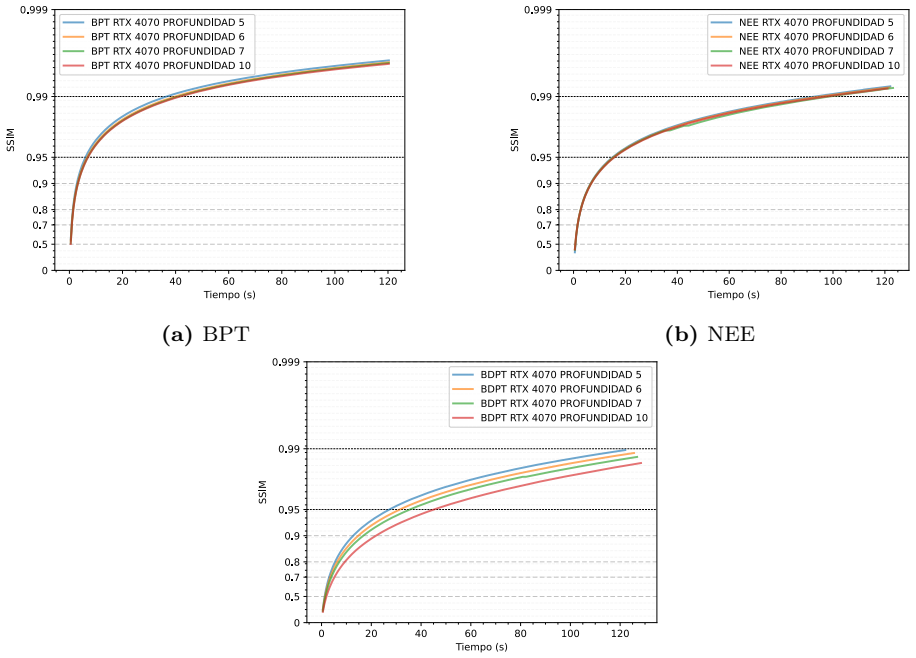
5.3.1. Comparación de Profundidad

Figura 5.6: Comparación de la escena de Veach al variar la profundidad máxima. Las imágenes fueron generadas mediante BPT durante 3600 segundos, utilizando profundidades máximas de 5, 7 y 10, respectivamente.

Se realizaron pruebas adicionales en ambas arquitecturas para evaluar el efecto de la profundidad máxima sobre la convergencia y sobre la imagen final. Las pruebas se hicieron en la escena de lámparas de Veach con profundidades 5, 6 y 7, en Sponza vista desde el atrio con profundidades 5, 6 y 7, y en Sponza vista desde una galería lateral con profundidades 5, 6, 7 y 10. Se presentan únicamente los resultados correspondientes a la RTX 4070, dado que los obtenidos con la RX 7600 son similares.

En la Figura 5.7 se observa como, en BPT y BPT-NEE, la velocidad de convergencia es independiente de la profundidad máxima. En BDPT, las profundidades mayores presentan una leve degradación en la velocidad de convergencia. Sin embargo, esta diferencia no altera la tendencia general observada entre las curvas. Por lo tanto, variar la profundidad máxima de rebotes no modificará las conclusiones del análisis de convergencia.

A nivel visual, al aumentar la profundidad máxima, las imágenes se ven más iluminadas en los tres algoritmos y los sangrados de color se vuelven más notorios, como se muestra en la Figura 5.6. Por lo tanto, aunque la convergencia varía poco, la profundidad máxima sí afecta la energía transportada y la apariencia final de la imagen. Las diferencias visuales introducidas por profundidades mayores se tendrán en cuenta más adelante en el análisis de las imágenes finales.



(a) BPT **(b)** NEE

(c) BDPT

Figura 5.7: Comparación de la convergencia (SSIM en función del tiempo) para la escena *Sponza* ejecutando en una RTX 4070. Se muestran los resultados para BPT, BPT-NEE y BDPT, considerando distintas profundidades máximas de rebotes.

5.3.2. CornellBox

Al analizar las curvas de convergencia en la tarjeta RTX 4070 se observa que los métodos BPT y BPT-NEE presentan un comportamiento casi idéntico. Esto se debe a la estructura de la escena: al tratarse de un recinto pequeño con una fuente de luz de gran tamaño y directamente expuesta, la probabilidad de que un rayo generado desde la cámara interseque la luz es elevada. En este escenario, el costo computacional adicional asociado al lanzamiento de rayos de sombra explícitos en BPT-NEE se ve compensado por la reducción de varianza que estos aportan, resultando en una eficiencia global similar a la de BPT.

En contraste, el algoritmo de trazado de caminos bidireccional presenta una mejora significativa en la velocidad de convergencia. Al comparar la métrica

SSIM (ver Figura 5.9), BDPT alcanza el valor 0,95 en 4,96 segundos, mientras que BPT y BPT-NEE requieren alrededor de 30,04 segundos y 30,6 segundos, respectivamente. Esta tendencia también se refleja en la métrica RMSE, donde BDPT reduce el error numérico a valores despreciables aproximadamente cinco veces más rápido que las otras técnicas, indicando que la mejora no es únicamente perceptual, sino también cuantitativa.

Esta diferencia de rendimiento se explica por la relación entre la naturaleza del algoritmo y la estructura de la escena. BDPT genera múltiples conexiones explícitas entre el subcamino de la cámara y el subcamino de la luz en cada muestra. En esta configuración, una gran proporción de los vértices generados desde la cámara posee línea de visión directa hacia la luz, lo que permite incorporar contribuciones de radiancia válidas de forma sistemática.

Por el contrario, en BPT y BPT-NEE la eficiencia se ve reducida debido a que la escena no es un cubo completamente cerrado, ya que se encuentra abierta en su cara frontal. Esto ocasiona que muchos rayos reboten y escapen hacia el exterior sin alcanzar una fuente de luz. Estos caminos sin contribución incrementan la varianza del estimador, introduciendo ruido que requiere un mayor tiempo de cómputo para ser reducido en comparación con las conexiones explícitas generadas por BDPT. En la Figura 5.8 se presenta una comparación entre las tres técnicas luego de 5 segundos de ejecución.

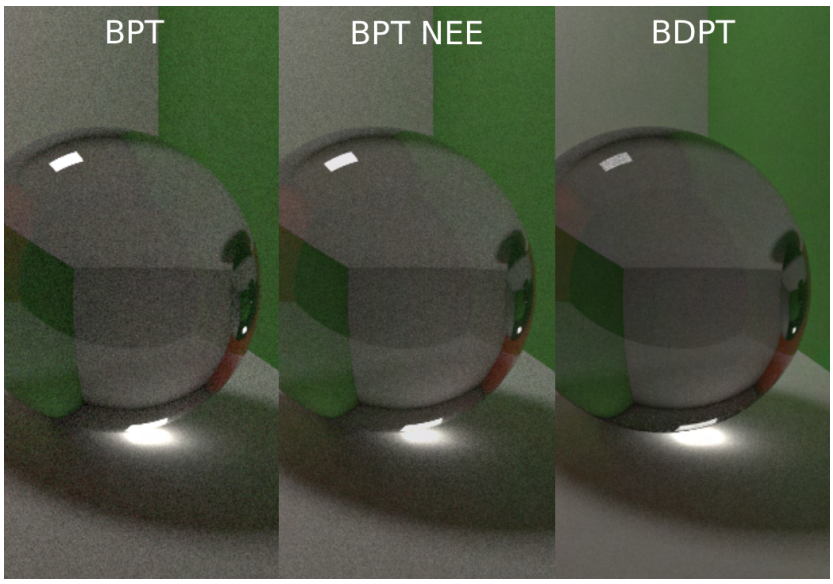


Figura 5.8: Comparación entre las distintas técnicas al renderizar la escena *Cornell-Box* luego de 5 segundos usando la RTX 4070. Se observa un ruido similar entre las técnicas BPT y BPT-NEE, mientras que BDPT presenta un ruido significativamente menor.

