UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Renderizado de medios participativos

Informe de Proyecto de Grado presentado por

Emiliano Luna

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

Dr. Ing. Eduardo Fernández

Montevideo, 9 de febrero de 2024

Para papá

Resumen

El renderizado de medios participativos es un conjunto de técnicas avanzadas de computación gráfica, diseñadas para crear representacions visuales de espacios tridimensionales. Incluyen elementos volumétricos que dispersan la luz, como nubes o niebla, agua turbia, plástico y vidrio, además de las superficies comúnmente utilizadas en el modelado tradicional.

Técnicamente, los fenómenos simulados por estas técnicas se deben a la interacción de la luz con un gran número de partículas microscópicas, que se modelan considerando sus propiedades estadísticas. Este enfoque evita el uso de memoria y reduce los tiempos de cómputo que conllevan la creación y operación con micro partículas de forma individual.

Estas técnicas se aplican en la generación de imágenes médicas, simulaciones científicas, videojuegos, películas, entre otros.

En este proyecto se describen e implementan técnicas para medios homogéneos, o sea aquellos en los que la distribución de las partículas es uniforme, así como para medios heterogéneos cuando no es el caso. Entre ellas se encuentran el Ray Marching, Delta Tracking y Ratio Tracking.

Como producción propia se desarrolló un *renderer* en C++, usando la librería NanoVDB para implementar y comparar diferentes técnicas de renderizado, ya sea en la calidad de las imágenes como en sus tiempos de ejecución. Durante la implementación se tomó en cuenta la eficiencia computacional, ya que estas técnicas suelen ser particularmente costosas.

El código resultante se encuentra disponible en Github [1]

Palabras clave: Rendering, Ray Marching, Delta Tracking, Ratio Tracking, Medio participativo

[1] https://github.com/emiliano-luna/volume-renderer

Glosario

Albedo: Proporción de radiancia que un medio devuelve luego de interactuar con él. Este valor no tiene unidad y se define entre 0 y 1, siendo 0 un medio que absorbe toda la radiancia incidente y 1 un medio que no absorbe radiancia incidente.

Función de densidad de probabilidad - Probability Distribution Function (PDF): Es una función que describe la probabilidad relativa con la que una variable aleatoria continua tomará determinado valor. La probabilidad de que la variable aleatoria continua caiga en una región específica del espacio de posibilidades estará dada por la integral de la densidad de esta variable entre uno y otro límite de dicha región.

Función de distribución acumulada - Cumulative Distribution Function (CDF): Función que devuelve la probabilidad que una variable aleatoria tome un valor menor o igual a un número dado.

Muestreo de importancia (Importance Sampling): Técnica usada en integración de Montecarlo en la cual se toma muestras de una distribución simple en lugar de usar la distribución real que es difícil de muestrear, y luego se las pondera de forma que se aproximen a la distribución real.

Integrador: Un algoritmo que ejecuta los cálculos necesarios para convertir información volumétrica en una imagen. En este proyecto cada integrador desarrollado está asociado a las distintas técnicas que se comentan en el informe.

Inversa de la función de distribución acumulada ó función cuantil ($\mathbf{CDF}^{-1}$): Dada una probabilidad para la que una variable aleatoria tome un valor menor o igual, devuelve el número en el dominio que lo cumple.

Muestreo de importancia múltiple (MIS o múltiple importance sampling): Técnica usada en integración de Montecarlo en la cual varias estrategias de muestreo se combinan para obtener una única muestra. No se implementó en este proyecto pero es un término común en el renderizado de medios participativos.

Muestreo por rechazo (Rejection Sampling): El muestreo por rechazo es una técnica de simulación usada en análisis numérico y estadística computacional para generar muestras de una distribución dada. Funciona para distribuciones de cualquier dimensión con una función de densidad conocida. La esencia del método es simular puntos de manera uniforme sobre el gráfico bidimensional de la función de densidad y solo conservar aquellos puntos que caen debajo de la curva de la función, que representa las probabilidades de los

diferentes resultados.

Se generan pares de números aleatorios como coordenadas en un plano que abarca la función de densidad y se selecciona solo aquellos que están por debajo de la curva de la función. Este proceso se repite hasta obtener la cantidad deseada de muestras válidas. Es un método versátil que puede adaptarse a distribuciones complejas donde otros métodos de muestreo fallan o son ineficientes.

Radiancia: Flujo radiativo que se desplaza a lo largo de un rayo diferencial y describe la distribución de la luz en un espacio. No varia si se mide para un rayo en el vacío. Se mide en watt por estereorradián por metro cuadrado ($Wsr^{-1}m^{-2}$) y va a depender de la dirección desde la que se observe.

Ruleta Rusa: Técnica de optimización para estimadores de Monte Carlo que busca evitar la evaluación de muestras costosas que tienen una contribución escasa en el resultado final.

Funciona cancelando probabilísticamente algunas muestras (rayos en este proyecto) cuando se encuentran por debajo de una constante dada. En caso de que un rayo no se cancele su contribución aumenta para compensar por los rayos que fueron cancelados. Por más que mejora la eficiencia del algoritmo esta técnica nunca reduce la varianza, en general la incrementa.

Tabla de símbolos

Símbolo	Definición
μ_a	Coeficiente de absorción.
μ_s	Coeficiente de dispersión.
μ_t	Coeficiente de extinción, $\mu_t = \mu_s + \mu_a$.
$\bar{\mu_n}$	Coeficiente de null-collision.
$\tau(\mathbf{x}, \mathbf{y})$	Espesor óptico entre $\mathbf{x}$ a $\mathbf{y}$.
f_s	Función de fase.
g_{HG}	Parámetro de asimetría de la función de fase de Henyey-Greenstein.
L	Radiancia.
L_e	Radiancia debido a la emisión volumétrica.
L_s	Radiancia debido a la dispersión hacia adentro.
L_o	Radiancia saliente.
G	Término de geometría.
$T(\mathbf{x}, \mathbf{y})$	Transmitancia entre $\mathbf{x}$ a $\mathbf{y}$.
$\bar{\mathbf{x}}$	Secuencia de puntos $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k$ de un camino que va desde la cámara a la escena.

Índice general

Capítulo 1

Introducción

En este proyecto se busca renderizar escenas con medios participativos de creciente complejidad, que incluyen medios homogéneos y heterogéneos. Para ello se desarrolla un *renderer* que recibe como entrada un archivo de configuración así como un modelo ó una grilla de NanoVDB, y devuelve como salida una imagen resultante.

El tiempo de ejecución es un elemento importante a considerar, ya que la generación de una imagen puede llevar decenas de minutos. Para ello se implementan técnicas de optimización como paralelismo de cómputo y terminación temprana con ruleta rusa.

Luego se comparan los resultados con el *renderer* PBRT (*pbrt4*, 2023). Si bien es cierto que dicho *renderer* tiene como principal objetivo ser didáctico, utiliza técnicas que escapan al alcance de este informe, como el renderizado espectral, muestreo de importancia múltiple (MIS, por sus siglas en inglés) y escenas con geometría compleja, además de medios participativos.

El renderizado de medios participativos es un área de investigación activa desafiante, cuyos problemas abarcan diferentes materias de la carrera de Ingeniería en Computación como la física, cálculo, álgebra, además de las materias de programación y la Computación Gráfica.

En los videojuegos, los medios participativos se representaron originalmente de forma rudimentaria, no sólo con fines artísticos sino para ocultar limitantes de hardware al acortar la distancia de renderizado, como es el caso del juego Silent Hill (Prakash y cols., 2009). A medida que crecieron los recursos computacionales disponibles se soportaron nubes volumétricas y ray marching para representar iluminación y sombreado realista. Un ejemplo de gráficos de punta con renderizado de medios participativos es el juego Red Dead Redemption 2 (Tatarchuk, 2019).

En paralelo se desarrolló el área para fines científicos, ya sea para visualización en general (Wang, Paljic, y Fuchs, 2010) como para imagenología médica, por ejemplo en el uso de tomografías y resonancias magnéticas (Hutchison y

cols., 2007). Otras aplicaciones pueden ser la ingeniería, el diseño, el análisis de datos e imagenología industrial. Un área que en particular produjo grandes avances son las películas y animación en general. Referencias fundamentales para este informe vienen de profesionales de Disney, Pixar, Dreamworks y Sony: (Fong, Wrenninge, Kulla, y Habel, 2017), (Wrenninge y Bin Zafar, 2011), (Wrenninge, 2012), (Novák, Selle, y Jarosz, 2014), (Novák, Georgiev, Hanika, y Jarosz, 2018).

El informe comienza con la revisión de los antecedentes, en particular mencionando los conceptos básicos del Renderizado de Medios Participativos (Sec. 2.1). Las principales referencia de esta sección, incluyendo las imágenes usadas, son (Novák, Georgiev, Hanika, y Jarosz, 2018) y (Kulla y Fajardo, 2012). También se obtuvieron imágenes de (Novák, Georgiev, Hanika, Křivánek, y Jarosz, 2018). Luego se comentan algunas técnicas para el muestreo de distancia de rayos (Sec. 2.3) y análogamente para la transmitancia (Sec. 2.4) y finalmente se las combina en la sección de construcción de caminos (Sec. 2.5).

A continuación se comenta la arquitectura (Sec. 3.1), así como las decisiones de diseño (Sec. 3.2) que se tomaron en general, y se explica la implementación de los primeros algoritmos que se desarrollaron.

Para la implementación del Delta Tracking y Ratio Tracking se tomaron como referencia los capítulos 11 y 14 de Physically Based Rendering (Matt Pharr y Humphreys, 2023). Dicho libro fue una referencia clave a lo largo de todo el proyecto. Los archivos con escenas para NanoVDB se obtienen en (*pbrt4-scenes*, 2023). El código de NanoVDB se encuentra en (*NanoVDB*, 2023), además tiene utilidades que permiten por ejemplo convertir grillas de OpenVDB a NanoVDB en caso de ser necesario.

En el capítulo 4 se muestran imágenes de ejemplo junto con sus parámetros y tiempos de ejecución, y se las compara con resultados análogos en PBRT. Al final se desarrollan las principales conclusiones y el trabajo a futuro resultante del proyecto (Cap. 5).

Capítulo 2

Revisión de antecedentes

A continuación se dan fundamentos teóricos que sirven de base para las técnicas utilizadas en el renderizado de medios participativos, a la vez que se describen algunas de ellas.

Las principales referencias de esta sección son (Novák, Georgiev, Hanika, y Jarosz, 2018), (Novák, Georgiev, Hanika, Křivánek, y Jarosz, 2018) y (Kulla y Fajardo, 2012).

2.1. Transferencia radiativa

Se comentan a continuación algunas interacciones básicas entre luz y materia (Figura 2.1), que culmina en la Ecuación de Renderizado de Volumen.

Dichos procesos impactan en la radiancia, o sea en el flujo radiativo que se desplaza a lo largo de un rayo diferencial. Esta puede aumentar (emisión y dispersión hacia adentro) o disminuir (absorción y dispersión hacia afuera) según las propiedades ópticas del medio que atraviesan.

Figura 2.1: Interacciones de la luz en un medio participativo. La absorción y la dispersión hacia afuera reducen la radiancia, mientras que la emisión y la dispersión hacia adentro la incrementan.

2.1.1. Absorción

Si se observa la radiancia que pasa por un segmento diferencial a lo largo de una dirección dada, dicha cantidad se verá reducida y el tamaño de la reducción depende del coeficiente de absorción μ_a.

2.1.2. Dispersión hacia afuera (out-scattering)

Si existen partículas que dispersan la luz, una cantidad de esta irá en direcciones distintas a la que tenia originalmente. El coeficiente de dispersión μ_s cuantifica la densidad de probabilidad de un evento de dispersión por unidad de distancia. Para su caracterización se utiliza la función de fase, que se verá en la Sec. 2.2.2.

2.1.3. Dispersión hacia adentro (in-scattering)

De forma análoga a la dispersión hacia afuera, existen radiancias ajenas a la dirección del rayo que son dispersadas en esa dirección. Esta luz también depende del coeficiente de dispersión μ_s.

2.1.4. Emisión

Finalmente, la radiancia puede aumentar debido a la emisión dada por un volumen, por ejemplo en el caso de procesos incandescentes, como en una llama, o luminiscentes, como en las sustancias fosforescentes. En este caso se adopta la convención de multiplicar la radiancia emitida por μ_a. Esto se motiva por un argumento físico según el cual, para que exista emisión debe haber un material que tenga la habilidad de absorber energía y liberarla en forma de fotones.

2.2. Ecuación de Transferencia Radiativa (RTE)

Se pueden combinar dichos procesos en una sola ecuación diferencial llamada Ecuación de Transferencia Radiativa (RTE, por sus siglas en inglés) (Chandrasekhar, 1960)

$$\frac{dL(\mathbf{x},\omega)}{dz} = -\mu_a(\mathbf{x})L(\mathbf{x},\omega) - \mu_s(\mathbf{x})L(\mathbf{x},\omega) + \mu_a(\mathbf{x})L_e(\mathbf{x},\omega) + \mu_s(\mathbf{x})L_s(\mathbf{x},\omega) \tag{2.1}$$

Los componentes de la RTE (2.1) en el lado derecho corresponden respectivamente a la absorción ($-\mu_a(\mathbf{x})L(\mathbf{x},\omega)$), dispersión hacia afuera ($-\mu_s(\mathbf{x})L(\mathbf{x},\omega)$), emisión ($\mu_a(\mathbf{x})L_e(\mathbf{x},\omega)$) y dispersión hacia adentro ($\mu_s(\mathbf{x})L_s(\mathbf{x},\omega)$). Estos equivalen a la radiancia L transmitida por un segmento diferencial de un rayo de largo dz, que comienza en $\mathbf{x}$ y tiene dirección ω. $L_s(\mathbf{x},\omega)$ es la radiancia generada por dispersión hacia adentro y $L_e(\mathbf{x},\omega)$ la generada por emisión.

2.2.1. Forma integral de la RTE

Como la absorción y la dispersión hacia afuera afectan la misma función de radiancia L, podemos juntarlos en un mismo término, para ello definimos $\mu_t(\mathbf{x}) = \mu_a(\mathbf{x}) + \mu_s(\mathbf{x})$ como el coeficiente de extinción, que describe la densidad de probabilidad de cualquier colisión por unidad de distancia.

La RTE es una ecuación diferencial sobre un segmento diferencial de un rayo. Sin embargo, en computación gráfica trabajamos con rayos de largo finito. Por esta razón se integra ambos lados de la ecuación espacialmente, a lo largo de la dirección ω, para obtener la forma integral de la RTE:

$$L(\mathbf{x}, \omega) = \int_0^z T(\mathbf{x}, \mathbf{y})[\mu_a(\mathbf{y}) L_e(\mathbf{y}, w) + \mu_s(\mathbf{y}) L_s(\mathbf{y}, \omega)]dy \tag{2.2}$$

La forma integral nos permite calcular la radiancia que cruza por un punto $\mathbf{x}$ en dirección ω, o sea a lo largo del rayo $\mathbf{x}+y\omega = \mathbf{y}$. Es necesario entonces integrar desde su origen $\mathbf{x}$ hasta su final $\mathbf{x} + z\omega$, evaluar las ganancias de radiancia y escalarlas según la función de transmitancia $T(\mathbf{x}, \mathbf{y})$ que toma en cuenta las pérdidas a lo largo del camino hasta $\mathbf{y}$:

$$T(\mathbf{x}, \mathbf{y}) = e^{-\int_0^y \mu_t(s)ds} \tag{2.3}$$

La función de trasmitancia (2.3) cuantifica la cantidad de luz que llega desde un punto $\mathbf{x}$ a otro $\mathbf{y}$. El espesor óptico se refiere a una medida de cuánto afecta un medio a la luz que pasa a través de él. Un grosor óptico alto significa que el medio es muy denso o efectivo en la dispersión o absorción de la luz, lo que puede resultar en una visibilidad reducida a través de él. En cambio, un grosor óptico bajo indica que el medio tiene poco efecto en la luz, permitiendo una mayor transparencia (2.4).

$$\tau(\mathbf{x}, \mathbf{y}) = \int_0^y \mu_t(s)ds \tag{2.4}$$

A continuación se analiza en más detalle los términos que suman radiancia en (2.2). El término $\mu_a(\mathbf{y}) L_e(\mathbf{y}, w)$ representa la radiancia debida a la emisión volumétrica. El término restante $\mu_s(\mathbf{y}) L_s(\mathbf{y}, \omega)$ representa la radiancia debida a la dispersión hacia adentro.

$$L_s(\mathbf{y}, \omega) = \int_{S^2} f_p(\omega, \overline{\omega}) L(\mathbf{y}, \overline{\omega}) d\overline{\omega} \tag{2.5}$$

La función de radiancia dispersada hacia adentro (2.5) se computa integrando la luz incidente sobre la esfera S^2 de todas las direcciones $\bar{w}$, moduladas por la función de fase $f_p(\omega, \overline{\omega})$. La función de fase trata la distribución direccional de la luz dispersada y es el análogo volumétrico al BRDF.