Comparación de Hardware

Al comparar el rendimiento entre la tarjeta NVIDIA RTX 4070 y la AMD RX 7600, se observa un comportamiento contrario a la expectativa inicial, dado que se esperaba un mayor rendimiento por parte de la tarjeta de NVIDIA, considerando su "superioridad" en términos de potencia computacional.

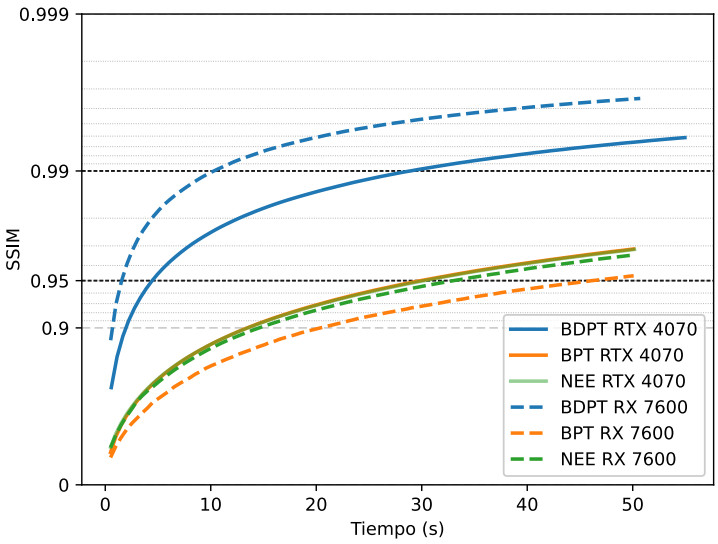


Figura 5.9: Comparación de convergencia (SSIM vs Tiempo) para la escena *CornellBox* en ambas arquitecturas. Las curvas para BPT y BPT-NEE coinciden para la RTX 4070.

Como se observa en la Figura 5.9, en las técnicas de trazado simple (BPT), la RTX 4070 es levemente mejor. Sin embargo, en BDPT, la curva de convergencia de la RTX 4070 es notablemente más lenta que en la RX 7600 para alcanzar el mismo nivel de calidad.

Esta diferencia se aprecia claramente al medir el tiempo necesario para alcanzar un umbral de calidad alto ($SSIM > 0{,}95$). La tarjeta RX 7600 logró este objetivo en tan solo 2,04 segundos, manteniendo una tasa de muestreo promedio de 46,9 FPS. Por el contrario, la RTX 4070 requirió 4,96 segundos para alcanzar la misma calidad, ejecutando a una velocidad significativamente menor de 16,4 FPS.

Atribuimos este comportamiento a la combinación de dos factores que penalizan la arquitectura de NVIDIA en esta prueba específica:

- **Primitivas Personalizadas**: Las esferas utilizan *shaders de intersección* programables (`.rint`) en lugar de la intersección de triángulos acelerada por hardware (RT Cores). Esto obliga a la GPU a alternar entre el hardware dedicado al trazado de rayos y la ejecución de shaders programables, lo que podría introducir sobrecostes y penalizar el rendimiento.

- **Divergencia Algorítmica (Branching):** BDPT introduce una lógica altamente ramificada, por lo que rayos dentro de un mismo grupo SIMT pueden tomar caminos de ejecución distintos. Este fenómeno ocurre tanto en NVIDIA como en AMD (ver Sección 2.5.3), pero su impacto depende de cómo cada arquitectura planifica y mantiene ocupadas sus unidades de ejecución cuando el grupo diverge. En nuestras mediciones, la penalización se manifiesta con mayor fuerza en la RTX 4070, lo que sugiere que en NVIDIA la pérdida de coherencia penaliza en mayor medida el rendimiento.

Nuestra hipótesis es que la caída de rendimiento observada en la RTX 4070 es el resultado combinado de estas dos penalizaciones. Para validar esta hipótesis, en la Sección 5.3.6 se compara el comportamiento de la escena utilizando esferas representadas mediante mallas triangulares y mediante primitivas personalizadas.

5.3.3. Dormitorio

(a) BDPT ejecutado por 1 s

(b) BPT ejecutado por 1 s

(c) BDPT ejecutado por 20 s

(d) BPT ejecutado por 20 s

Figura 5.10: Comparación visual entre BDPT y BPT para la escena *Dormitorio*.

La escena *Dormitorio* presenta un entorno iluminado por dos luces de gran tamaño, lo que incrementa significativamente la probabilidad de que los caminos generados desde la cámara intersequen una fuente luminosa. A diferencia de la escena anterior, la estimación de la iluminación directa no representa un desafío dominante en este caso.

Al analizar el comportamiento en la tarjeta NVIDIA RTX 4070, se observa una diferencia marcada respecto a la escena *CornellBox* (Sección 5.3.2). En este escenario, el algoritmo BPT converge de forma significativamente más rápida que BDPT, alcanzando un valor de $SSIM = 0{,}95$ en aproximadamente 6 segundos, mientras que BDPT requiere cerca de 17,5 segundos para lograr el mismo nivel de calidad.

Este comportamiento se explica por la relación entre la estructura de la escena y la naturaleza de los algoritmos. Dado que las fuentes de luz son grandes y fácilmente accesibles, la estrategia de muestreo directa utilizada por BPT resulta altamente eficiente. En contraste, el costo computacional adicional asociado a la generación de conexiones explícitas en BDPT no se ve compensado por una reducción significativa de la varianza. Asimismo, al tratarse de una escena cerrada, no se produce una pérdida relevante de rayos que escapen del entorno sin contribuir a la iluminación, como ocurría en *CornellBox*.

El análisis visual de las imágenes confirma estas observaciones. En particular, BDPT presenta una mayor dificultad para eliminar el ruido presente en reflejos especulares (ver Figura 5.10), como el reflejo de las ventanas sobre el suelo de madera. Esto se debe a que, en superficies predominantemente especulares, la técnica depende en gran medida de que el subcamino de la cámara alcance directamente la fuente de luz, de forma similar a BPT. Sin embargo, al generar un menor número de muestras por segundo debido a su mayor complejidad, el ruido asociado a estas contribuciones tarda más tiempo en desaparecer.

Comparación de Hardware

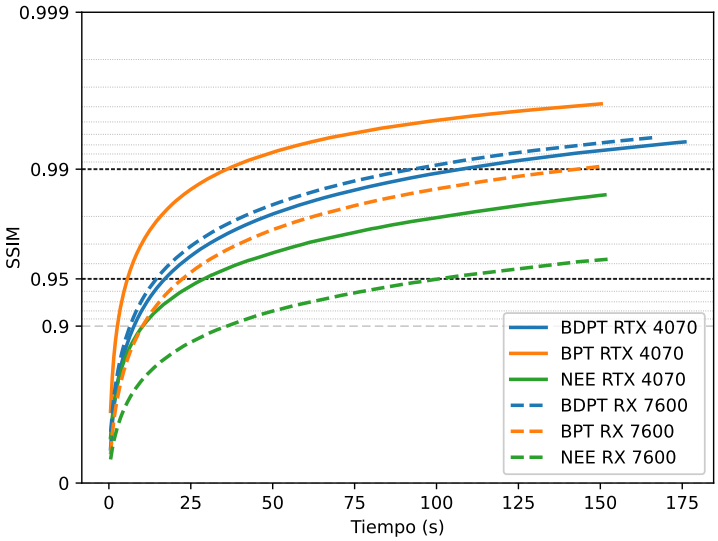


Figura 5.11: Comparación de convergencia (SSIM vs Tiempo) para la escena de *Dormitorio* en ambas tarjetas graficas.

Como se puede ver en la Figura 5.11, en la escena *Dormitorio* el comportamiento de BDPT es similar al de *CornellBox*. Aquí también la RX 7600 resulta más eficiente, alcanzando un valor de $SSIM = 0{,}95$ en aproximadamente 14,5 segundos frente a los 17,5 segundos requeridos por la RTX 4070.

Esta diferencia también se observa en la tasa de muestreo: la RTX 4070 procesa BDPT a 15,3 FPS, mientras que la RX 7600 alcanza los 18,0 FPS. Estos resultados refuerzan la hipótesis de que no solo la utilización de primitivas personalizadas influye en el rendimiento relativo entre arquitecturas, sino que la propia naturaleza del algoritmo BDPT, al introducir una lógica más ramificada y divergente, penaliza en mayor medida a la arquitectura de NVIDIA.

A diferencia de *CornellBox*, en la escena *Dormitorio* el mejor resultado se observa en BPT con la RTX 4070. Esto se debe a que los caminos generados desde la cámara tienen una probabilidad alta de alcanzar la luz. Además, al tratarse de un algoritmo con menor divergencia y sin uso de primitivas personalizadas, la RTX 4070 logra aprovechar mejor sus recursos y obtiene el mejor desempeño en esta escena. Sin embargo, en la RX 7600 BDPT continúa siendo más eficiente que BPT, lo que sugiere que el costo adicional asociado a la lógica más ramificada del algoritmo impacta menos en esta arquitectura.

5.3.4. Lámparas de Veach

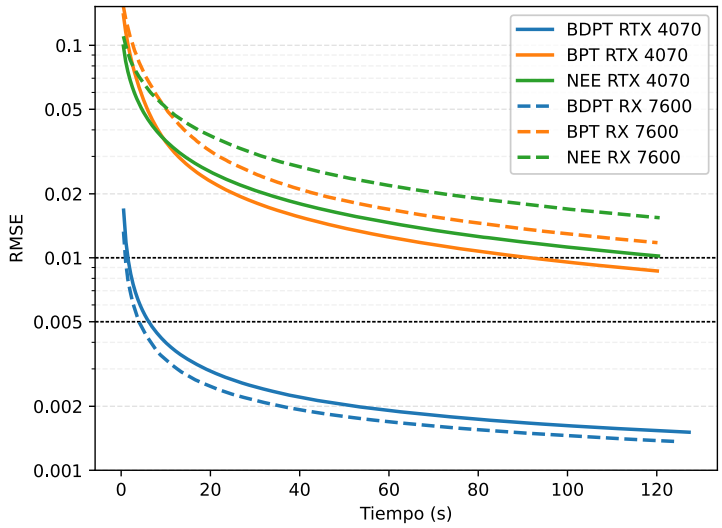


Figura 5.12: Comparación de convergencia (RMSE vs Tiempo) para la escena de *Lámparas de Veach* en ambas tarjetas gráficas.

La escena de *Lámparas de Veach* presenta condiciones complejas para los algoritmos basados en el trazado de caminos desde la cámara. Las fuentes de luz son pequeñas y se encuentran parcialmente ocluidas por la estructura de las

propias lámparas, lo que reduce significativamente la probabilidad de que un camino generado aleatoriamente desde la cámara interseque directamente una fuente luminosa.

En la Figura 5.12 se muestra la convergencia medida con RMSE en función del tiempo para ambas tarjetas gráficas.

En este contexto, se observa que el algoritmo BDPT converge de forma considerablemente más rápida que los otros métodos evaluados. Al generar subcaminos desde las fuentes de luz, BDPT puede establecer conexiones válidas con el subcamino de la cámara a través de vértices intermedios donde la oclusión es menos probable, evitando así la necesidad de construir caminos aleatorios que conecten la cámara con alguna fuente de luz, lo que ocurre con baja probabilidad en esta escena.

Al analizar el comportamiento de BPT y BPT-NEE, se observa que BPT-NEE presenta una ligera ventaja durante los primeros segundos de renderizado. Esta mejora inicial puede atribuirse a la incorporación de rayos de sombra, que añaden información explícita sobre la ubicación de las fuentes de luz. Sin embargo, este beneficio se obtiene a costa de un mayor costo computacional por muestra, lo que permite que BPT, al evaluar un mayor número de caminos en el mismo intervalo de tiempo, termine convergiendo más rápido a largo plazo.

Comparación de Hardware

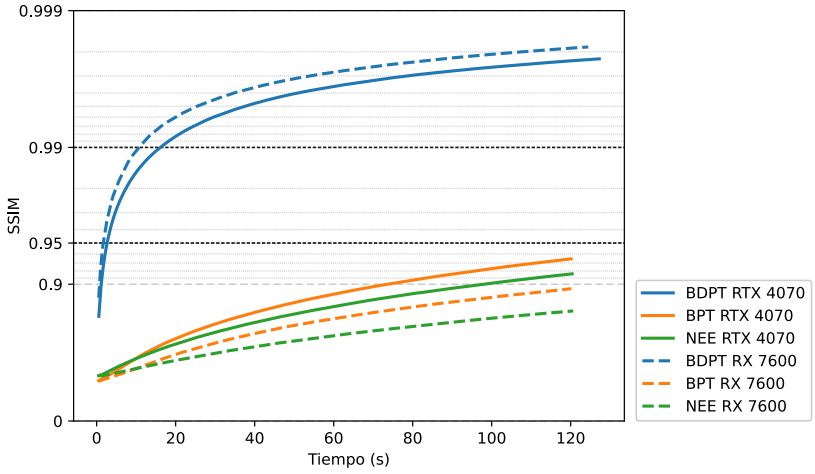


Figura 5.13: Comparación de convergencia (SSIM vs Tiempo) para la escena de *Lámparas de Veach* en ambas tarjetas gráficas.

En las primeras iteraciones de BPT-NEE, la métrica SSIM presenta un comportamiento irregular, con estancamientos iniciales en la RTX 4070 y picos hacia valores inferiores en la RX 7600. Este fenómeno se debe a la aparición de luciérnagas (píxeles de intensidad extrema, también conocidos como *fireflies*),

generadas cuando algunos rayos atraviesan las oclusiones de forma fortuita y producen contribuciones de energía extremadamente altas. A medida que avanza el proceso de renderizado, estos errores se diluyen y la convergencia se estabiliza.

Finalmente, al comparar el rendimiento entre arquitecturas (ver Figuras 5.12 y 5.13), se observa nuevamente una ventaja de la RX 7600 al ejecutar BDPT. En esta escena, la tarjeta de AMD alcanza una tasa de muestreo de 33,6 FPS, mientras que la RTX 4070 se limita a 22,6 FPS. Esta diferencia, superior al 48 %, refuerza la hipótesis de que la elevada complejidad lógica y la alta divergencia de control introducidas por BDPT penalizan de forma más pronunciada a la arquitectura de NVIDIA.

5.3.5. Sponza

En esta sección se analizan los resultados obtenidos para la escena *Sponza*, considerando las configuraciones de iluminación y los puntos de vista definidos previamente en la Sección 5.2.4.

Iluminación mediante Skybox

En la configuración iluminada por skybox, una porción significativa de la superficie emisiva se encuentra ocluida por la geometría del atrio (ver Figura 5.14). Como consecuencia, muchas de las muestras de iluminación directa generan pruebas de visibilidad fallidas, lo que impacta negativamente en la eficiencia de los algoritmos.