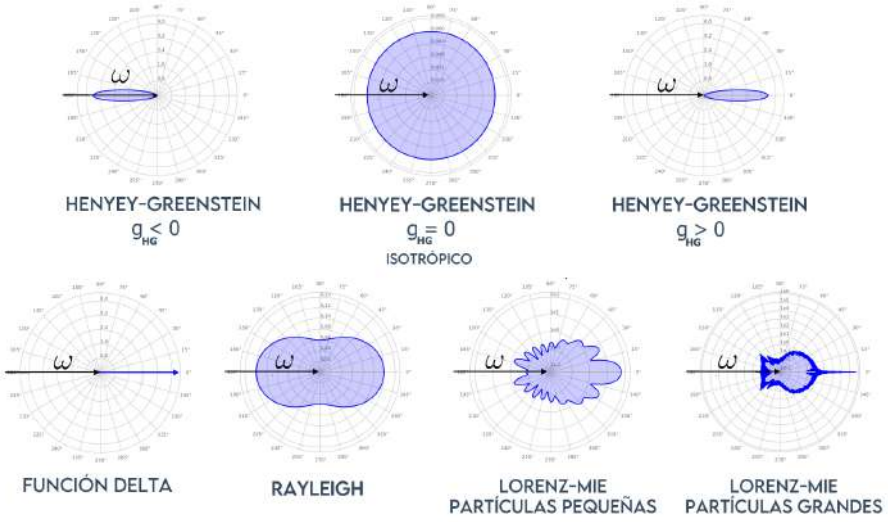

Figura 2.2: Algunos ejemplos de funciones de fase. En la primera fila de arriba se observan las diferentes formas que toma Henyey-Greenstein según el valor de su parámetro g_{HG}. En la segunda fila se muestran algunas otras funciones de fase comunmente utilizadas.

2.2.2. Función de fase

La función de fase describe la distribución direccional de la luz al ser dispersada por partículas.

En la Figura 2.2 se observan diferentes modelos para expresar funciones de fase. Cuando una función de fase dispersa la mayoría de los rayos en el hemisferio hacia el que se dirige originalmente la luz, diremos que hace dispersión hacia adelante, en el caso opuesto diremos que hace dispersión hacia atrás (Figura 2.3). Cuando la dispersión se realiza de forma equiprobable en todas las direcciones, se dice que la dispersión es isotrópica.

En el código se va a usar la función de fase de Henyey-Greenstein (2.6), que hace dispersión hacia adelante si su parámetro de asimetría g_{HG} cumple que $g_{\mathrm{HG}} > 0$, hace dispersión hacia atrás si $g_{\mathrm{HG}} < 0$ y es isotrópico si $g_{\mathrm{HG}} = 0$. Como se puede deducir el caso isotrópico es un caso particular de Henyey-Greenstein.

$$p_{HG}(cos\theta) = \frac{1}{4\pi} \frac{1 - g_{\mathrm{HG}}^2}{(1 + g_{\mathrm{HG}}^2 + 2g_{\mathrm{HG}}(cos\theta))^{3/2}} \tag{2.6}$$

En esta ecuación θ describe el ángulo entre la dirección incidente y la saliente.

Figura 2.3: Dispersión hacia adelante y hacia atrás.

2.2.3. Ecuación de Renderizado de Volumen (VRE)

Por completitud se debe considerar que además del medio participativo y de la luz, puede existir una superficie sólida al final del rayo, cuando ($\mathbf{y} = \mathbf{z}$). Se suma entonces un término que represente la superficie al final del rayo compuesto por el producto de la radiancia saliente $L_o(\mathbf{z}, \omega)$ en la superficie por la transmitancia a lo largo del rayo.

$$L(\mathbf{x}, \omega) = \left[\int_0^z T(\mathbf{x}, \mathbf{y}) \left(\mu_a(\mathbf{y}) L_e(\mathbf{y}) + \mu_s(\mathbf{y}) L_s(\mathbf{y}, \omega) \right) dy \right] + T(\mathbf{x}, \mathbf{z}) L_o(\mathbf{z}, \omega) \tag{2.7}$$

Esta ecuación suele ser referida como la Ecuación de Renderizado de Volumen (VRE, por sus siglas en inglés).

Integración de Montecarlo

La integración de Montecarlo permite estimar el valor de la integral de una función f en un dominio D, que típicamente es difícil de evaluar analíticamente. Para conseguirlo se toman puntos en D aleatoriamente siguiendo alguna distribución de probabilidad, se evalúa la función f en dichos puntos y se estima la integral como un promedio ponderado en el que cada punto se divide por la densidad de probabilidad en el mismo.

Por más información sobre el tema ver (Weinzierl, 2000).

Estimador de VRE

A continuación se toma una sola muestra para usar una notación más clara, se usa <> encerrando $L(\mathbf{x}, \omega)$ para indicar que se habla de un estimador de Monte Carlo. El mismo se puede definir como $< F >= \frac{f(\mathbf{x})}{p(\mathbf{x})}$, donde $f(\mathbf{x})$ es el integrando de F y $p(\mathbf{x})$ la PDF de muestrear los puntos $\mathbf{x}$.

$$< L(\mathbf{x}, \omega) >= \frac{T(\mathbf{x}, \mathbf{y})}{p(\mathbf{y})} \left[\mu_a(\mathbf{y}) L_e(\mathbf{y}) + \mu_s(\mathbf{y}) L_s(\mathbf{y}, \omega)\right] + \frac{T(\mathbf{x}, \mathbf{z})}{P(\mathbf{z})} L_o(\mathbf{z}, \omega) \quad (2.8)$$

Se puede aplicar dicho estimador (2.8) a la VRE de forma directa. Se evalúa el integrando en una distancia y elegida aleatoriamente, según alguna técnica de muestreo de distancia (Sec. 2.3), dividida por la densidad de probabilidad de muestrear dicha distancia $p(\mathbf{y})$.

Si la distancia muestreada está más allá de la superficie más cercana ($\mathbf{y} > \mathbf{z}$) se evalúa el segundo término (la radiancia de la superficie) dividida la probabilidad de muestrear una posición detrás de la superficie $P(\mathbf{z})$. En este caso los dos términos no ocurren nunca al mismo tiempo, esta terminología se utiliza en (Novák, Georgiev, Hanika, Křivánek, y Jarosz, 2018).

Muestrear una distancia y evaluar transmitancia es la base de la mayoría de los algoritmos de renderizado de volúmenes.

2.3. Muestreo de distancia

A continuación se mencionan métodos para obtener la distancia que un fotón debe viajar en un medio participativo antes de interactuar con él.

Existen dos familias de métodos: los análogos y los no análogos. Esta terminología viene del transporte de neutrones y se refiere a si el método se adhiere o no al proceso físico real. Una partícula que sigue un camino generado por un método análogo se dispersa o absorbe estrictamente según las propiedades ópticas del material. Estos métodos producen las muestras de camino libre, que son análogas a las trayectorias que toman los fotones en el mundo real.

Por otro lado, los métodos no análogos se desvían del proceso físico y por lo tanto sus distancias pueden ser muy diferentes de las distancias de los caminos libres reales. Aún así se pueden conseguir resultados sin sesgo si las muestras son ponderadas para contrarestar las desviaciones del proceso físico.

Se analizan en esta sección los métodos análogos, o sea cómo muestrear la distancia de camino libre hacia la próxima interacción con el medio. Los métodos no análogos buscan mejor performance a costo de menor realismo y no se comentan en este informe, un ejemplo de ellos es Weighted Delta Tracking (Morgan y Kotlyar, 2015).

Previamente se vio que las interacciones de radiancia con el medio están descritas por la Ecuación RTE (2.1). Integrando espacialmente se obtiene primeramente la relación entre la radiancia incidente y la radiancia saliente. Luego ambas se relacionan usando la transmitancia, que cuantifica la probabilidad que un fotón llegue a viajar más allá de una distancia dada t.

$$T(t) = e^{-\tau(t)} = e^{-\int_0^t \mu_t(s)ds} = P(X > t) \tag{2.9}$$

Se puede observar que la probabilidad complementaria de que un fotón interactúe antes de alcanzar t es igual a la definición de función de distribución acumulada (CDF, por sus siglas en inglés), que está representada como F en la Ecuación (2.10).

$$F(t) = P(X \leq t) \tag{2.10}$$

Como estas probabilidades sumadas dan 1 ($F(t)+T(t) = 1$), se puede definir la CDF para muestrear distancias como la probabilidad complementaria de la transmitancia.

$$F(t) = 1 - T(t) = 1 - e^{-\tau(t)} \tag{2.11}$$

Si se implementase Importance Sampling o Multiple Importance Sampling (MIS), se calcularía el PDF de las muestras individuales derivando el CDF.

$$p(t) = \frac{dF(t)}{dt} = \frac{d}{dt}(1 - e^{-\tau(t)}) = \mu(t)e^{-\tau(t)} \tag{2.12}$$

Finalmente se calcula la función cuantil (inversa de la CDF) para tener una forma de llegar desde un número aleatorio a una distancia muestreada t. Esto resulta en una Ecuación (2.14), en la que el espesor óptico $\tau(t)$ está en el lado izquierdo y un logaritmo natural en el derecho. Dicho de otra forma, estamos buscando una distancia t en la que el espesor óptico equivale al logaritmo negado.

$$\xi = 1 - e^{-\tau(t)} \tag{2.13}$$

$$\tau(t) = \int_0^t \mu_t(s)ds = -ln(1 - \xi) \tag{2.14}$$

La dificultad del cálculo de la distancia t depende de qué forma tenga el coeficiente de extinción μ_t. Según el caso hay métodos analíticos, semi-analíticos y aproximados. Los métodos analíticos sólo sirven para medios homogéneos ya que en cada punto de este se sabe de antemano qué coeficiente tiene, en cambio los semi-analíticos y aproximados van a ser usados para medios heterogéneos, en los cuales el coeficiente varia espacialmente.

A continuación se mencionan algunos de estos métodos.

2.3.1. Método analítico

Algunas funciones de extinción permiten expresar el espesor óptico τ (2.4) de forma simple. Ejemplo de esto ocurre en un medio homogéneo, donde la función de extinción $\mu_t(t)$ es simplemente una constante.

Muestreo de densidad

El espesor óptico (2.4) en el caso de un medio con μ_t constante es una función $\tau(t) = \mu_t t$.

Si sustituimos en la función cuantil (2.14), se puede resolver fácilmente dividiendo el lado derecho por la constante de extinción, lo que nos devuelve el CDF invertido.

$$t = F^{-1}(\xi) = -\frac{ln(1-\xi)}{\mu_t} \tag{2.15}$$

Finalmente se sustituye en la Ecuación (2.12) para llegar a $p(t)$

Resumiendo, para muestrear un volumen homogéneo entonces se siguen los siguientes pasos.

1. Seleccione un número aleatorio ξ
2. Asigne $t = -\frac{ln(1-\xi)}{\mu_t}$
3. Realice $p(t) = \mu_t e^{-t\mu_t}$

Muestrear la distancia en volúmenes homogéneos es simple: dada una posición, elegimos un número aleatorio y computamos la distancia a la primera colisión y luego el PDF, de ser necesario. Se resuelve analíticamente y es fácil de implementar.

2.3.2. Método semi-analítico - Regular Tracking

Los siguientes métodos se consideran para métodos heterogéneos. En caso que un volumen se pueda representar como una colección de volúmenes simples, por ejemplo una grilla de vóxeles en el que cada celda tiene un μ_t (constante dentro de la celda pero diferente al de las demás) la integración del lado izquierdo de (2.14) se reemplaza por una suma (2.16), donde el Δ_i corresponde a la distancia recorrida por el rayo en el voxel i, siendo k el último voxel que recorre.

$$\sum_{i=1}^{k} \mu_{t,i}(s)\Delta i = -ln(1-\xi) \tag{2.16}$$

El método semi-analítico toma la siguiente forma:

1. Se selecciona un número aleatorio ξ.

2. Mientras el lado izquierdo de (2.16) sea menor que el lado derecho, sumo y se continúa a la siguiente intersección.

3. Se encuentra la ubicación exacta analíticamente en el último voxel, que se identifica cuando el lado izquierdo de (2.16) pasaría a ser mayor que el derecho.

La principal desventaja de este método es la necesidad de calcular todas las interacciones con interfaces que separan las regiones homogéneas, lo que puede ser costoso computacionalmente. Se puede visualizar en la Figura 2.4.

2.3.3. Método aproximado

En algunos casos se justifica muestrear los caminos libres aproximadamente, favoreciendo la velocidad de ejecución.

Muestreo uniforme - Ray Marching

La idea del Ray Marching es ignorar los límites de las secciones de μ_t constante y caminar a lo largo de un rayo con una distancia constante. La suma del lado izquierdo en (2.16) ya no va a corresponder exactamente a la integral de (2.14) pero es fácil de implementar y evita la necesitad de encontrar las interfaces.

El algoritmo es análogo al del regular tracking con la diferencia que los pasos a lo largo del rayo tienen distancia fija.

Como se puede observar en la Figura 2.5 se camina a lo largo del rayo tocando sólo algunos de los vóxeles y se asume que μ_t es constante en cada paso. Cuando el lado derecho de la ecuación pasa a ser mayor que el izquierdo se busca en el último vóxel la ubicación exacta analíticamente.

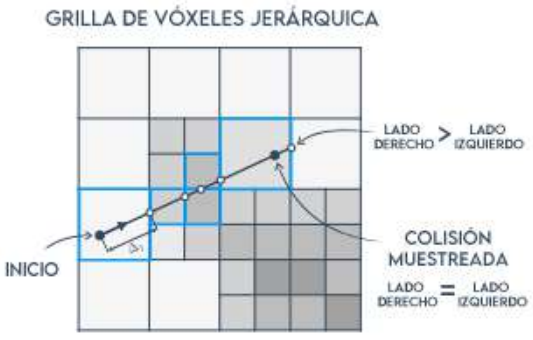

Figura 2.4: Regular tracking.

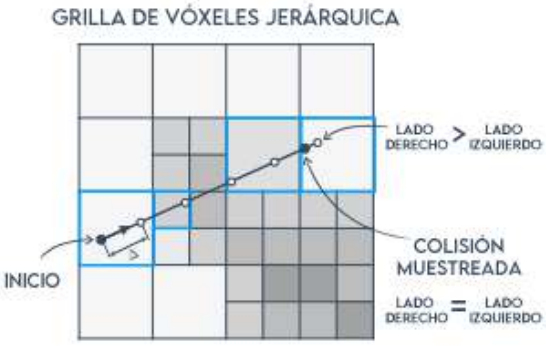

Figura 2.5: Ray marching.

Se dice que el Regular Tracking (Sec. 2.3.2) es insesgado ya que resuelve de forma exacta la integral (2.14), lo que no ocurre en este método ya que es

aproximado. El error de esta aproximación va a depender de la variación entre secciones: si es muy grande va a ser mayor.

El ray marching puede aplicarse a cualquier medio con coeficiente de extinción μ_t potencialmente variable en el espacio; sin embargo, la suposición de una variación constante, lineal, cuadrática, etc., de μ_t conduce a una sobreestimación sistemática de la transmitancia, como se describe formalmente en la desigualdad de de Jensen para funciones convexas (Jensen, 1906). Por lo tanto, el ray marching puede considerarse como un estimador sesgado del valor esperado.

Los tres métodos vistos hasta ahora tienen en común que eligen un valor aleatorio de espesor óptico y luego caminan a lo largo del rayo buscando la ubicación correspondiente.

Muestreo adaptativo

En este caso el largo del salto es variable y se ajusta según las propiedades locales del medio. La idea es que se tengan más muestras en regiones de alta densidad y menos en regiones más uniformes.

Tiende a ser más eficiente en capturar detalles aunque introduce el problema de cómo elegir los saltos de forma correcta.

2.3.4. Algoritmos de Colisiones-Nulas (Null-Collision)

Este procedimiento se basa en las llamadas Colisiones Nulas, y tiene su origen en los modelos propuestos en física para la simulación del transporte de neutrones y permite tener un muestreo insesgado. La idea de los algoritmos es agregar materia ficticia al medio, que no afecta a la luz.

Delta Tracking

También conocido como Woodcock Tracking,Pseudo Scattering, Hole Tracking ó Null-Collision method entre otros. Se suele explicar en artículos científicos usando argumentos motivados por la física, en los que se razona sobre la presencia de materia fictica y cómo la luz interactúa con ella (Butcher y Messel, 1958)(Bertini, 1963)(Skullerud, 1968).