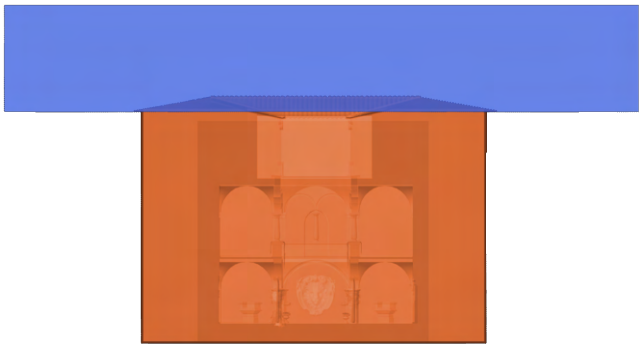
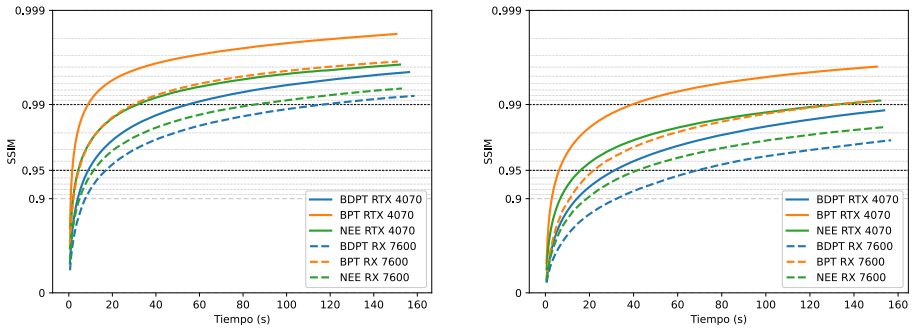


Figura 5.14: Corte longitudinal de la escena *Sponza*. La superficie en azul corresponde al skybox, mientras que la estructura del atrio se muestra en color anaranjado.

Este efecto penaliza de forma particularmente severa a BDPT como se observa en la Figura 5.15 siendo este el algoritmo que más lento converge. Además de muestrear puntos sobre la superficie emisiva del skybox, el algoritmo genera subcaminos desde la luz que, en numerosos casos, intersecan el techo del atrio, el cual no es visible desde el interior de la escena. Como resultado, estos subcaminos contienen vértices ocluidos sobre los cuales se realizan pruebas de visibilidad costosas que no aportan contribuciones útiles.

BPT-NEE también se ve afectado por la oclusión de la iluminación ambiental, ya que muchas muestras de iluminación directa fallan las pruebas de visibilidad.



(a) Escena desde el exterior.

(b) Escena desde una galería interna.

Figura 5.15: Comparación de convergencia (SSIM vs Tiempo) para la escena de *Sponza* iluminada mediante skybox, ejecutada en la tarjeta NVIDIA RTX 4070.

Al utilizar una toma lateral desde una galería interna (ver Figura 5.5), observamos resultados similares a los ya descritos para la vista del atrio, sólo que con una convergencia ligeramente peor, debido a que la cámara se encuentra en un lugar interno de la estructura, lo que dificulta el acceso a la fuente de luz. Esto muestra que, en entornos iluminados por fuentes de luz en los que gran parte de su área se encuentra ocluida, las técnicas basadas exclusivamente en el trazado de caminos desde la cámara resultan más robustas que los métodos bidireccionales, los cuales sufren severamente por la incapacidad de generar caminos de luz útiles que logren rebotar hacia el interior de la estructura.

Iluminación mediante lámparas

Al reemplazar el skybox por múltiples lámparas pequeñas distribuidas en el atrio (ver Figura 5.16), el comportamiento de los algoritmos cambia totalmente (ver Figura 5.17). A diferencia del caso anterior, donde una gran fracción de las muestras sobre la superficie emisiva del skybox conduce a subcaminos de luz poco útiles para conectar con el interior del atrio, las lámparas se ubican dentro de la escena. Esto hace que los subcaminos generados desde la luz se propaguen hacia el interior de la escena, incrementando la probabilidad de producir vértices conectables y contribuciones válidas.

Figura 5.16: Escena *Sponza* iluminada mediante lámparas internas, ejecutada usando BDPT durante 1000 segundos.

En esta configuración, BDPT mejora de forma considerable respecto al caso iluminado por skybox y se convierte en el método con la convergencia más rápida. Este comportamiento es consistente con lo observado en la escena de *Lámparas de Veach* (Sección 5.3.4), ya que la iluminación está dominada por fuentes pequeñas dentro de la escena, para las cuales la generación de subcaminos desde la luz y las conexiones explícitas con el subcamino de la cámara aumentan significativamente la probabilidad de encontrar caminos de transporte relevantes.

En la vista general del atrio, BPT comienza con una convergencia muy lenta, generando inicialmente una imagen casi negra debido a la baja probabilidad de intersecar aleatoriamente una lámpara pequeña. Sin embargo, a medida que avanza el tiempo de renderizado, BPT converge más rápidamente que BPT-NEE. Esto se debe a que su menor costo computacional por muestra le permite evaluar un mayor número de caminos en el mismo intervalo de tiempo, reproduciendo el comportamiento ya observado en la Sección 5.3.4.

En el caso de BPT-NEE, la convergencia presenta una inestabilidad inicial e incluso un empeoramiento transitorio de la métrica SSIM. Este fenómeno se debe a la aparición de luciérnagas (*fireflies*), generadas por contribuciones de energía extremadamente altas cuando algunos rayos logran intersecar las lámparas de forma fortuita. A medida que se acumulan más muestras, estas contribuciones se suavizan y la convergencia se estabiliza.

Resultados análogos se observan al utilizar la vista lateral desde la galería interior, aunque con una convergencia globalmente más lenta debido al mayor nivel de oclusión y a la complejidad geométrica del encuadre, que incrementan la cantidad de caminos que no logran aportar contribuciones relevantes.

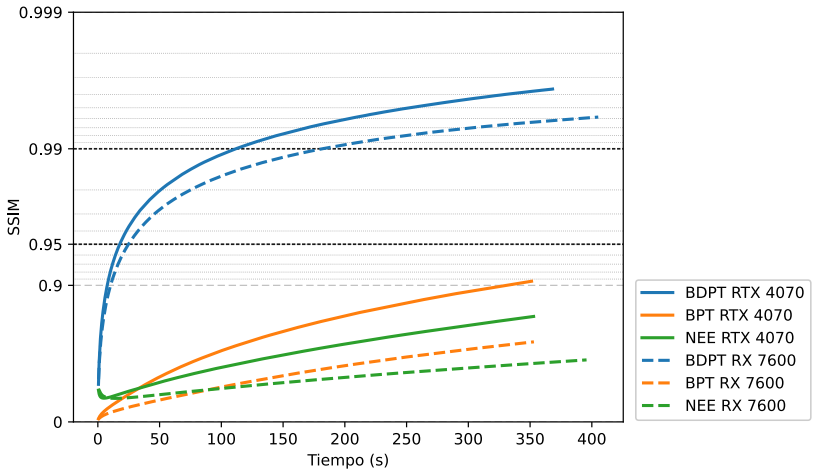


Figura 5.17: Comparación de convergencia (SSIM vs Tiempo) para la escena de *Sponza* iluminada mediante lámparas, ejecutada en ambas tarjetas gráficas

Comparación de Hardware

En esta escena de alta complejidad geométrica, la NVIDIA RTX 4070 mantiene una ventaja clara en las técnicas de trazado simple (BPT y BPT-NEE), superando a la AMD RX 7600. Sin embargo, al emplear algoritmos con mayor divergencia, como el BDPT, esta diferencia se reduce de forma considerable, en línea con lo observado en las escenas anteriores.

Dado que el comportamiento relativo del hardware reproduce las tendencias observadas en las escenas anteriores, no se profundiza nuevamente en este aspecto para esta configuración.

5.3.6. Primitivas Personalizadas vs. Mallas de Triángulos

Para verificar si los *shaders* de intersección personalizados afectan el rendimiento, se repitió la prueba de la Sección 5.3.2, sustituyendo las esferas con primitivas personalizadas por mallas de triángulos estándar. Esta modificación permite utilizar la aceleración nativa de los núcleos específicos para trazado de rayos de las tarjetas, sin ejecutar los *Intersection Shaders* programables (`.rint`).

Como se puede observar en la Figura 5.18, en la tarjeta AMD RX 7600, el rendimiento disminuyó levemente al cambiar a mallas, lo que sugiere que su arquitectura híbrida maneja mejor las primitivas personalizadas y no sufre el cambio de contexto que estas pueden provocar.

En el caso de la NVIDIA RTX 4070, como se observa en la Figura 5.19, en la técnica BDPT no hubo prácticamente cambios, la tasa de muestreo varió de 16,4 FPS (primitivas personalizadas) a 16,0 FPS (mallas), y el tiempo para alcanzar un SSIM de 0,95 incluso aumentó ligeramente de 4,97 s a 5,07 s. Este comportamiento se explica por la naturaleza altamente divergente de BDPT.

La gran cantidad de bifurcaciones lógicas (*branching*) se convierte en el cuello de botella dominante.

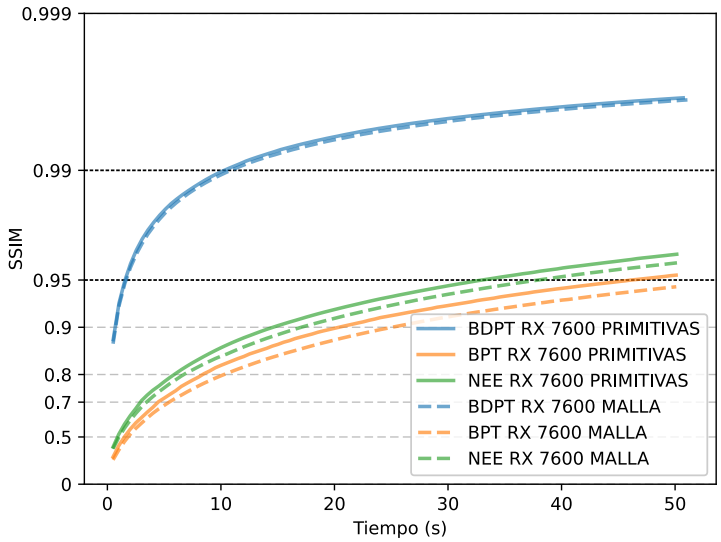


Figura 5.18: Comparación de convergencia (SSIM vs Tiempo) para la escena de *CornellBox* con primitivas personalizadas para las esferas contra mallas ejecutada con AMD RX 7600.

Por el contrario, en el algoritmo BPT el cambio de esferas con primitivas personalizadas a mallas sí se traduce en una mejora para la tarjeta de NVIDIA. La tasa de muestreo saltó de 663 FPS con esferas con primitivas personalizadas a 810 FPS con mallas (mejora del 22 %), reduciendo el tiempo de convergencia para un SSIM $> 0{,}95$ de 30,05 segundos a 25,03 segundos.

Los resultados de BPT con mallas muestran que la arquitectura de NVIDIA posee una capacidad de intersección de rayos con triángulos significativamente mayor que la RX 7600, alcanzando 810 FPS frente a 356 FPS. Sin embargo, esta ventaja se reduce en BDPT, donde la divergencia de hilos (*branching*) introducida por el algoritmo penaliza en mayor medida a esta arquitectura. En el caso de las esferas implementadas con primitivas personalizadas, la diferencia entre ambas placas se reduce, con un efecto levemente favorable para AMD y desfavorable para NVIDIA.

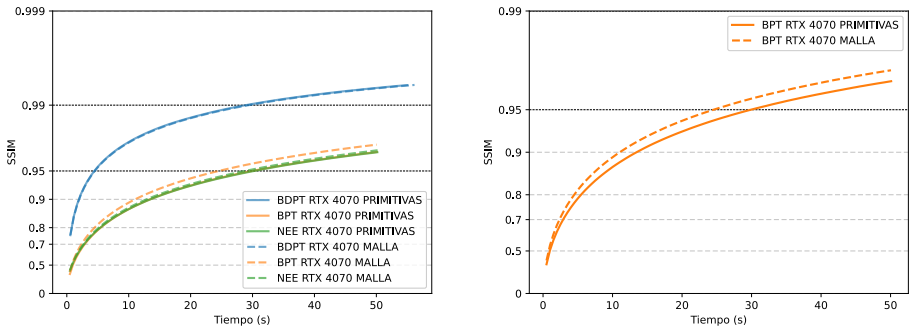


Figura 5.19: Comparación de convergencia SSIM para la escena de *CornellBox* con esferas de mallas contra primitivas personalizadas ejecutada en la tarjeta RTX 4070. Nótese que los resultados para BPT y BPT-NEE coinciden para la prueba con primitivas personalizadas.

5.3.7. Tasa de Muestreo (FPS)

En esta sección se comparan ambas tarjetas gráficas a partir de la tasa de muestreo obtenida para cada escena y técnica (Tabla 5.2). Este análisis complementa los resultados de convergencia presentados anteriormente y permite comparar el costo computacional de BPT, BPT-NEE y BDPT en distintas configuraciones geométricas y de iluminación.

	RTX 4070			**RX 7600**		
Escena	**BPT**	**NEE**	**BDPT**	**BPT**	**NEE**	**BDPT**
CornellBox mallas	810.5	314.6	16.0	355.9	237.5	43.7
CornellBox geom. pers.	663.2	299.8	16.4	429.3	274.8	46.9
Dormitorio	558.5	152.2	15.3	141.4	44.3	18.0
Veach	1065.2	359.6	22.6	596.2	161.1	33.6
Sponza	397.5	102.4	44.1	114.9	38.2	23.2
Sponza galeria	411.0	103.9	52.6	122.8	41.7	24.3
Sponza luces	290.8	47.6	20.9	101.7	15.9	15.3
Sponza galeria luces	300.5	50.1	25.2	127.9	17.5	17.3

Tabla 5.2: FPS por escena y técnica en ambas tarjetas gráficas.

En BPT y BPT-NEE, la RTX 4070 presenta una ventaja clara en todas las escenas. Esta diferencia es consistente con su mayor rendimiento en intersecciones rayo-triángulo, lo que impacta directamente en técnicas donde el costo dominante está asociado al recorrido de rayos.

Para comparar ambas tarjetas gráficas de forma más directa, la Tabla 5.3 presenta el rendimiento de la RX 7600 relativo al de la RTX 4070. Valores superiores al 100 % (resaltados en rojo) indican que la RX 7600 supera a la RTX 4070 en la técnica correspondiente.

Escena	BPT	NEE	BDPT
CornellBox mallas	44 %	76 %	273 %
CornellBox geom. pers.	65 %	92 %	286 %
Dormitorio	25 %	29 %	118 %
Veach	56 %	45 %	149 %
Sponza	29 %	37 %	53 %
Sponza galeria	30 %	40 %	46 %
Sponza luces	35 %	33 %	73 %
Sponza galeria luces	43 %	35 %	69 %

Tabla 5.3: Rendimiento relativo de la RX 7600 respecto a la RTX 4070 (FPSRX/FPSRTX). Se indican en rojo los valores superiores al 100 %.

En BDPT, el comportamiento cambia. Como se observa en la Tabla 5.2 y en la Tabla 5.3, la RX 7600 supera a la RTX 4070 en CornellBox, Dormitorio y Lámparas de Veach, mientras que en las distintas variantes de Sponza la relación se invierte.

En CornellBox, Dormitorio y Lámparas de Veach, BDPT presenta una tasa de muestreo menor en la RTX 4070 que en la RX 7600 (Tabla 5.2). Esto sugiere que el costo de construir y conectar subcaminos, junto con el cálculo de los pesos de MIS, penaliza más a la arquitectura de NVIDIA, lo que se refleja en el rendimiento relativo mostrado en la Tabla 5.3.