La materia ficticia que se agrega tiene Albedo igual a 1 (ver Glosario) por lo que no se pierde radiancia al colisionar con ella, y su función de fase es una función delta (la dirección no cambia como resultado de una colisión con ella). Dicha partícula genera las llamadas Null-Collisions, en las que toda la luz continúa como si no hubiese colisionado. Ver Figura 2.6.

Además del coeficiente de extinción μ_t del medio real, se define el coeficiente de null-collision μ_n que representa el medio ficticio. El coeficiente μ_n se ajusta espacialmente, de forma que la suma de los dos coeficientes sea constante en el medio.

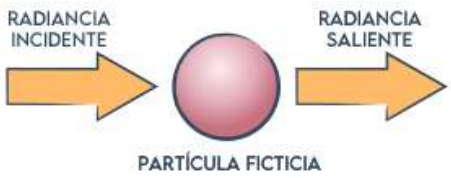

Figura 2.6: Colisión nula en Delta Tracking. La radiancia saliente es igual a la entrante en cantidad y dirección.

$$\bar{\mu}(\mathbf{x}) = \mu_t(\mathbf{x}) + \mu_n(\mathbf{x}) \tag{2.17}$$

Este valor $\bar{\mu}$ (2.17) se llama mayorante de la función de extinción, ó simplemente mayorante. Al ser μ_t y μ_n valores no negativos, el mayorante va a actuar como cota superior de μ_t, como se puede observar en las Figuras 2.7, 2.8 y 2.9.

Para muestrear el camino libre, se usa la función cuantil derivada del mayorante constante para obtener la distancia a la primera colisión tentativa. Luego se decide si la colisión es real (Figura 2.8) o nula (Figura 2.7). Esto se realiza eligiendo al azar un número entre 0 y 1, dado que la probabilidad de una colisión real es $\frac{\mu_t}{\bar{\mu}}$ y la probabilidad de una colisión nula es $\frac{\mu_n}{\bar{\mu}}$. Este proceso se repite hasta que se encuentre una colisión real.

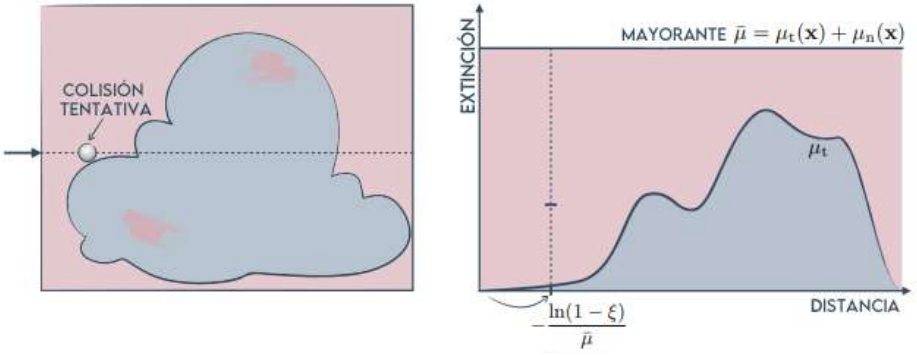

Figura 2.7: Colisión nula. En este caso el punto muestreado se encuentra por encima de $\mu_t(\mathbf{x})$.

La distancia a la primera colisión real representa la muestra de camino libre. Así funciona Delta Tracking: se avanza muestreando distancias en el medio combinado y probabilísticamente clasificando cada colisión como real o nula. Este algoritmo se puede interpretar como una forma de rejection sampling.

La elección del mayorante tiene un impacto en el algoritmo. Es preferible tener un mayorante ajustado ya que esto deriva en menos colisiones nulas. Análogamente, si el mayorante es demasiado grande van a existir más colisiones nulas

(Figura 2.9). Mientras menos colisiones nulas existan más eficiente es el algoritmo.

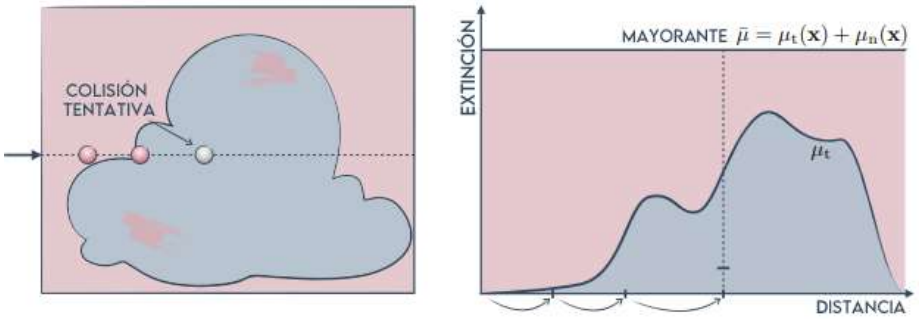

Figura 2.8: Colisión real. En este caso el punto muestreado se encuentra por debajo de $\mu_t(\mathbf{x})$. Nótese que previamente a esta colisión existieron dos colisiones nulas.

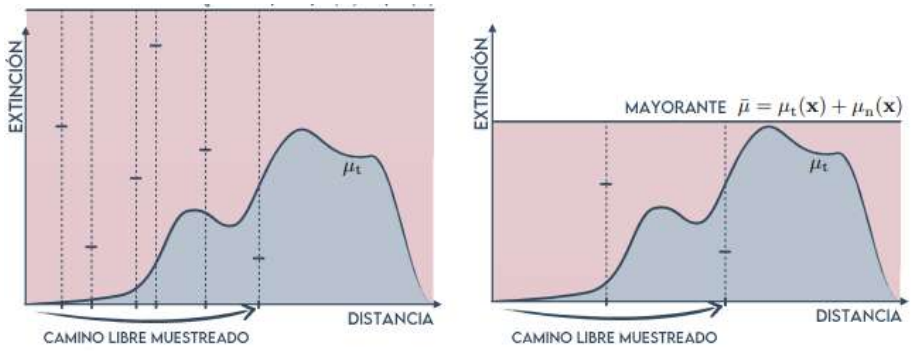

Figura 2.9: Mayorante grande a la izquierda y ajustado a la derecha. En el caso del mayorante grande se tienen más colisiones nulas. Cuando el mayorante está ajustado se tiene el caso opuesto.

2.4. Muestreo de transmitancia

A continuación se mencionan algunos métodos para estimar la transmitancia (2.3) en medios participativos.

2.4.1. Estimadores integrando espesor óptico

Los razonamientos que se comentan a continuación son análogos a los usados en la Sec. 2.3 para los métodos analítico, semi-analítico y aproximado. En el caso

del método analítico, cuando existe un medio homogéneo se va a tener que el μ_t es constante y por lo tanto el espesor óptico es lineal en t, dado que la integral $\int_0^t \mu_t ds = \mu_t t$. Es trivial hallar la transmitancia sustituyendo $\mu_t t$ en lugar del espesor óptico en (2.3).

En el caso semi-analítico se suma el espesor óptico τ (2.4) de cada volumen individual ponderado por la distancia recorrida en cada uno y en el método aproximado se toma el espesor óptico en cada punto muestreado.

2.4.2. Estimadores de Null-Collisions

Delta Tracking

Delta Tracking puede ser usado como un estimador de largo de camino para estimar la transmitancia de la siguiente forma:

1. Se realiza el trackeo y se guarda la distancia a la primera colisión real.
2. Si la distancia es más corta que la distancia del segmento de interés (el segmento A-B en la Figura 2.10) la transmitancia se estima como 0.
3. Si la muestra de Delta Tracking excede la distancia de B, se estima como 1.

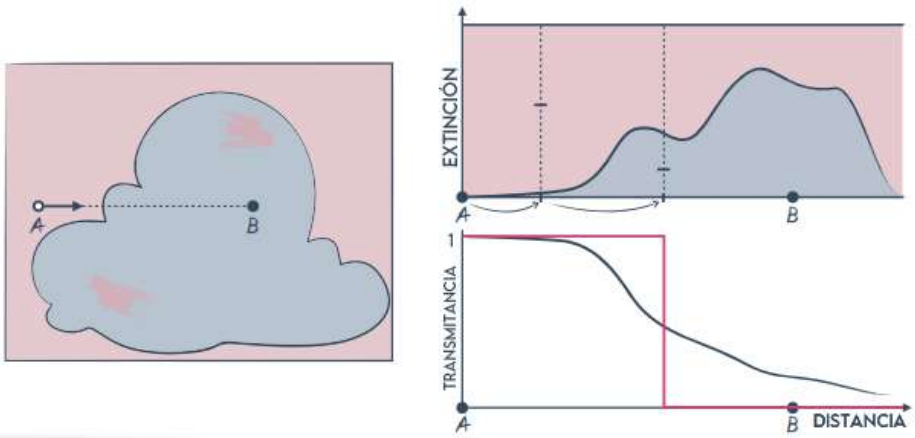

Figura 2.10: Delta Tracking. Abajo a la derecha se observa la transmitancia estimada luego de una muestra. Originalmente la transmitancia se estima en 1. La bajada se da por el resultado de la muestra representada por el gráfico de arriba a la derecha. Si la muestra tomada hubiese estado más allá de B, la transmitancia estimada se habria mantenido en 1.

Se puede refinar la función estimada trackeando varias veces las distancias (Figura 2.11 con 3 estimaciones) en este proceso iterativo en cada paso se promedian las instancias de Delta Tracking según la cantidad de muestras, por ejemplo

en el caso de 3 muestras cada bajada en la transmitancia corresponde a 1/3. En la mayoría de casos esto es muy costoso e ineficiente computacionalmente.

Durante cada trackeo se obtiene bastante información del medio (μ_t en cada colisión tentativa), pero se reduce el resultado a un valor binario al clasificar la colisión como real o nula probabilísticamente.

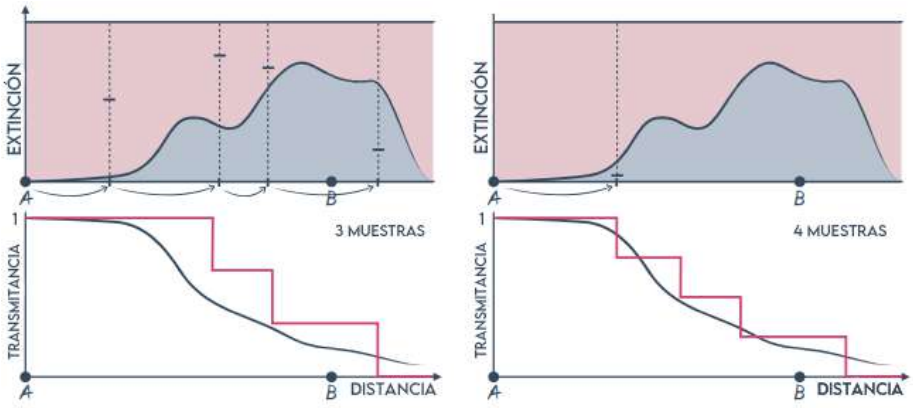

Figura 2.11: Delta Tracking. A la izquierda 3 muestras y a la derecha 4. A la izquierda, como la muestra tomada está más allá de B, se estima la transmitancia como 1. Por eso en la gráfica de abajo hay un valor mayor a 0 en ese punto (en las muestras anteriores era 0). En la derecha la muestra está antes de B, por eso hay una bajada en la transmitancia estimada mostrada por la gráfica de abajo a la derecha.

Residual Ratio Tracking

También conocido como Ratio Tracking, este estimador busca reducir la varianza de Delta Tracking. La idea es eliminar la terminación aleatoria y reemplazarla por su probabilidad, lo que sirve como un peso.

En lugar de obtenerse una respuesta binaria, el algoritmo puntúa un peso: el producto de los cocientes entre el coeficiente de colisiones nulas μ_t y el mayorante μ_{maj} en todos los puntos visitados a lo largo del camino, ver Figura 2.12. La aproximación de la transmitancia es constante en bloques.

Tiene como ventaja la reducción de varianza comparado a Delta Tracking y la eficiencia al no necesitar tantos rayos para estimar de forma aproximada la transmitancia.

Observemos el impacto de la cantidad de material ficticio en la aproximación de la transmitancia resultante. En el caso que haya mucho material ficticio (Figura 2.13), las distancias entre muestras tienden a ser cortas, por lo que terminan siendo muchas y los pesos locales en cada colisión relativamente altos. La transmitancia se estima de forma bastante aproximada.

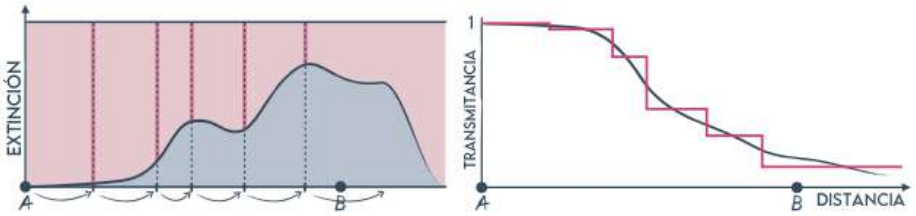

Figura 2.12: Ratio Tracking. En este caso cada distancia muestreada contribuye al estimado de la transmitancia (gráfica de la derecha).

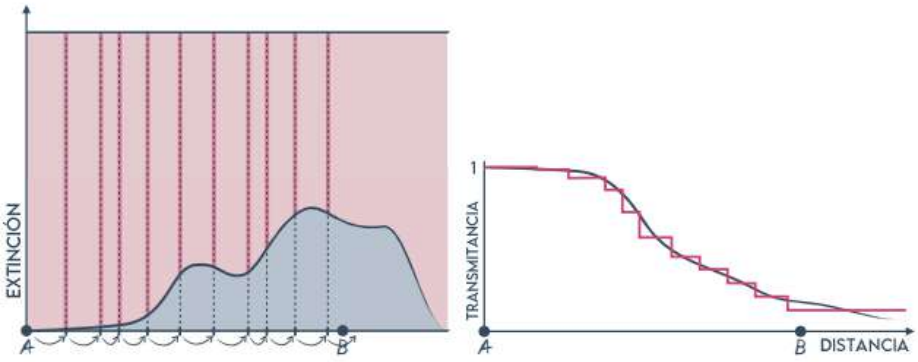

Figura 2.13: Ratio Tracking con mucho material ficticio.

Si no se agrega material ficticio, el cociente en la primera colisión es cero y la aproximación es nuevamente binaria, como en Delta Tracking. En práctica se suele usar Ratio Tracking al principio y una vez que los pesos se encuentren debajo de un cierto valor, se cambia a Delta Tracking usando la terminación del camino probabilísticamente.

2.5. Construcción de caminos

Finalmente se comentan técnicas usadas para combinar los métodos de muestreo anteriormente mencionados, tanto para la distancia (Figura 2.14) como para la dirección (Figura 2.15). En el caso del muestreo de distancia esta es proporcional a la transmitancia entre el punto original y el muestreado, pues ambas dependen de la densidad del medio. En tanto la dirección va a ser proporcional a la función de fase, ya que esta define la distribución direccional en el medio.

La principal referencia de esta sección es (Miller, Georgiev, y Jarosz, 2019).

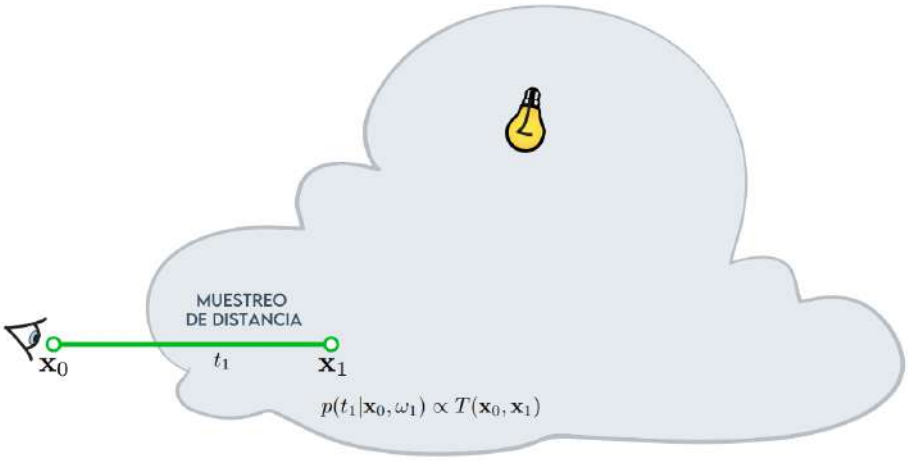

Figura 2.14: Muestreo de distancia.