En Sponza, en cambio, la RTX 4070 mantiene ventaja también en BDPT. Este comportamiento se explica por el mayor peso del costo de intersección sobre geometría compleja, donde el hardware dedicado de NVIDIA resulta más eficiente. Esto se observa también al comparar CornellBox con mallas frente a CornellBox con primitivas personalizadas: al aumentar el uso de intersecciones rayo-triángulo, la RTX 4070 mejora su rendimiento relativo, mientras que la RX 7600 lo reduce.

Un caso particularmente ilustrativo es el de Sponza iluminada mediante lámparas pequeñas. En esta configuración, el rendimiento de BDPT cae significativamente respecto a la variante con skybox (Tabla 5.2). La causa principal es que disminuye la probabilidad de que los subcaminos generados desde la cámara intersequen directamente las fuentes de luz, lo que incrementa la cantidad de conexiones explícitas entre subcaminos. Esto aumenta el branching y el costo computacional del algoritmo.

Dado que, como se discutió anteriormente, la arquitectura de NVIDIA se ve más penalizada por divergencia de control, esta situación reduce de forma marcada la tasa de muestreo en la RTX 4070. En la RX 7600 este efecto también está presente, pero es menos pronunciado, lo que se refleja en una menor degradación relativa del rendimiento en BDPT (Tabla 5.3).

En BPT-NEE, el impacto de múltiples luces también es visible. En las escenas con lámparas, el costo de evaluar iluminación directa crece con la cantidad de fuentes, ya que cada punto requiere múltiples pruebas de visibilidad. En la RX 7600, esto lleva a que las tasas de muestreo de BPT-NEE y BDPT sean cercanas. En la RTX 4070, en cambio, la diferencia entre ambas técnicas se

mantiene, lo que sugiere que el recorrido de rayos continúa siendo relativamente más eficiente que la lógica adicional de BDPT.

Finalmente, estos resultados refuerzan una distinción importante: la tasa de muestreo y la velocidad de convergencia no son equivalentes. Una técnica puede generar menos muestras por segundo y, sin embargo, converger más rápido si cada muestra aporta información más útil. CornellBox constituye el ejemplo más claro de este comportamiento, ya que BDPT presenta una tasa de muestreo muy inferior a BPT y BPT-NEE, pero en las curvas de convergencia mostradas anteriormente resulta más eficiente. Por este motivo, los FPS deben interpretarse como una medida del costo computacional por muestra, y no como un indicador suficiente del desempeño global del algoritmo.

5.4. Comparación con Mitsuba

Con el objetivo de validar el comportamiento de los algoritmos implementados, se compararon los resultados obtenidos con los generados por Mitsuba 3 (Mitsuba Renderer Team, 2025), un motor de renderizado de referencia de código abierto ampliamente utilizado en investigación.

Para minimizar las diferencias no atribuibles a los algoritmos de transporte de luz, ambos motores fueron configurados para utilizar el modelo de materiales *Disney BSDF*. En el caso de Mitsuba, se empleó el integrador `path`, el cual implementa un trazado de caminos unidireccional que combina rayos de sombra y muestreo de importancia múltiple (MIS).

En las pruebas de esta sección se emplea una profundidad máxima de 10 rebotes, ya que el objetivo principal es la comparación de la imagen final generada por los algoritmos. Esta configuración permite capturar de manera más completa los fenómenos de iluminación global, evitando la pérdida de contribuciones asociadas a limitar el número de rebotes.

5.4.1. Resultados Cuantitativos

La Tabla 5.4 muestra los valores obtenidos para las métricas de comparación presentadas en la Sección 5.1.3, utilizando la imagen generada por Mitsuba como referencia.

En general, los resultados indican un alto grado de similitud entre las imágenes generadas por las distintas implementaciones y la referencia. Todas las técnicas presentan valores elevados de SSIM, lo que sugiere que la estructura global de las imágenes se conserva adecuadamente.

En términos de error absoluto, BPT-NEE obtiene el menor valor de MSE y RMSE, seguido de BPT. Esto indica que ambas técnicas reproducen con mayor precisión los valores de radiancia de la imagen de referencia. Por el contrario, BDPT presenta errores significativamente mayores en estas métricas, evidenciando discrepancias más pronunciadas en la intensidad de ciertas contribuciones.

Sin embargo, BDPT alcanza el mayor valor de SSIM, lo que indica que, a pesar de las diferencias en intensidad, la estructura espacial de la imagen se mantiene consistente con la referencia. Este comportamiento sugiere que las discrepancias observadas en BDPT están asociadas principalmente a errores de escala o atenuación, como los identificados en los fenómenos de reflexión y transmisión, más que a diferencias estructurales en la imagen.

Técnica	MSE	RMSE	SSIM	ꟻLIP
BPT	$4{,}9349 \times 10^{-5}$	$7{,}0249 \times 10^{-3}$	0,9903	$2{,}8473 \times 10^{-2}$
NEE	$3{,}8666 \times 10^{-5}$	$6{,}2182 \times 10^{-3}$	0,9921	$2{,}1467 \times 10^{-2}$
BDPT	$1{,}3924 \times 10^{-4}$	$1{,}1800 \times 10^{-2}$	0,9932	$3{,}6367 \times 10^{-2}$

Tabla 5.4: Métricas para cada algoritmo, utilizando la imagen generada por Mitsuba como referencia

5.4.2. Comparación Visual

Además del análisis cuantitativo, se realizó una comparación visual entre las imágenes producidas por Mitsuba y las generadas por nuestros algoritmos utilizando la métrica perceptual ꟻLIP, presentada en la Sección 5.1.3.

La Figura 5.20 muestra la imagen de referencia generada por Mitsuba, las salidas de cada algoritmo implementado y los correspondientes mapas de error ꟻLIP. En estos mapas, los colores fríos indican diferencias perceptualmente poco significativas, mientras que los colores cálidos señalan discrepancias más perceptibles.

5.4.3. Discusión de Resultados

BPT (Trazado de Caminos Inverso)

La imagen producida mediante BPT presenta una alta similitud visual con la referencia de Mitsuba. El mapa ꟻLIP muestra predominantemente colores fríos, lo que indica que las diferencias perceptuales son mínimas en la mayor parte de la imagen. La transmisión y la refracción de la luz a través del objeto transparente se mantienen coherentes con la referencia. No obstante, se aprecian ligeras discrepancias en la intensidad de las cáusticas y los reflejos.

En conjunto, estas observaciones sugieren que la implementación del modelo de materiales reproduce adecuadamente el comportamiento de referencia para los fenómenos presentes en la escena.

BPT-NEE (Estimador del Siguiente Evento)

Entre las técnicas evaluadas, BPT-NEE presenta el mayor grado de similitud con la referencia de Mitsuba. Los resultados son, en términos generales, muy cercanos a los obtenidos con BPT estándar, aunque se observan discrepancias ligeramente menores en superficies como el suelo, las paredes y la mesa.

Figura 5.20: Comparación de los tres algoritmos utilizando Mitsuba como referencia. La primera columna corresponde a la referencia, la segunda a nuestra implementación y la tercera al mapa de error ꟻLIP. Un color frío (por ejemplo, azul o violeta) indica diferencias perceptualmente poco significativas, mientras que un color cálido (por ejemplo, amarillo o naranja) indica una diferencia claramente perceptible.

La transmisión de la luz a través del objeto transparente presenta diferencias similares a las observadas en BPT estándar.

Asimismo, el borde del área iluminada por la lámpara de pie difiere levemente respecto a la referencia. Esta variación podría estar relacionada con diferencias en el tratamiento de los rayos de sombra o en la estrategia de muestreo empleada por Mitsuba.

BDPT (Trazado de Caminos Bidireccional)

El algoritmo BDPT presenta las discrepancias más pronunciadas respecto a la referencia, especialmente en los fenómenos de transmisión y refracción de la luz a través del objeto transparente, los cuales aparecen considerablemente más oscuros en comparación con las otras técnicas.