2.5.1. Path Tracing

En la Ecuación (2.18) se observa el valor de un píxel j, o sea la integral de la radiancia $f_j(\bar{\mathbf{x}})$ de todos los posibles caminos $\bar{\mathbf{x}}$ que pasan por él, donde $\bar{\mathbf{x}}$ es la secuencia de $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k$ puntos de un camino que va desde la cámara a la escena.

$$I_j = \int_P f_j(\bar{\mathbf{x}}) d\bar{x} \tag{2.18}$$

Para calcular de forma aproximada el valor de I_j, se generan varios caminos $\bar{\mathbf{x}}_i$ y se promedia las radiancias f_j de los caminos. Esto se define en la Ecuación (2.19).

$$< I_j >= \frac{1}{N} \sum_{i=1}^{N} f_j(\bar{\mathbf{x}}_i) \tag{2.19}$$

A continuación se observa el cálculo de la radiancia de cada camino (2.20). Este incluye la función de fase o BSDF $f_s(\mathbf{x}_i, \mathbf{x}_{i-1}, \mathbf{x}_{i+1})$, el término de geometría $G(\mathbf{x}_i, \mathbf{x}_{i+1})$ (el cual describe la visibilidad entre diferentes puntos, o sea si existe algún cuerpo que hace imposible trazar un rayo entre $\mathbf{x}_i$ y $\mathbf{x}_{i+1}$, en este proyecto se simplifica a 1 ya que no existen volúmenes y superficies al mismo tiempo en la escena), la transmitancia $T(\mathbf{x}_i, \mathbf{x}_{i+1})$ y finalmente la radiancia emitida al terminar el camino en una fuente de luz $L_e(\mathbf{x}_k, \mathbf{x}_k - 1)$.

$$f_j(\bar{\mathbf{x}}) = \left[\prod_{i=1}^{k-1} f_s(\mathbf{x}_i, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}) G(\mathbf{x}_i, \mathbf{x}_{i+1}) T(\mathbf{x}_i, \mathbf{x}_{i+1}) \right] L_e(\mathbf{x}_k, \mathbf{x}_k - 1) \tag{2.20}$$

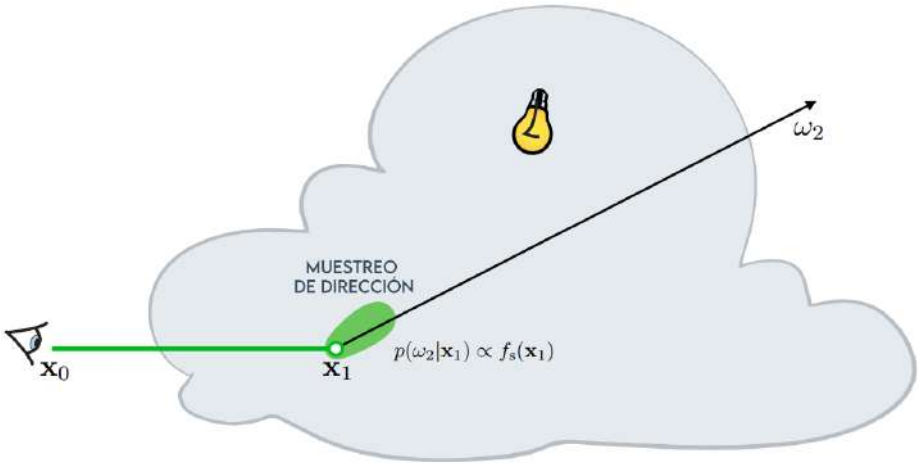

Figura 2.15: Muestreo de dirección.

Muestreo de caminos Unidireccional

En este método se toma una serie de decisiones de muestreo para la distancia y dirección hasta que se llegue a una fuente de luz. Se selecciona una nueva dirección en cada colisión a partir de la PDF. Si la luz es pequeña termina habiendo mucha varianza ya que es difícil llegar a ella. También tiene como limitante que no se puede usar para renderizar luces puntuales. Se dice que es unidireccional ya que los caminos siempre van desde la cámara a las fuentes de luz, a diferencia de los métodos bidireccionales.

Next-Event Estimation

En Next-Event Estimation (NEE), en cada interacción se muestrea directamente una fuente luminosa tirando un rayo hacia ella: $L_{dNEE}(\mathbf{x}, \omega, \mathbf{y})$ en la Ecuación (2.21), siendo $\mathbf{x}$ el último punto del camino construido hasta ahora; ω el vector dirección entre $\mathbf{x}$ e $\mathbf{y}$ y $\overline{\omega}$ la dirección opuesta; por último $\mathbf{y}$ es el punto de la luz elegida.

$$L_{dNEE}(\mathbf{x}, \omega, \mathbf{y}) = L_o(\mathbf{y}, \overline{\omega}) * p(\mathbf{y}) * T(\mathbf{x}) * T(\mathbf{x}, \mathbf{y}) * f_p(\omega, \overline{\omega}) \tag{2.21}$$

Se debe ponderar la radiancia que la luz emite en dirección a $\mathbf{x}$: $L_o(\mathbf{y}, \overline{\omega})$, según la probabilidad de que un punto sea elegido en ella $p(\mathbf{y})$ (depende de la técnica de muestreo de luces que se haga, en este proyecto se trabaja con una única luz puntual por lo que $p(\mathbf{y}) = 1$ siempre), la transmitancia acumulada a lo largo del camino desde la cámara hasta el punto actual $T(\mathbf{x})$, la transmitancia entre $\mathbf{x}$ y la luz $\mathbf{y}$: $T(\mathbf{x}, \mathbf{y})$, así como la función de fase $f_p(\omega, \overline{\omega})$.

En escenas con múltiples luces se debe elegir también entre muestrear una luz

por iteración probabilísticamente, o varias, siendo cada contribución ponderada adecuadamente.

NEE reduce la varianza ya que muestrea la luz directamente, en vez de buscar eventualmente llegar a ella, por lo tanto se converge más rápidamente a una imagen con poco ruido. También es de gran ayuda para escenas con fuentes de luz pequeñas. Por otra parte, introduce costo computacional ya que se deben computar los rayos de sombra por cada iteración y se debe determinar la visibilidad de la luz desde los puntos en escenas con geometría.

Ratio Next-Event Estimation

Nace como una extensión de NEE donde la contribución de cada rayo de sombra se pondera por el ratio entre el coeficiente de extinción a ese punto y el mayorante, de forma similar a Ratio Tracking.

$$L_{dRNEE}(\mathbf{x}, \omega, \mathbf{y}) = \frac{\mu_t}{\bar{\mu}} L_{dNEE}(\mathbf{x}, \omega, \mathbf{y}) \tag{2.22}$$

Este ratio se comporta como una probabilidad que permite reducir varianza al asegurarse que la relevancia de cada muestra sea relativa al contexto del volumen entero.

Capítulo 3

Desarrollo

En este proyecto se ha desarrollado un *renderer* de medios participativos, donde se busca hacer un balance entre eficiencia en costo computacional y dificultad de programación. Se presenta un esquema detallado de las elecciones arquitectónicas y decisiones de diseño que permiten trabajar analíticamente los medios homogéneos, y trabajar utilizando Ray Marching, Delta Tracking y Ratio Tracking los medios heterogéneos.

3.1. Arquitectura

Se comienza viendo los requisitos funcionales y no funcionales que sirvieron de guía a la hora de diseñar la aplicación.

3.1.1. Requisitos funcionales

1. **Procesar volúmenes 3D heterogéneos:** Para poder ejecutar integradores de medios heterogéneos se va a necesitar una forma de representar estos volúmenes en memoria. A su vez dicha representación debe ser eficiente en tiempo de acceso y uso de memoria.

2. **Soportar múltiples integradores:** A modo de poder probar y comparar los diferentes integradores desarrollados, se debe poder elegir cual usar antes de cada ejecución, el desarrollo de nuevos integradores no debe afectar los previamente existentes.

3. **Datos de entrada configurables:** Con el objetivo de variar parámetros en las imágenes y probar nuevamente de forma rápida se va a necesitar aceptar un archivo con los datos de entrada en un formato de texto plano.

4. **Exportar imágenes:** Finalmente se va a necesitar exportar las imágenes a disco en algún formato estándar.

3.1.2. Requisitos no funcionales

1. **Performance:** Es clave hacer un buen uso de los recursos computacionales disponibles de forma de reducir el tiempo de procesamiento para obtener una imágen, tanto para la producción de imágenes finales como para el desarrollo de los diferentes integradores.
2. **Escalabilidad:** Incrementar los diferentes parámetros de entrada debería ser soportado por la aplicación para, por ejemplo, producir imágenes con más resolución o caminos por píxel.

3.1.3. Diagrama

En la Figura 3.1 se observa una vista de alto nivel de la aplicación.

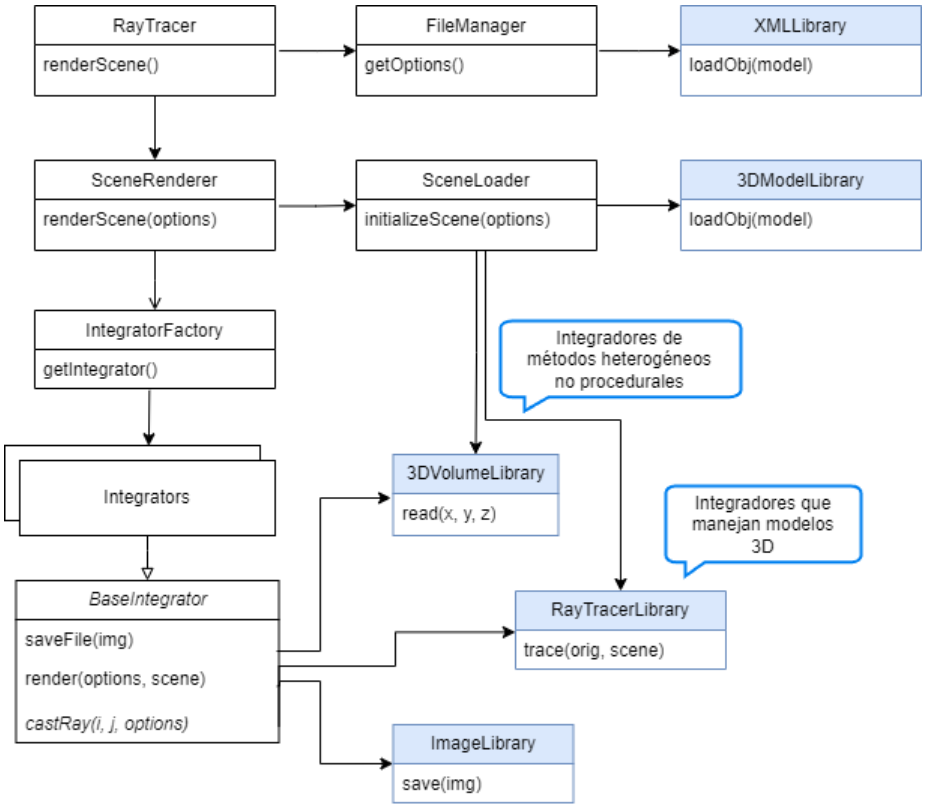

Figura 3.1: Diagrama de la arquitectura del sistema. Los componentes con encabezado celeste son librerías de terceros que se importan como plugins al proyecto.

La clase *RayTracer* es el punto de entrada de la aplicación. Él se va a encargar de obtener la configuración del usuario, mediante el FileManager, y pasarla

al *Scenerenderer*. A continuación este va a pedirle al *SceneLoader* cargar los archivos de modelos 3D y grillas 3D de densidad variable. Luego, a partir de los parámetros establecidos por el usuario va a pedir un integrador a *IntegratorFactory*. Dicho integrador además implementa *BaseIntegrator*, que ejecuta la lógica para dibujar la imagen, siendo el método abstracto *castRay(i, j, options)* el que contiene la lógica específica de cada integrador. Según la necesidad de cada uno de ellos se va invocar la librería para manejar volúmenes 3D de densidad variable o el ray tracer para superficies 3D. Finalmente se exporta la imagen resultante usando la librería correspondiente.

3.2. Diseño

Usando de base la arquitectura mencionada anteriormente se procede a tomar decisiones de diseño como son la elección de las librerías necesarias, así como definir las técnicas utilizadas para tratar los requisitos no funcionales.

3.2.1. Librerías

Para la elección de las librerías se optó por elegir entre aquellas que sean de uso libre y gratuito, además de ser de código abierto. Se pueden visualizar las elecciones en la Figura 3.1 sustituyendo *XMLLibrary* por pugixml, *3DModelLibrary* por tinyobjloader, *3DVolumeLibrary* por NanoVDB, *RayTracerLibrary* por Embree e *ImageLibrary* por FreeImage.

NanoVDB - Librería para volúmenes 3D

Es fundamental definir una estructura de datos eficiente para almacenar datos de volumen. Se investigaron las opciones de OpenVDB (*OpenVDB*, 2012) (por más detalle ver (Museth, Avramoussis, y Bailey, 2019)) y NanoVDB (Museth, 2021), que nace como una versión simplificada de OpenVDB. Finalmente se eligió usar NanoVDB para medios heterogéneos, que entrega una estructura de datos eficiente y compacta diseñada para volúmenes con variabilidad. NanoVDB es particularmente adecuado para manejar grandes conjuntos de datos con estructuras complejas y puede ser descargado en (*NanoVDB*, 2023).

Corre tanto en CPU como en GPU y al ser una extensión de OpenVDB se pueden encontrar múltiples modelos disponibles para uso libre en internet, si bien el formato es diferente al que tienen las grillas de NanoVDB, éste tiene una herramienta para convertirlas grillas de OpenVDB a su formato.

Un punto clave a la hora de elegir esta librería es la simplicidad de su uso, dado sólo es necesario importar un archivo, si bien existen otros que se puede optar por importar según las herramientas que se necesite usar. Otro punto clave es que esta librería es usada por PBRT (*pbrt4*, 2023). Existe un proyecto en Github llamado pbrt4-scenes con escenas de prueba donde se pueden conseguir modelos en el formato de NanoVDB (*pbrt4-scenes*, 2023).

Embree - Librería de Ray Tracing

Embree (Wald, Woop, Benthin, Johnson, y Ernst, 2014) es una librería desarrollada por Intel para Ray Tracing en CPU, que permite cargar geometría y dada una posición y dirección de origen recorrer la escena. Si bien no es estrictamente necesario usar Embree para escenas con geometría tan simple como las que se terminaron usando en los integradores desarrollados, se aprovechó el hecho de que la lógica para su uso ya estaba implementada en el proyecto.

FreeImage - Librería para manejo de Imágenes

Para exportar las imágenes se usó FreeImage (*FreeImage*, 2023), permite exportar el buffer en memoria que genera nuestra aplicación a formatos populares, entre los que optamos por PNG. Es open source y multiplataforma.

pugixml - Librería para manejo de archivos XML

La configuración ingresada por el usuario para el dibujado de la escena se realiza mediante un archivo XML llamado *configrenderer.xml*, que se encuentra ubicado en la misma carpeta que el ejecutable. Para leer su configuración se hizo uso de la librería pugixml (*pugixml*, 2023) de modo de facilitar la implementación de dicho código.

tinyobjloader - Librería para importar modelos 3D

Algunos integradores usan modelos 3D de formato OBJ con el objetivo de representar medios homogéneos o heterogéneos generados proceduralmente. Para importar dichos archivos se importó el plugin tinyobjloader (*tinyobjloader*, 2023).

3.2.2. Requisitos no funcionales

Performance

La aplicación corre en CPU por lo que va a ser esencial hacer uso de paralelismo en el cómputo, dividiendo la imagen en partes que cada núcleo puede procesar independientemente. El procesamiento de imágenes que se realiza en este proyecto es un buen caso de uso de paralelización, pues al dibujar un píxel no se necesita información de los píxeles adyacentes.

Se implementó dicha técnica dividiendo la imagen en bloques horizontales con igual altura, que además es configurable. Dichos bloques van a ser tomados por hilos de CPU a medida que terminen de ejecutar el bloque que tienen asignado. Los hilos se crean a razón de uno por núcleo de procesador de forma de maximizar su uso y minimizar el tiempo de ejecución. Un ejemplo de esto es la Figura 3.2. Notar que en este caso hay 8 bloques aunque podría haber más. También la cantidad de núcleos de procesador es independiente: podría ejecutarse con varios o uno sólo indistintamente.

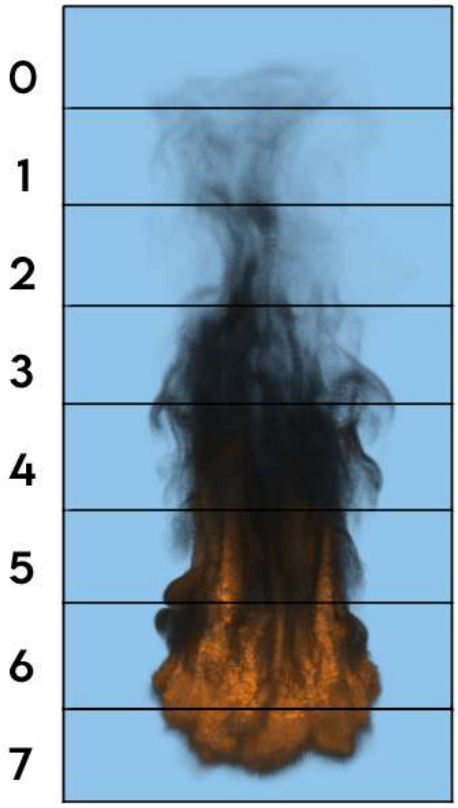

Figura 3.2: Ejemplo de imagen dividida en bloques horizontales. En este caso la altura de cada bloque es una octava parte de la altura de la imagen original. El ancho siempre es igual al ancho de la imagen.

Otro punto clave es el uso eficiente de los acceso a memoria, que es algo particularmente importante en el caso de los volúmenes de densidad variable espacialmente. En este punto nos va a ayudar NanoVDB.

Escalabilidad

Es clave tener un archivo de configuración con suficientes parámetros, tratando de tener la menor cantidad de valores fijos en el código. De esta forma si se implementa un nuevo integrador o técnica opcional se evita tener que volver a validar los casos previamente implementados.

El paralelismo implementado también fue clave en la escalabilidad, pues dibujar imágenes más grandes incrementa linealmente el tiempo de ejecución.

Esto ocurre debido a que el procesamiento agregado genera nuevos bloques horizontales para procesar la imagen, o agranda los bloques ya existentes.

3.3. Implementación

El código se puede descargar en (*eluna-renderer*, 2023).

Se usó como código base un proyecto desarrollado por quien suscribe para la unidad curricular Computación Gráfica Avanzada (*Computación Gráfica Avanzada*, 2023) dictada por la UdelaR. El código permite leer archivos XML de configuración, provee lógica para renderizar imágenes y posibilita la paralelización del cómputo.

Se realizaron varios integradores siguiendo las técnicas vistas en secciones anteriores del informe, a continuación se comentan detalles de su implementación.

3.3.1. Muestreo de densidad

Este integrador para medios homogéneos es el más fácil de implementar y sirve como medio de partida para los demás. Por cada píxel de la imagen se lanza uno o varios rayos hacia la escena y en las intersecciones con el modelo de una esfera se guardan la primera y última posición. Dichas intersecciones se calculan usando Embree (Sec. 3.2.1).

Para este integrador así como para los demás que utilizan una esfera en la escena se siguió como guía (*Volume rendering for developers: Foundations*, 2023). Además en el informe hay imágenes de dicho sitio web, editadas traduciendo el texto a español para ilustrar los diferentes conceptos.

3.3.2. Ray marcher con Next Event Estimation

Como siguiente integrador se construyó un Forward Ray Marcher (Sec. 2.3.3), o sea un ray marcher que lanza rayos desde la cámara hacia la escena. La ventaja de ir en este sentido y no desde la escena hacia la cámara es que se puede terminar prematuramente el cómputo dadas determinadas condiciones, por ejemplo que el volumen se haya vuelto demasiado opaco en un momento dado. Se toman pasos de tamaño fijo entre interacción e interacción.

Como un ray marcher en un medio homogéneo no tiene sentido por sí sólo pues es más fácil estudiar estos medios analíticamente. En este caso toma sentido porque también se implementa Next Event Estimation (Sec. 2.5.1) para muestrear una luz puntual en cada interacción con el medio. Esta implementación nos sirve de base para cuando se trabaja con medios heterogéneos. Se puede visualizar este integrador en la Figura 3.3.

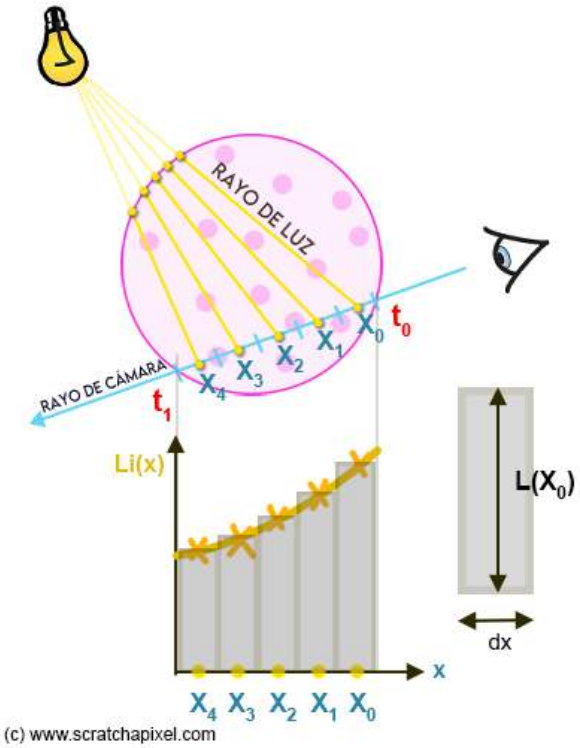

Figura 3.3: Ray marcher con NEE.

3.3.3. Ray marcher mejorado

Se implementa otro integrador similar al anterior con algunas mejoras: tiene en cuenta la función de fase, introduce Jittering, y puede terminar de forma temprana usando ruleta rusa.

Se agregó el término de la función de fase a la contribución de la luz, utilizando la función de Henyey-Greenstein (Sec. 2.2.2). Para evitar la generación de "bandas" como las mostradas en la Figura 3.4, se aleatorizó la posición en cada interacción dentro del largo de paso en el ray marcher (esto se denomina Jittering).

Se implementó el método de ruleta rusa, para así terminar temprano el Ray Marching en caso que el medio se vuelva demasiado opaco. De esta forma se puede terminar probabilísticamente un rayo si se encuentra debajo de un valor de transmitancia fijo dado. En caso de que sobreviva el rayo, su valor se multiplica por el inverso del complemento de la probabilidad de terminación, de forma de compensar por los rayos que terminen antes.

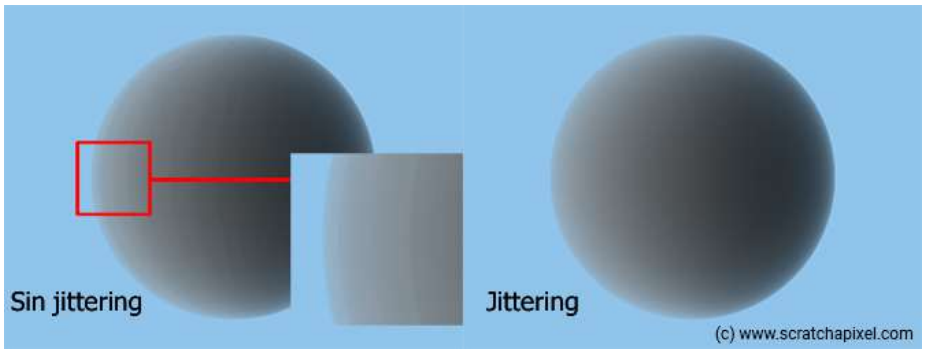

Figura 3.4: Jittering.

3.3.4. Ray marcher en medio heterogéneo

Finalmente se introducen medios heterogéneos para permitir dibujar escenas más interesantes. En este integrador se generan proceduralmente a partir de una textura 3D usando una función de ruido Perlin. Dicha función devuelve un valor de densidad para una posición en el espacio dado (ver Figura 3.5) y es utilizada tanto para calcular la transmitancia del rayo, así como para calcular el peso de los rayos de sombra en cada interacción con el medio.

Se usa el ray marcher del integrador anterior con el fin de buscar de forma aproximada los caminos que deben seguir los rayos disparados hacia la escena, y su correspondiente contribución a la imagen final.

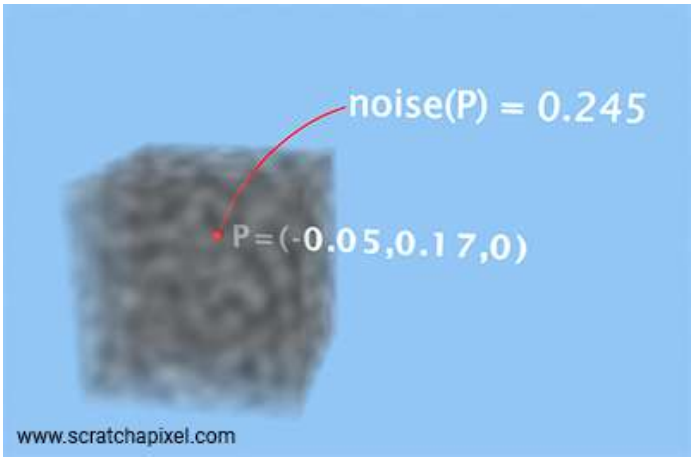

Figura 3.5: Función de ruido de Perlin.

3.3.5. Ray marcher usando NanoVDB

Este integrador obtiene la densidad a partir de un archivo de NanoVDB, para que puedan renderizarse medios participativos más complejos. Dicho integrador es un ray marcher con paso de tamaño fijo, además usa NEE, que utiliza ray marching en cada rayo de luz.

3.3.6. Ray marcher en medio emisivo

En este caso se introdujo emisión al medio, para ello se van a necesitar archivos NanoVDB que tengan una grilla de temperatura, además de la grilla de densidad que se usó para el integrador anterior.

La contribución por emisión en el medio se suma en cada punto de la grilla que tiene densidad y emisividad positivos estrictamente.

3.3.7. Delta Tracking

A continuación se comenta el primer integrador basado en colisiones nulas, el Delta Tracking. Se muestrea la distancia siguiendo el método descrito en 2.3.4, a partir de la nueva posición se elige al azar una posible interacción con el medio según el valor de la densidad local. En caso de colisión nula se sigue en la misma dirección y en absorción se termina y devuelve la radiancia acumulada. El caso más complejo es la dispersión, que cuando ocurre se muestrea una nueva dirección usando Henyey-Greenstein y se continúa el proceso hasta que se cumpla con alguna condición de salida. Esto ocurre cuando se salió del volumen ó se llegó a un límite de cantidad de dispersiones.

En este integrador todavía no se calculan contribuciones por iluminación directa, esto se agrega en el próximo integrador.

3.3.8. Residual Ratio Tracking

Para este integrador se implementó lo explicado en 2.4.2. Se acumula un valor de transmitancia por rayo que comienza valiendo 1 y se decrementa según la densidad del medio. Tanto la emisión del medio como el aporte por luces se escala por este valor.

Se agregó NEE. En este caso el rayo de sombra se lanza en cada evento de dispersión y se calcula la transmitancia desde el punto hasta la luz de forma similar al rayo principal, con la diferencia que dicho rayo de luz no se puede absorber ni dispersar.

Capítulo 4

Experimentación

En este capítulo se muestran imágenes obtenidas con el *renderer* de medios participativos realizado para la tarea bajo distintas condiciones y con los distintos integradores desarrollados.

Los tiempos mencionados fueron obtenidos en una PC con procesador Intel i7-7700K@4.20Ghz y 32GB RAM@2400MHz. Se usó Windows 10 como Sistema Operativo.

Para configurar las escenas se usan entre otros parámetros a σ_a y σ_s, que multiplican la densidad de los modelos, para obtener los valores de $\mu_a(\mathbf{x})$ y $\mu_s(\mathbf{x})$ correspondientes.

4.1. Medio Homogéneo

4.1.1. Muestreo de densidad

Se obtuvo varias imágenes para diferentes parámetros de σ_a (Figura 4.1). El tiempo de ejecución promedio fue de 448ms.

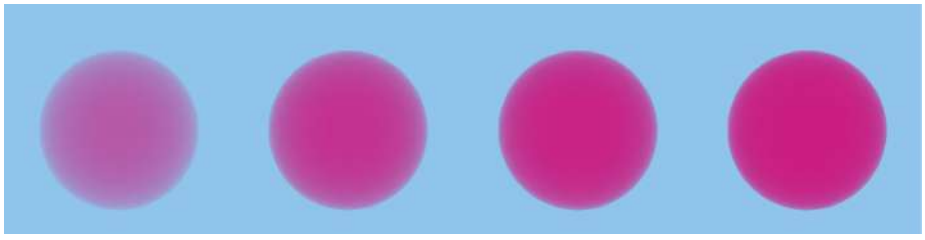

Figura 4.1: Muestreo de densidad. De izquierda a derecha se usaron valores de σ_a 0,2 0,4 0,6 *y* 0,8.

4.1.2. Ray marcher con Next Event Estimation

El tiempo de ejecución promedio fue de 1242ms.

En la Figura 4.2 se utilizó una luz puntual en la dirección $(0, 1, 0)$ con valor RGB $(0{,}9825, 0{,}225, 0{,}675)$, lo que cambia entre imágenes es σ_a.

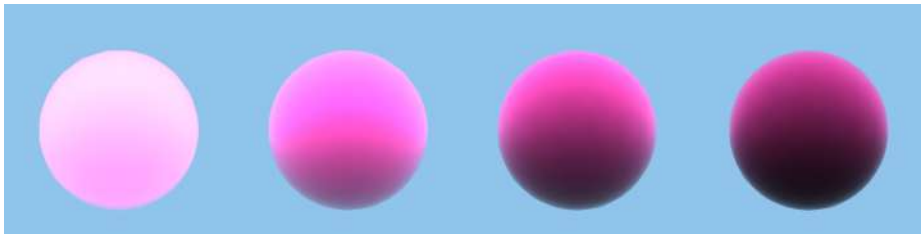

Figura 4.2: Ray marcher con NEE y luz superior. De izquierda a derecha se usaron valores de σ_a 0,2 0,4 0,6 y 0,8.

En el caso de la Figura 4.3 sólo se cambia la dirección de la luz y se dejan los demás parámetros fijos.

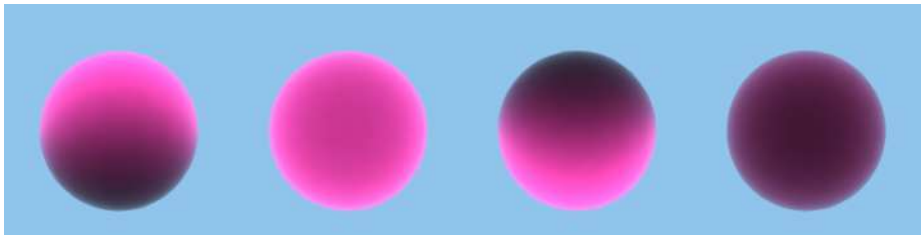

Figura 4.3: Ray marcher con NEE. De izquierda a derecha tiene luz superior, frontal, inferior y trasera. Para $\sigma_a = 0{,}6$ fijo.

4.1.3. Ray marcher mejorado

En la Figura 4.4 se muestran resultados para diferentes valores de la constante g_{HG} de Henyey-Greenstein. A partir de este *renderer* la densidad de la escena afecta la duración de la ejecución debido a la terminación temprana por ruleta rusa. En el caso de la Figura 4.5 se realiza la misma prueba pero cambiando la dirección de la luz.

Si se usan los mismos parámetros que en las ejecuciones anteriores la esfera resulta con un color mucho más oscuro, se debe compensar por la introducción de la función de fase que va a reducir la contribución por luz directa. Por esta razón se aumentó la intensidad de la luz veinte veces y σ_a pasó de valer 0.6 a 0.05.

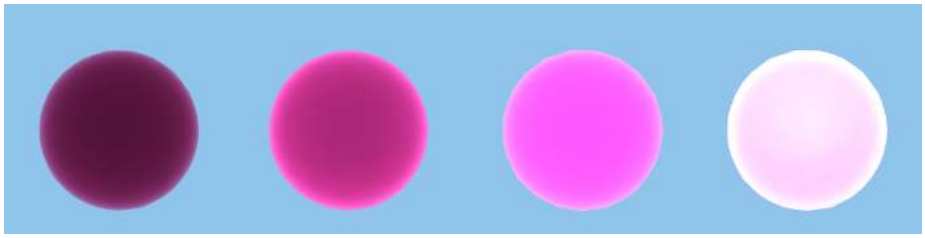

Figura 4.4: Ray marcher mejorado. Luz frontal. De izquierda a derecha la constante g_{HG} de Henyey-Greenstein toma valores 0,25 0 $-$ 0,25 y $-$ 0,5.

g_{HG}	**Tiempo de ejecución (ms)**
0,25	1771
0	1821
−0,25	1826
−0,5	1834

Tabla 4.1: Tiempos de ejecución para Figura 4.4.