La Figura 5.21 muestra la salida del algoritmo BDPT al incrementar signifi-

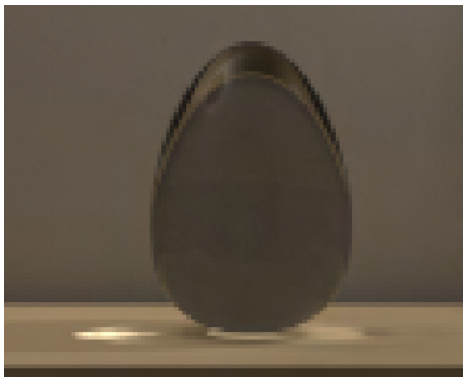


Figura 5.21: Salida del algoritmo BDPT para la escena *Lámparas de Veach*. Se incrementó significativamente la exposición para visualizar el fenómeno de refracción.

cativamente la exposición. En esta imagen se observa que tanto la transmisión a través del objeto transparente como la reflexión en su superficie están presentes en la solución final, aunque con una intensidad significativamente reducida. Por otro lado, el reflejo generado por la lámpara derecha sobre el objeto transparente no aparece en la imagen final, incluso al modificar la exposición.

Este comportamiento parece estar relacionado con la interacción entre el modelo de materiales y el cálculo de los pesos de muestreo de importancia múltiple (MIS). Al analizar la pirámide de Veach antes de aplicar los pesos MIS (ver Figura 4.13), se observa que las técnicas correspondientes a las conexiones $(s,t) = (0,3)$, $(0,4)$ y $(1,3)$ capturan correctamente el reflejo de la lámpara derecha. Sin embargo, tras la aplicación de los pesos MIS, dichas contribuciones se atenúan significativamente, resultando prácticamente imperceptibles en la imagen final.

El área iluminada por la lámpara de pie presenta un comportamiento similar al observado en BPT-NEE. Asimismo, el efecto de la lámpara derecha sobre las paredes es considerablemente menor en BDPT que en el resto de las técnicas.

Capítulo 6

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones del proyecto, evaluando los resultados obtenidos en relación con los objetivos planteados inicialmente. Asimismo, se analizan las principales dificultades encontradas durante el desarrollo y se proponen posibles líneas de trabajo futuro para extender y mejorar la solución implementada.

6.1. Conclusiones

Este proyecto se centró en estudiar, implementar y analizar técnicas de trazado de caminos (inverso, con estimación del siguiente evento y bidireccional), evaluando tanto su comportamiento algorítmico como su desempeño práctico, así como en analizar la implementación de trazado de rayos en Vulkan sobre GPUs de distintos fabricantes.

Los resultados obtenidos permitieron confirmar que la eficiencia de cada técnica depende fuertemente de la iluminación y la estructura de la escena. En escenas con fuentes de luz pequeñas y ocluidas, el trazado de caminos bidireccional mostró una mejora significativa en la convergencia, al facilitar la generación de caminos que conectan de forma explícita la cámara con las fuentes luminosas. En contraste, en escenas dominadas por luces grandes y de fácil acceso, los métodos basados únicamente en el trazado desde la cámara resultaron más eficientes debido a su menor coste de cálculo.

Asimismo, se observó que las variaciones en la configuración de iluminación pueden generar diferencias importantes en el comportamiento de un mismo algoritmo, como ocurre en *Sponza* con *Skybox* y con luces pequeñas (Sección 5.3.5). Esto evidencia que no existe una técnica que resulte óptima en todos los casos y que la elección del método adecuado debe realizarse en función del tipo de escena y de los fenómenos de iluminación que se desean capturar.

Desde el punto de vista de la ejecución en GPU, se identificaron diferencias relevantes entre arquitecturas gráficas. En particular, los resultados sugieren que algoritmos con alta divergencia de control y lógica compleja, como BDPT, presentan un impacto más marcado en el rendimiento en arquitecturas como la de NVIDIA, mientras que la arquitectura de AMD muestra un comportamiento más estable frente a este tipo de algoritmos.

En conjunto, el trabajo realizado permitió cumplir los objetivos planteados inicialmente, aportando una implementación funcional y un análisis experimental que contribuye a una mejor comprensión del comportamiento de las técnicas de trazado de caminos en distintos escenarios de iluminación.

6.2. Dificultades y Limitaciones

Durante el desarrollo del proyecto se enfrentaron diversas dificultades técnicas y conceptuales.

Una de las principales dificultades estuvo vinculada a la adaptación del algoritmo BDPT a GPU, ya que este fue originalmente concebido para un contexto secuencial. Esto implicó contemplar múltiples casos particulares asociados a la construcción y conexión de subcaminos, tal como se describe en el capítulo de implementación (Sección 4.11).

Otra dificultad relevante fue la evaluación de los resultados obtenidos. Tanto el modelo de materiales como los algoritmos de transporte de luz producen salidas cuya corrección no siempre puede determinarse únicamente mediante inspección visual. En este contexto, resulta complejo distinguir entre un comportamiento físicamente incorrecto, una alta varianza inherente al método o un posible problema de implementación, aun cuando se realizaron comparaciones sistemáticas con el objetivo de obtener los resultados más consistentes posibles.

Asimismo, el proceso de depuración se vio condicionado por las limitaciones propias del desarrollo en GPU. La ausencia de herramientas de depuración equivalentes a las disponibles en entornos CPU, junto con la ejecución altamente paralela, dificultó la identificación de errores sutiles, en particular aquellos relacionados con probabilidades, acumulación de energía y divergencia de control.

6.3. Trabajo Futuro

A partir de los resultados obtenidos, se identifican múltiples líneas de trabajo futuro que permitirán extender y mejorar el proyecto:

Una de las extensiones más relevantes consiste en incorporar *Multiple Importance Sampling* (MIS) en la implementación de BPT-NEE. Actualmente, la ausencia de esta técnica contribuye a la aparición de luciérnagas en determinadas escenas, especialmente en configuraciones con fuentes pequeñas o altamente energéticas. La inclusión de MIS permitiría combinar de forma más robusta las distintas estrategias de muestreo, reduciendo la varianza y acercando el comportamiento del algoritmo al observado en renderizadores de referencia como

Mitsuba.

Asimismo, se identificaron comportamientos a revisar en la implementación de BDPT, particularmente en la evaluación de reflejos sobre materiales transparentes. Un análisis más profundo de estos casos permitiría mejorar la fidelidad física del modelo y garantizar una correcta propagación de la energía en caminos que involucran refracción y reflexión especular.

Otra posible extensión sería incorporar una etapa de *denoising* al pipeline de renderizado. La inclusión de filtros espaciales o técnicas basadas en aprendizaje automático permitiría evaluar la calidad visual final con un número limitado de muestras, obteniendo resultados visualmente más estables en las primeras iteraciones.

También se plantea como trabajo futuro la extensión del sistema de iluminación para soportar luces direccionales y puntuales. Esto permitiría analizar escenarios adicionales y fenómenos de iluminación no contemplados en las configuraciones actuales.

Por otro lado, resulta relevante profundizar en el análisis de materiales complejos, en particular en el manejo de anisotropía en texturas. Si bien se implementó soporte para este tipo de materiales, su validación exhaustiva requiere un estudio más detallado.

Finalmente, se propone como trabajo futuro extender las evaluaciones realizadas a un conjunto más amplio de hardware. Para ello, se plantea el desarrollo de un *benchmark* que considere un conjunto fijo de escenas, condiciones de iluminación y métricas de evaluación, permitiendo realizar comparaciones sistemáticas y reproducibles entre diferentes configuraciones de hardware.

Referencias

AirGuanZ. (2025). *Imgui filebrowser.* https://github.com/AirGuanZ/imgui-filebrowser/tree/master. (Accedido: 2025-11-22)

Boyd, R. W. (1983). *Radiometry and the detection of optical radiation.* New York, NY, USA: Wiley-Interscience.