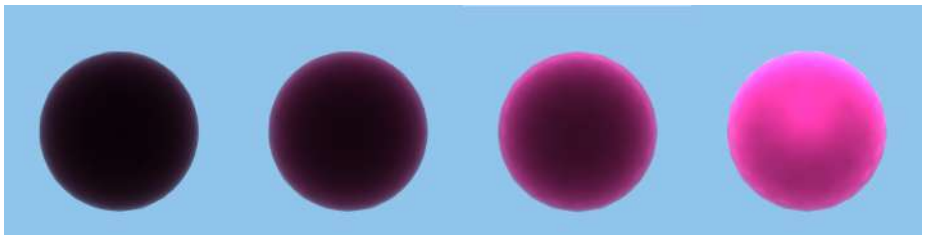

Figura 4.5: Ray marcher mejorado. Luz trasera. De izquierda a derecha la constante g_{HG} de Henyey-Greenstein toma valores 0 0,25 0,5 y 0,75.

g_{HG}	**Tiempo de ejecución (ms)**
0	1762
0,25	1789
0,5	1820
0,75	1811

Tabla 4.2: Tiempos de ejecución para Figura 4.5.

4.2. Medio Heterogéneo

4.2.1. Ray marcher en medio heterogéneo

La Figura 4.6 compara este integrador para diferentes valores de σ_a, dada una luz en dirección fija.

Figura 4.6: Ray marcher en medio heterogéneo. Luz superior, de izquierda a derecha σ_a toma valores 0,2 0,4 0,6 *y* 0,8.

σ_a	**Tiempo de ejecución (ms)**
0,2	6095
0,4	6103
0,6	6114
0,8	6242

Tabla 4.3: Tiempos de ejecución para Figura 4.6.

4.2.2. Ray marcher usando NanoVDB

En la Figura 4.7 se observa una ejecución de este integrador. Los archivos NanoVDB se obtuvieron en el repositorio de Github de escenas para PBRT (*pbrt4-scenes*, 2023).

Figura 4.7: Ray marcher usando NanoVDB. Archivo bunny_cloud.nvdb, σ_a = 0,2 y 16 rayos por píxel. Imagen generada en 132s. Luz direccional superior de color fucsia.

4.2.3. Ray marcher en medio emisivo

En este caso el resultado se puede ver en la Figura 4.8.

Figura 4.8: Ray marcher en medio emisivo. Archivo fire.nvdb, $\sigma_a = 2$ y 256 rayos por píxel. Imagen generada en 302s. Luz direccional superior de color fucsia.

4.2.4. Delta Tracking y Ratio Tracking

Se muestran imágenes de ejemplo de los integradores basados en Delta Tracking y Ratio Tracking desarrollados en este proyecto. Además se muestran imágenes en condiciones similares del *renderer* PBRT usando el integrador *SimpleVolPathIntegrator* (análogo a Delta Tracking de este proyecto) y *VolPathIntegrator* (análogo a Ratio Tracking de este proyecto aunque con features más avanzadas, como por ejemplo MIS).

Se intenta configurar los parámetros de forma que las imágenes tengan tiempo de renderizado similares.

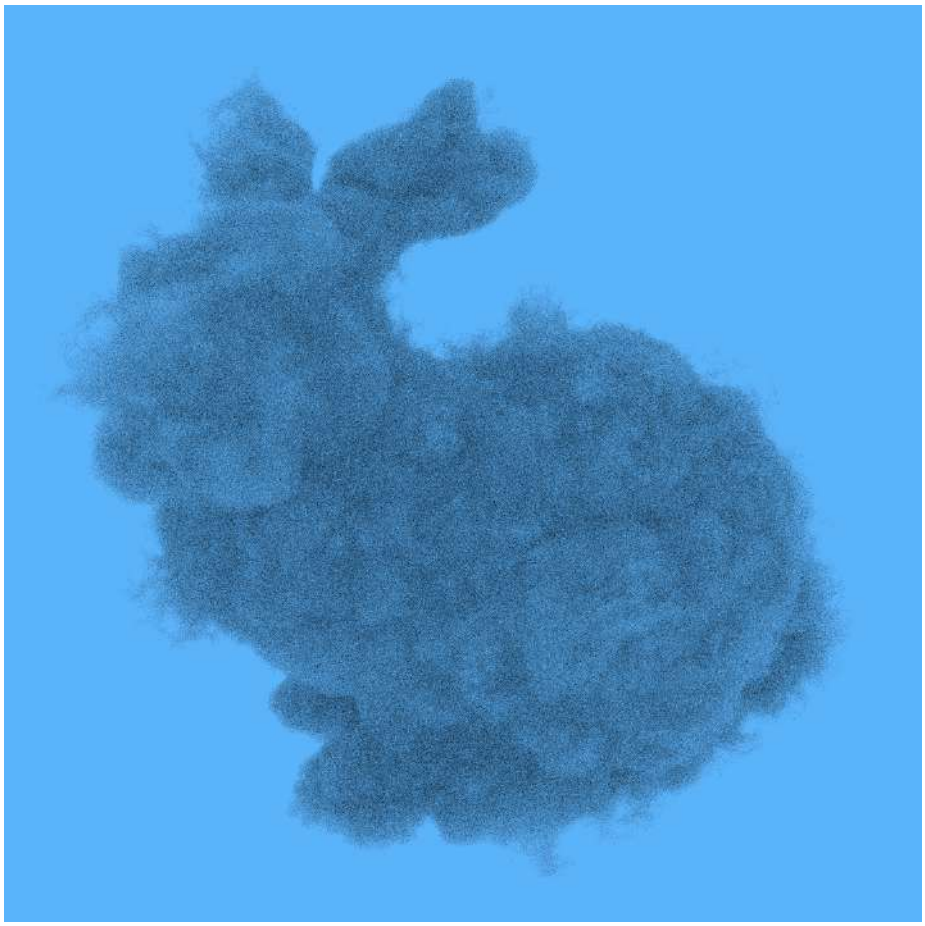

Figura 4.9: Imagen generada con Delta Tracking. Archivo bunny_cloud.nvdb, 16 rayos por píxel. Imagen generada en 62s. Configuración completa en el anexo A.3.1.

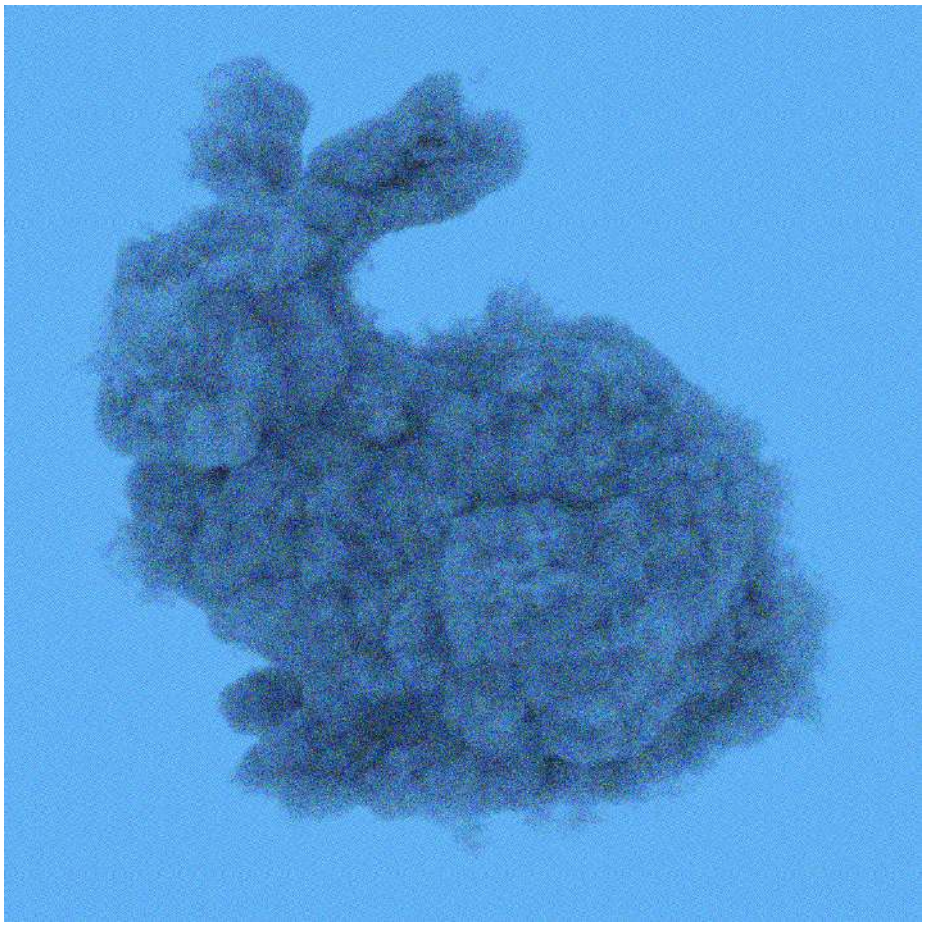

Figura 4.10: Imagen generada con SimpleVolPathIntegrator de PBRT. Archivo bunny_cloud.nvdb. Imagen generada en 91s. Configuración completa en el anexo A.3.3.

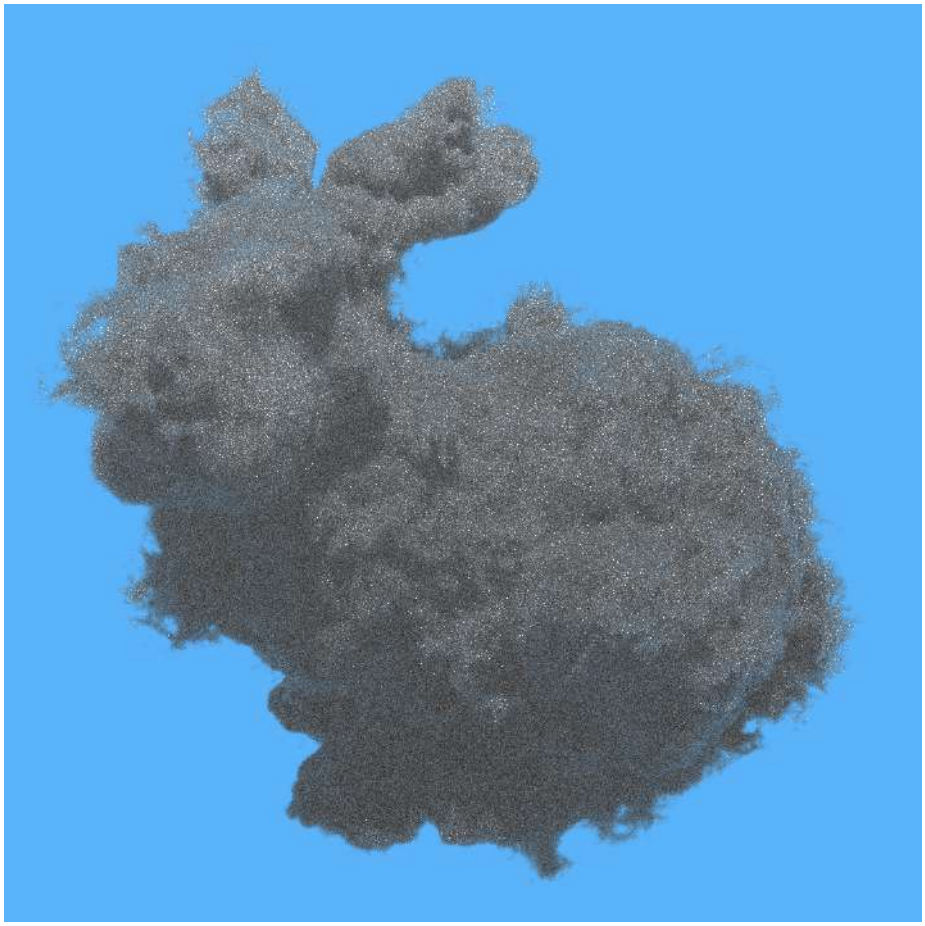

Figura 4.11: Imagen generada con Ratio Tracking con luz superior. Archivo bunny_cloud.nvdb, 16 rayos por píxel. Imagen generada en 2182s. Configuración completa en el anexo A.3.2.

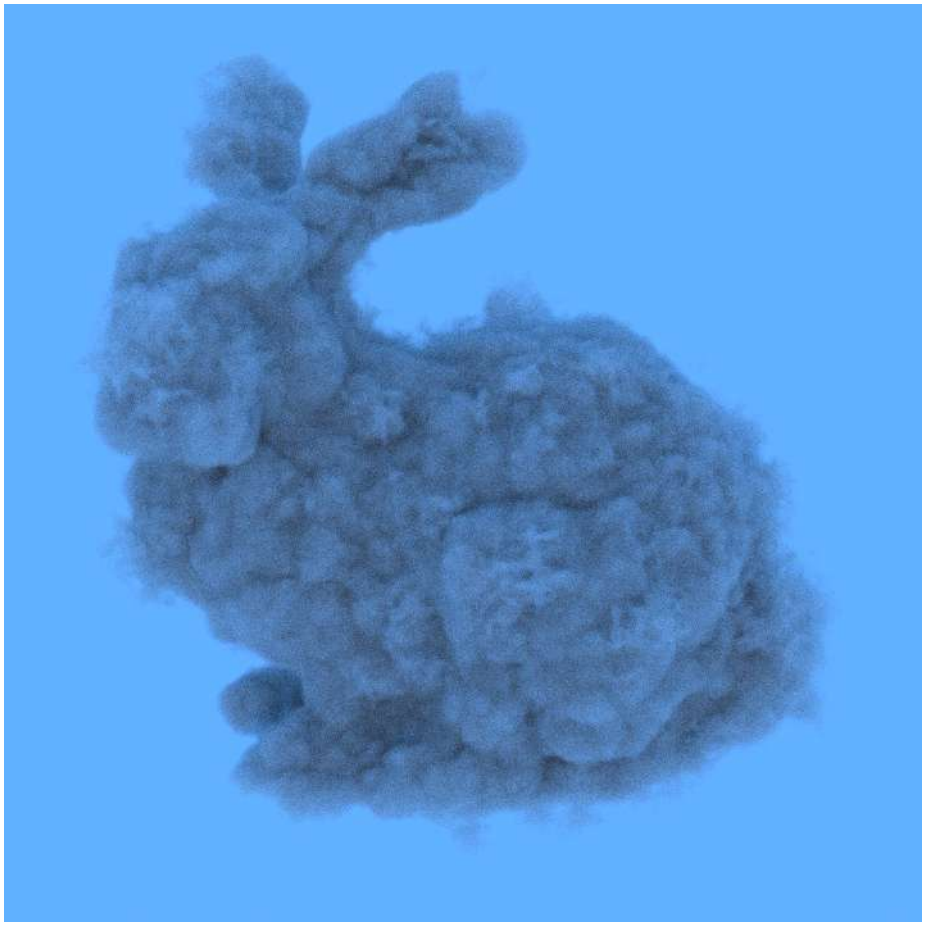

Figura 4.12: Imagen generada con VolPathIntegrator de PBRT, con luz superior. Archivo bunny_cloud.nvdb, 64 rayos por píxel. Imagen generada en 516s. Configuración completa en el anexo A.3.4.

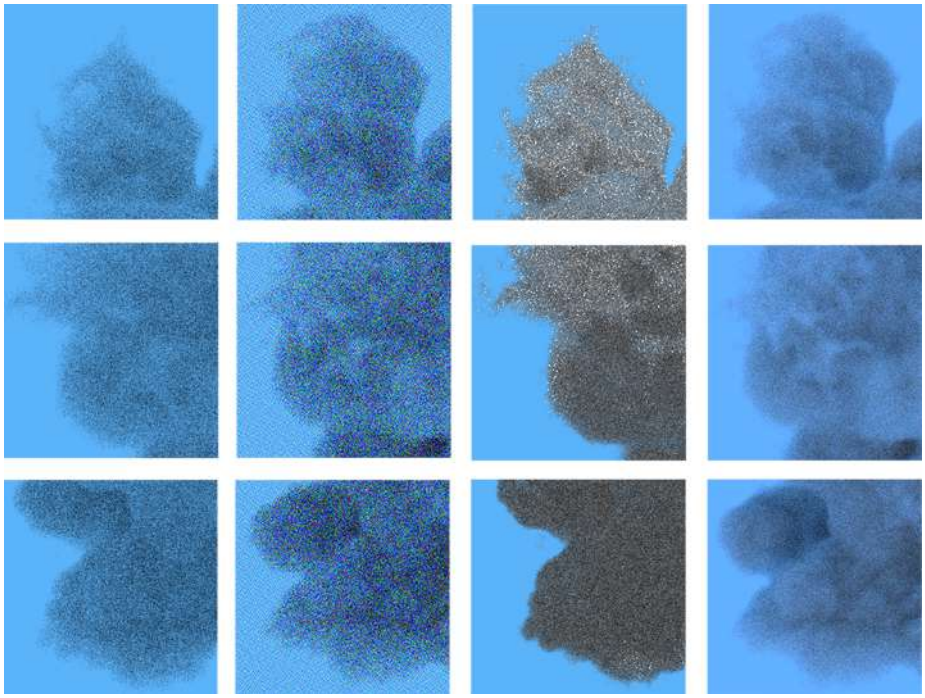

Figura 4.13: Comparación de las diferentes imagenes usando bunny_cloud.nvdb. Las columnas de izquierda a derecha son: Delta Tracking, SimpleVolPathIntegrator de PBRT, Ratio Tracking y VolPathIntegrator de PBRT.

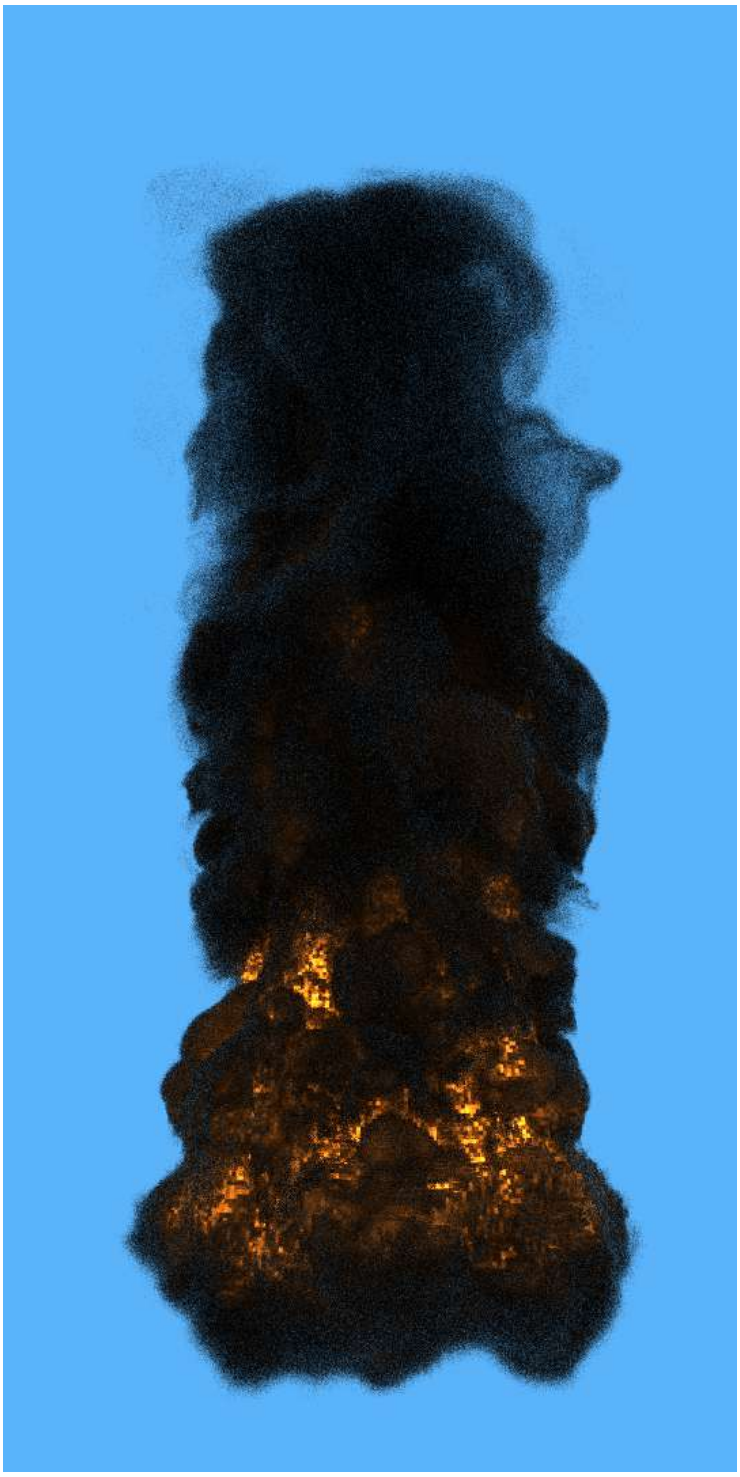

Figura 4.14: Imagen generada con Delta Tracking. Archivo fire.nvdb, 16 rayos por píxel. Imagen generada en 226s. Configuración completa en el anexo A.3.1.

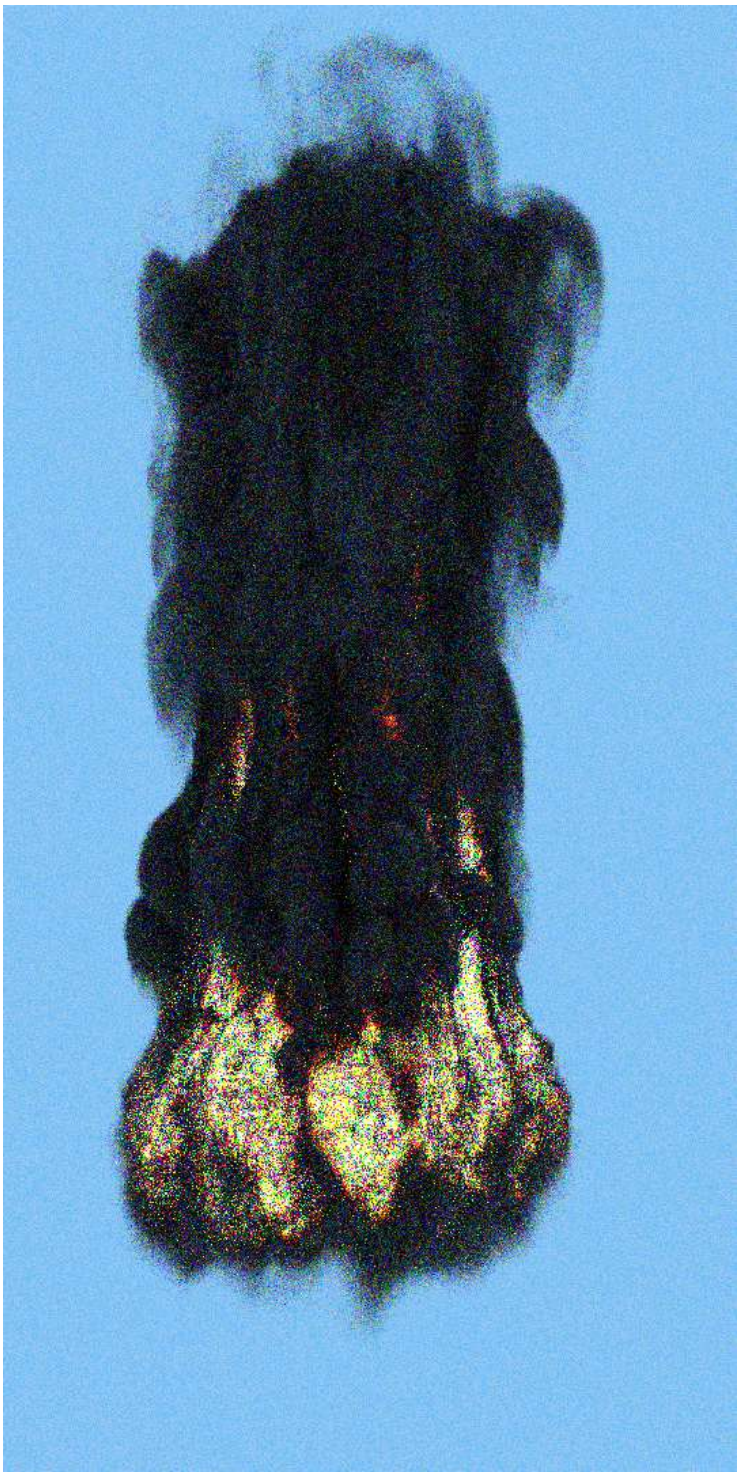

Figura 4.15: Imagen generada con SimpleVolPathIntegrator de PBRT. Archivo fire.nvdb. Imagen generada en 77s. Configuración completa en el anexo A.3.3.

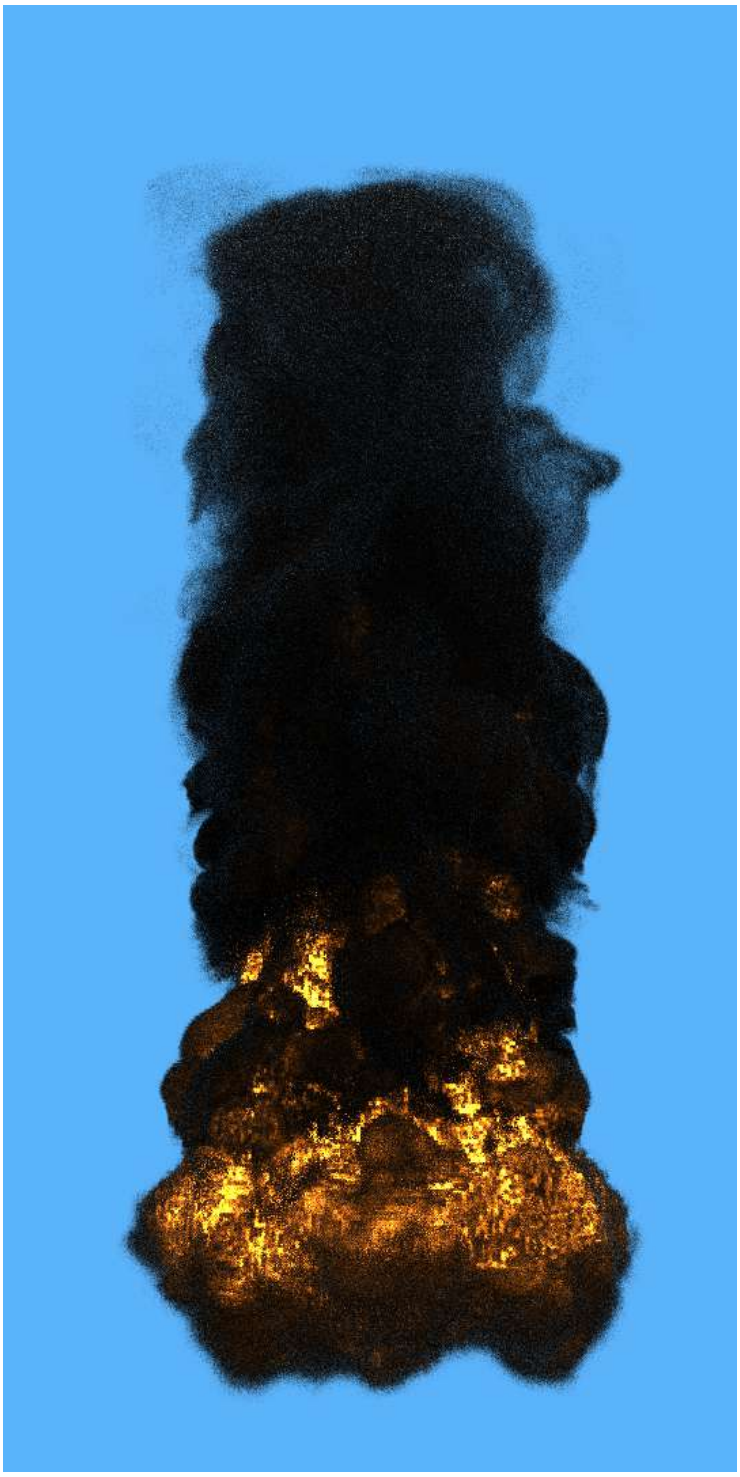

Figura 4.16: Imagen generada con Ratio Tracking con luz superior. Archivo fire.nvdb, 16 rayos por píxel. Imagen generada en 283s. Configuración completa en el anexo A.3.2.

Figura 4.17: Imagen generada con VolPathIntegrator de PBRT, con luz superior. Archivo fire.nvdb, 64 rayos por píxel. Imagen generada en 337s. Configuración completa en el anexo A.3.4.

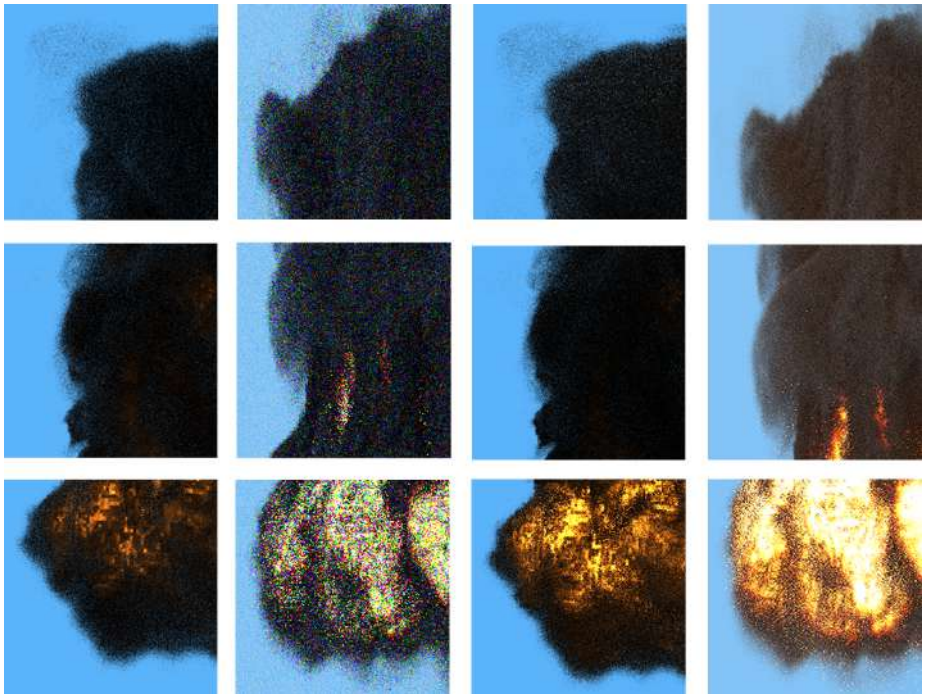

Figura 4.18: Comparación de las diferentes imagenes usando fire.nvdb. Las columnas de izquierda a derecha son: Delta Tracking, SimpleVolPathIntegrator de PBRT, Ratio Tracking y VolPathIntegrator de PBRT.

4.3. Análisis de Resultados

Al no existir una imagen "verdadera" para usar de comparación, resulta difícil hallar una forma objetiva de evaluación de los resultados. Hay trabajos que resuelven el problema inverso de calibrar parámetros para generar la imagen adecuada partiendo de fotografías de nubes (Dobashi y cols., 2012), pero los resultados obtenidos no siempre son convincentes.

La configuración de los *renderers* es clave. Se pueden conseguir imágenes con más o menos ruido ajustando los parámetros. En este proyecto sólo se usó una cantidad pequeña de los parámetros disponibles en PBRT (*pbrt4*, 2023). De todos modos hubo que decidirse por una configuración por cuestión de tiempo.

La imagen con mayor ruido es el conejo generado con Ratio Tracking (ver Figura 4.11), el cual se observa por los píxeles blancos que aparecen. Con este intregador las imágenes tienden a más ser oscuras si se usan los mismos parámetros que en Delta Tracking. Esto ocurre por tener ahora como factor una transmitancia (menor ó igual a 1) y que se multiplica a los valores que se suman radiancia al camino. Para obtener una imagen similar a la obtenidas con Delta Tracking se exageraron los parámetros que aumentan la radiancia, como por ejemplo es el valor de la luz. Estos valores terminan causando los píxeles blancos: si un rayo se dispersa en dirección a la luz rápidamente (sin perder mucha transmitancia) va a obtener un color que es mayor al blanco gráfico (1.0, 1.0, 1.0).

Se intentó implementar Importance Sampling, pero surgieron artefactos visuales y quedó como trabajo futuro. Por último se agregaron multiplicadores a la densidad, tanto para los rayos que salen de la cámara como los rayos de sombra, de forma de poder modificar las propiedades del modelo para hacer diferentes pruebas.

Un código como el desarrollado, debería pasar por una evaluación sistemática, para corregir *bugs* y calibrar parámetros. Pero también, un aspecto importante para toda técnica físicamente realista es la evaluación objetiva del realismo de los resultados. Una rápida revisión de la literatura (Szirmay-Kalos, Kovács, y Abbas, 2001), (Brugger, Freude, y Wimmer, 2020), (Kolivand, Sunar, Kakh, Al-Rousan, y Ismail, 2018) muestra que los métodos más utilizados son los analíticos, donde se comparan los resultados contra soluciones matemáticamente exactas, los basados en encuestas a personas, donde se comparan imágenes reales y simuladas, y aquellos que se basan en comparaciones cuantitativas y cualitativas con resultados provistos por experimentos físicos. Es claro que la evaluación sistemática quedó por fuera del alcance del proyecto. En consecuencia, la evaluación estuvo centrada en la intuición visual de quien suscribe, buscando generar imágenes que logren cierta sensación de realismo.

Capítulo 5

Conclusiones y Trabajo Futuro

Se buscó en este proyecto conocer los fundamentos del renderizado de medios participativos, así como explorar diversas técnicas de simulación. Se implementaron varios integradores, siendo de particular complejidad el Delta Tracking y Ratio Tracking. A su vez se los comparó con integradores similares en un *renderer* robusto como PBRT (*pbrt4*, 2023).