Burley, B. (2012). *Physically-based shading at disney* (Inf. Téc.). Walt Disney Animation Studios. Descargado de https://disneyanimation.com/publications/physically-based-shading-at-disney/

Burley, B. (2015). *Extending the disney brdf to a bsdf with integrated subsurface scattering* (Inf. Téc.). Walt Disney Animation Studios. Descargado de https://blog.selfshadow.com/publications/s2015-shading-course/burley/s2015_pbs_disney_bsdf_notes.pdf

Cornut, O. (2025). *Imgui.* https://github.com/ocornut/imgui. (Accedido: 2025-10-13)

Fontana, J. (2024). *Uso de vulkan para el trazado de rayos.* https://github.com/jquinfontana/VulkanRayTracing. (Accedido: 2025-10-13)

Fujita, S. (2025). *Tinyexr.* https://github.com/syoyo/tinyexr. (Accedido: 2025-10-13)

Hecht, E. (2002). *Optics, fifth edition.* San Francisco: Addison Wesley.

Jarzynski, M., y Olano, M. (2020, 17 de October). Hash functions for gpu rendering. *Journal of Computer Graphics Techniques (JCGT)*, *9*(3), 20–38. Descargado de http://jcgt.org/published/0009/03/02/

Kajiya, J. T. (1986). The rendering equation. *SIGGRAPH Comput. Graph.*, *20*(4), 143–150.

Khronos Group. (2023). *Vulkan ray tracing best practices for hybrid rendering.* Descargado de https://www.khronos.org/blog/vulkan-ray-tracing-best-practices-for-hybrid-rendering

Kramer, L. (2023). *Rdna™3: Beyond the current gen.* https://gpuopen.com/download/RDNA3_Beyond-the-current-gen-v4.pdf. (AMD GPUOpen presentation. Accedido: 2025-12-09)

Lafortune, E. P., y Willems, Y. D. (1993). Bi-directional path tracing. En *Proceedings of the third international conference on computational graphics and visualization techniques (compugraphics '93)* (pp. 145–153). Alvor, Portugal.

Marrs, A., Shirley, P., y Wald, I. (Eds.). (2021). *Ray tracing gems ii.* Apress. (http://raytracinggems.com/rtg2)

Melenero, J. F. (2009). *Reflexión interna total.* https://es.wikipedia.org/wiki/Ley_de_Snell#/media/Archivo:ReflexionTotal.svg. (CC BY-SA 4.0)

Mitsuba Renderer Team. (2025). *Mitsuba renderer.* Descargado 2026-01-06, de https://mitsuba-renderer.org/

NVIDIA. (2023). *Nvidia ada gpu architecture.* https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf. (Technical Whitepaper. Accedido: 2025-12-09)

O'Neill, M. E. (2014, septiembre). *Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation* (Inf. Téc. n.º HMC-CS-2014-0905). Claremont, CA: Harvey Mudd College.

Pharr, M., Jakob, W., y Humphreys, G. (2023). *Physically based rendering: From theory to implementation* (4th ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers.

Schlick, C. (1994). An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum*, *13*(3), 233–246.

Szirmay-Kalos, L. (1999). *Monte-carlo methods in global illumination.*

Veach, E. (1997). *Robust monte carlo methods for light transport simulation* (Tesis Doctoral, Stanford University, Stanford, California). Descargado de https://graphics.stanford.edu/papers/veach_thesis/

Walter, B., Marschner, S. R., Li, H., y Torrance, K. E. (2007). *Microfacet models for refraction through rough surfaces.* Goslar, DEU: Eurographics Association.

Anexo A

Formato de escenas y modelo de materiales

A.1. Formato general del archivo de escenas (.scn)

Los archivos de escena utilizan formato JSON y contienen los siguientes campos principales:

- `resolution`: resolución de salida `[width, height]`.
- `maxdepth`: profundidad máxima de rebotes.
- `camera`: parámetros de cámara.
- `materials`: lista de materiales disponibles.
- `entities`: lista de objetos que conforman la escena.

Cámara

- `position` (`[x,y,z]`): posición de la cámara.
- `lookat` (`[x,y,z]`): punto hacia el que mira.
- `fov` (`float`): campo de visión en grados.
- `antialiasing_radius` (`float`, opcional): radio del circulo de distorsión del antialiasing.

A.2. Materiales

Cada material tiene un conjunto de parámetros escalares y/o texturas.

- `name` (`string`): identificador del material.

- `color` (`[r,g,b]`): color base.
- `emission` (`[r,g,b]`, opcional): radiancia emitida.
- `metallic` (`float`, opcional): mezcla metal/dieléctrico.
- `roughness` (`float`, opcional): rugosidad del lóbulo especular.
- `opacity` (`float`, opcional): controla la mezcla con transmisión.
- `ior` (`float`, opcional): índice de refracción.
- `subsurface` (`float`, opcional): aproximación de dispersión subsuperficial.
- `speculartint` (`float`, opcional): tinte del componente especular.
- `anisotropic` (`float`, opcional): intensidad de anisotropía.
- `sheen` (`float`, opcional): componente sheen.
- `sheentint` (`float`, opcional): tinte del sheen.
- `tiling` (`float`, opcional): factor multiplicativo aplicado a la posición UV en el espacio de la textura (para permitir la repetición de la misma).

Texturas soportadas

- `albedotexture`: textura RGB que modula el color base.
- `metallictexture`: textura escalar para metallic.
- `roughnesstexture`: textura escalar para roughness.
- `opacitytexture`: textura escalar para opacity.
- `anisotropictexture`: textura RGB que define la dirección de anisotropía.
- `masktexture`: textura escalar para recorte.

A.2.1. Entidades

Cada elemento de `entities` representa un objeto renderizable.
Todos los objetos pueden definir:

- `position` (`[x,y,z]`)
- `rotation` (`[x,y,z]`)
- `scale` (`[x,y,z]`)

A su vez, deben definir un tipo de objeto fijando un valor al parámetro *type*, a continuación se definen los tipos de objetos y sus parámetros.

Tipo mesh

- `type`: `"mesh"`
- `file`: ruta a archivo `.obj`

Asignación de materiales:

- Si el `.obj` define materiales mediante `usemtl`, se busca un material con el mismo `name`.
- Si no se encuentra, se utiliza `default_material`.
- Si se define el campo `material`, este reemplaza todos los materiales del mesh.

Esfera con Primitivas Personalizadas

- `type`: `"sphere"`
- `radius` (`float`)
- `material` (`string`)
- `anisotropic_direction` (`[x,y,z]`, opcional)
- `inverted_normal` (`int`, opcional)

A.3. Modelo de Materiales Implementado

El modelo implementado es una adaptación del BSDF de Disney, basado en microfacetas GGX con soporte para:

- componente difusa no lambertiana,
- componente especular microfacet,
- transmisión especular,
- anisotropía,
- sheen,
- rugosidad,
- metálico.

BaseColor

El color final por píxel se calcula como:

$$baseColor = color \times albedoTexture(uv)$$

si existe textura de albedo.

Metállico y Rugosidad

- `metallic` controla la transición entre material metálico y dieléctrico.
- `roughness` controla el ancho del lóbulo especular.
- Si existen texturas correspondientes, estas reemplazan el valor escalar.

Opacidad y Transmisión

- `opacity` controla la mezcla con el lóbulo de transmisión.
- Puede ser escalar o provenir de `opacitytexture`.

Masking

- `masktexture` implementa un masking.
- Si el valor muestreado es menor a 0.5, la intersección se descarta y el rayo continúa.

Anisotropía

- `anisotropic` define la intensidad del efecto.
- La orientación proviene de `anisotropictexture` o de `anisotropic_direction`.

Emisión

- `emission` define radiancia emitida.
- Materiales con emisión distinta de cero actúan como fuentes de luz.