A lo largo del proyecto se decidió reducir su alcance en distintas instancias debido a limitantes del tiempo, en parte por ser un proyecto de un solo estudiante. Originalmente se planteó que el *renderer* soporte superficies a la vez que volúmenes 3D, pero finalmente se optó por concentrar esfuerzos en renderizar los medios participativos. Se investigaron métodos para reducir la varianza de las imágenes, como Importance Sampling y Multiple Importance Sampling (MIS), pero quedaron fuera del alcance del proyecto por sus complejidades conceptuales y de implementación.

Una mejora posible del *renderer* es la de incluir más tipos de luces, así como agregar soporte para múltiples luces en la escena. Otra mejora consiste en renderizar superficies geométricas junto con grillas de NanoVDB al mismo tiempo. El agregado de MIS espectral, muestreando la luz en distintas longitudes de onda es otra extensión posible. Para mejorar la performance, seria importante posibilitar su ejecución en GPU. Como dato a destacar, NanoVDB ya puede ejecutarse en GPU.

Varios integradores se pudieron implementar y no fueron desarrollados en este proyecto, como Weighted Delta Tracking, Decomposition Tracking y Bidirectional Path Tracing. Todos ellos están mencionados en (Novák, Georgiev, Hanika, y Jarosz, 2018). También Differential Ratio Tracking (Nimier-David, Müller, Keller, y Jakob, 2022) pudo haberse desarrollado, por mencionar algunos.

Una mejor base teórica hubiera beneficiado a la implementación, que fue

desarrollada en paralelo al estudio de libros y artículos. Fue un desafío priorizar tareas y decidir cuales abandonar: ocasionalmente me vi en la situación de pasar varios días intentando implementar sin éxito alguna técnica en particular, hasta que decido cambiar mi foco. Como punto positivo tuve un buen punto de partida al contar con proyectos previos realizados durante mis estudios en Computación Gráfica (*Computación Gráfica Avanzada*, 2023).

Referencias

Bertini, H. W. (1963, 5). Monte carlo calculations on intranuclear cascades (thesis).
doi: 10.2172/4692927

Brugger, E., Freude, C., y Wimmer, M. (2020). *Test scene design for physically based rendering.*

Butcher, J. C., y Messel, H. (1958, Dec). Electron number distribution in electron-photon showers. *Phys. Rev.*, *112*, 2096–2106. doi: 10.1103/PhysRev.112.2096

Chandrasekhar, S. (1960). *Radiative transfer.*

Computación gráfica avanzada. (2023). https://eva.fing.edu.uy/course/view.php?id=1067. FING UDELAR.

Dobashi, Y., Iwasaki, W., Ono, A., Yamamoto, T., Yue, Y., y Nishita, T. (2012, nov). An inverse problem approach for automatically adjusting the parameters for rendering clouds using photographs. *ACM Trans. Graph.*, *31*(6). doi: 10.1145/2366145.2366164

eluna-renderer. (2023). https://github.com/emiliano-luna/volume-renderer. GitHub.

Fong, J., Wrenninge, M., Kulla, C., y Habel, R. (2017). Production volume rendering: Siggraph 2017 course. En *Acm siggraph 2017 courses.* New York, NY, USA: Association for Computing Machinery. Descargado de https://doi.org/10.1145/3084873.3084907 doi: 10.1145/3084873.3084907

Freeimage. (2023). https://freeimage.sourceforge.io/. FreeImage.

Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., ... Paragios, N. (2007). Hardware-accelerated volume rendering for real-time medical data visualization. En *Advances in visual computing (978-3-540-76855-5)* (p. 801 - 810).

Jensen, J. L. W. V. (1906). Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, *30*(none), 175 – 193. doi: 10.1007/BF02418571

Kolivand, H., Sunar, M. S., Kakh, S. Y., Al-Rousan, R., y Ismail, I. (2018, 01 de Oct). Photorealistic rendering: a survey on evaluation. *Multimedia Tools and Applications*, *77*(19), 25983-26008. Descargado de https://doi.org/10.1007/s11042-018-5834-7 doi: 10.1007/s11042-018-5834-7

Kulla, C., y Fajardo, M. (2012). Importance sampling techniques for path

tracing in participating media. *Computer Graphics Forum*, *31*(4), 1519-1528. doi: 10.1111/j.1467-8659.2012.03148.x

Matt Pharr, W. J., y Humphreys, G. (2023). *Physically based rendering, fourth edition*. The MIT Press.

Miller, B., Georgiev, I., y Jarosz, W. (2019, julio). A null-scattering path integral formulation of light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, *38*(4). doi: 10/gf6rzb

Morgan, L., y Kotlyar, D. (2015). Weighted-delta-tracking for monte carlo particle transport. *Annals of Nuclear Energy*, *85*, 1184-1188. doi: 10.1016/j.anucene.2015.07.038

Museth, K. (2021). Nanovdb: A gpu-friendly and portable vdb data structure for real-time rendering and simulation. En *Acm siggraph 2021 talks*. New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3450623.3464653

Museth, K., Avramoussis, N., y Bailey, D. (2019). Openvdb. En *Acm siggraph 2019 courses*. New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3305366.3328070

Nanovdb. (2023). https://github.com/AcademySoftwareFoundation/openvdb/tree/master/nanovdb/nanovdb. Academy Software Foundation.

Nimier-David, M., Müller, T., Keller, A., y Jakob, W. (2022, julio). Unbiased inverse volume rendering with differential trackers. *ACM Trans. Graph.*, *41*(4), 44:1–44:20. doi: 10.1145/3528223.3530073

Novák, J., Selle, A., y Jarosz, W. (2014, nov). Residual ratio tracking for estimating attenuation in participating media. *ACM Trans. Graph.*, *33*(6). Descargado de https://doi.org/10.1145/2661229.2661292 doi: 10.1145/2661229.2661292

Novák, J., Georgiev, I., Hanika, J., y Jarosz, W. (2018, mayo). Monte Carlo methods for volumetric light transport simulation. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)*, *37*(2). doi: 10/gd2jqq

Novák, J., Georgiev, I., Hanika, J., Křivánek, J., y Jarosz, W. (2018, agosto). Monte Carlo methods for physically based volume rendering. En *Acm siggraph courses*. doi: 10/c5fj

Openvdb. (2012). https://www.openvdb.org/. Academy Software Foundation.

pbrt4. (2023). https://github.com/mmp/pbrt-v4. GitHub.

pbrt4-scenes. (2023). https://github.com/mmp/pbrt-v4-scenes. GitHub.

Prakash, E., Brindle, G., Jones, K., Zhou, S., Chaudhari, N., y Wong, K. (2009, 12). Advances in games technology: Software, models, and intelligence. *Simulation Gaming*, *40*. doi: 10.1177/1046878109335120

pugixml. (2023). https://pugixml.org/. pugixml.

Skullerud, H. R. (1968, nov). The stochastic computer simulation of ion motion in a gas subjected to a constant electric field. *Journal of Physics D: Applied Physics*, *1*(11), 1567. doi: 10.1088/0022-3727/1/11/423

Szirmay-Kalos, L., Kovács, L., y Abbas, A. (2001, 02). Testing monte-carlo glo-

bal illumination methods with analytically computable scenes. En (p. 419-).

Tatarchuk, N. (2019). Advances in real-time rendering in games, part 2. En *Acm siggraph 2019 courses.* New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3305366.3335036

tinyobjloader. (2023). https://github.com/tinyobjloader/tinyobjloader. tinyobjloader.

Volume rendering for developers: Foundations. (2023). https://www.scratchapixel.com/lessons/3d-basic-rendering/volume-rendering-for-developers/intro-volume-rendering.html. Scratchapixel.

Wald, I., Woop, S., Benthin, C., Johnson, G. S., y Ernst, M. (2014, jul). Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.*, *33*(4). doi: 10.1145/2601097.2601199

Wang, N., Paljic, A., y Fuchs, P. (2010). A study of perception of volumetric rendering for immersive scientific visualization. En *Proceedings of the 20th international conference on artificial reality and telexistence (icat'10)* (pp. 145–152).

Weinzierl, S. (2000). *Introduction to monte carlo methods.*

Wrenninge, M. (2012). *Production volume rendering: Design and implementation.* doi: 10.1201/b12698

Wrenninge, M., y Bin Zafar, N. (2011). Production volume rendering fundamentals: Siggraph 2011 course. En *Acm siggraph 2011 courses.*

Anexo A

Anexo 1

A.1. Manual de Instalación y Uso

Para probar la aplicación se puede clonar usando Git el repositorio (*eluna-renderer*, 2023) desde una PC con Windows.

Una vez descargada se abre la solución de Visual Studio ubicada en \ *VolumeRenderer*\ *VolumeRenderer.sln* y ya se puede ejecutar, el archivo de configuración se encuentra en \ *VolumeRenderer*\ *SceneRenderer*\ *configRenderer.xml*. En la misma carpeta van a quedar las imágenes resultantes.

También se puede ejecutar por consola mediante el ejecutable \ *VolumeRenderer*\ *x64*\ *Release*\ *SceneRenderer.exe*, el archivo XML para dicho ejecutable se encuentra en la misma carpeta.

Las grillas de NanoVDB usadas ya se encuentran en el repositorio y fueron descargadas de (*pbrt4-scenes*, 2023).

A.2. Configuración

En la tabla a continuación se comentan los distintos valores que se pueden usar para configurar el *renderer*.

En la próxima sección se ven configuraciones de ejemplo.

Nombre	**Descripción**
models	Contiene un model
model	Modelo OBJ a cargar
sigma_a	Se multiplica por la densidad en un punto $\mathbf{x}$ para obtener $\mu_a(\mathbf{x})$
sigma_s	Se multiplica por la densidad en un punto $\mathbf{x}$ para obtener $\mu_s(\mathbf{x})$
heyneygreenstein_g	Parámetro g de Henyey-Greenstein
rayPerPixelCount	Cantidad de rayos por píxel, tiene un alto impacto en la performance.
stepSize	Define un mínimo y máximo para el largo de salto en Delta Tracking y Ratio Tracking
maxDepth	Máxima cantidad de eventos de dispersión que pueden ocurrir
integrator	Integrador a ejecutar, los posibles valores son: *densitySampling, homogeneousRayMarcherNEE, homogeneousRayMarcherImproved, heterogeneousPerlinNoise, nanoVDBSimple, nanoVDBEmission, deltaTracking, ratioTracking*
width	Ancho de la imagen en píxeles
height	Alto de la imagen en píxeles
multiThreaded	Define si ejecutar con paralelismo o no
multiThreadedChunkSize	Altura en píxeles del bloque horizontal usado en el paralelismo, debe ser un divisor de height
camera	Define parámetros de la cámara
backgroundColor	Color de fondo en la escena
light	Definición de la luz en la escena
emission	Color de emisión
medium	Color que toma un rayo al ser absorbido, se usa en DeltaTracking y RatioTracking
lightRayDensityMultiplier	Multiplicador que se aplica a la densidad muestreada en los rayos de luz
shadowRayDensityMultiplier	Multiplicador que se aplica a la densidad muestreada en los rayos de sombra

A.3. Configuraciones de ejemplo

Las siguientes configuraciones se pueden usar para reproducir algunas imágenes vistas en el informe.

Las imágenes producidas por pbrt se flippearon verticalmente y se acortaron horizontalmente para tener igual orientación y ancho que las demás imágenes generadas.

A.3.1. Delta Tracking

Conejo, sin luz

```
<?xml version="1.0"?>
<options>
  <models>
  </models>
  <densityField baseDir="..\DensityFields\">bunny_cloud.nvdb</densityField>
  <sigma_a>0.05</sigma_a>
  <sigma_s>0.95</sigma_s>
  <heyneygreenstein_g>0.2</heyneygreenstein_g>
  <rayPerPixelCount>16</rayPerPixelCount>
  <stepSize min="0.01" max="0.5"/>
  <maxDepth>16</maxDepth>
  <integrator>deltaTracking</integrator>
  <width startOffset="540" reference="2160">1080</width>
  <height startOffset="540" reference="2160">1080</height>
  <multiThreaded>true</multiThreaded>
  <multiThreadedChunkSize>20</multiThreadedChunkSize>
  <camera x="-10.0" y="33.5" z="46.0" pitch="-19.0" yaw="-8.0" roll="0.0" fov="100"/>
  <!-- trying to match pbrt -->
  <backgroundColor r="0.36" g="0.702" b="0.98"/>
  <emission r="0.25" g="0.125" b="0.025" multiplier="0.3"/>
</options>
```

Explosión, sin luz

```
<?xml version="1.0"?>
<options>
  <models>
  </models>
  <densityField baseDir="..\DensityFields\">fire.nvdb</densityField>
  <sigma_a>2.0</sigma_a>
  <sigma_s>4.0</sigma_s>
  <heyneygreenstein_g>0.8</heyneygreenstein_g>
  <rayPerPixelCount>16</rayPerPixelCount>
  <stepSize min="0.0001" max="0.01"/>
```

```
  <maxDepth>16</maxDepth>
  <integrator>deltaTracking</integrator>
  <width startOffset="0" reference="540">540</width>
  <height startOffset="0" reference="1080">1080</height>
  <multiThreaded>true</multiThreaded>
  <importanceSampling>false</importanceSampling>
  <multiThreadedChunkSize>20</multiThreadedChunkSize>
  <!-- camera for fire.nvdb -->
  <camera x="-1.0" y="10" z="75.0" pitch="20" yaw="0" roll="0" fov="75"/>
  <!-- trying to match pbrt -->
  <backgroundColor r="0.36" g="0.702" b="0.98"/>
  <light x="0" y="1" z="0" r="19.65" g="4.5" b="13.5" multiplier="0.0"/>
  <emission r="0.25" g="0.125" b="0.025" multiplier="5.0"/>
</options>
```

A.3.2. Ratio Tracking

Conejo, con luz

```
<?xml version="1.0"?>
<options>
  <models>
  </models>
  <densityField baseDir="..\DensityFields\">bunny_cloud.nvdb</densityField>
  <sigma_a>0.1</sigma_a>
  <sigma_s>0.9</sigma_s>
  <heyneygreenstein_g>0.8</heyneygreenstein_g>
  <rayPerPixelCount>1</rayPerPixelCount>
  <stepSize min="0.001" max="0.1"/>
  <maxDepth>16</maxDepth>
  <integrator>ratioTracking</integrator>
  <width startOffset="540" reference="2160">1080</width>
  <height startOffset="540" reference="2160">1080</height>
  <multiThreaded>true</multiThreaded>
  <importanceSampling>false</importanceSampling>
  <multiThreadedChunkSize>20</multiThreadedChunkSize>
  <camera x="-10.0" y="33.5" z="46.0" pitch="-19.0" yaw="-8.0" roll="0.0" fov="100"/>
  <!-- trying to match pbrt -->
  <backgroundColor r="0.36" g="0.702" b="0.98"/>
  <light x="0" y="1" z="0" r="1.0" g="1.0" b="1.0" multiplier="80.0"/>
  <emission r="0.25" g="0.125" b="0.025" multiplier="16.0"/>
  <medium r="1.0" g="1.0" b="1.0" multiplier="1.0"/>
</options>
```

Explosión, con luz

```
<?xml version="1.0"?>
```

```
<options>
  <models>
  </models>
  <densityField baseDir="..\DensityFields\">fire.nvdb</densityField>
  <sigma_a>2.0</sigma_a>
  <sigma_s>2.0</sigma_s>
  <heyneygreenstein_g>0.8</heyneygreenstein_g>
  <rayPerPixelCount>16</rayPerPixelCount>
  <stepSize min="0.0001" max="0.01"/>
  <maxDepth>16</maxDepth>
  <integrator>ratioTracking</integrator>
  <width startOffset="0" reference="540">540</width>
  <height startOffset="0" reference="1080">1080</height>
  <multiThreaded>true</multiThreaded>
  <importanceSampling>false</importanceSampling>
  <multiThreadedChunkSize>20</multiThreadedChunkSize>
  <camera x="-1.0" y="10" z="75.0" pitch="20" yaw="0" roll="0" fov="75"/>
  <!-- trying to match pbrt -->
  <backgroundColor r="0.36" g="0.702" b="0.98"/>
  <light x="0" y="1" z="0" r="1.0" g="1.0" b="1.0" multiplier="80.0"/>
  <emission r="0.25" g="0.125" b="0.025" multiplier="16.0"/>
</options>
```

A.3.3. Simple Volume Path Integrator (PBRT)

Conejo, sin luz

```
Sampler "halton"
LookAt 00 120 50       7 0 17   0 0 1

Camera "perspective"
    "float fov" 25

Film "rgb"
  "integer xresolution" 1920
  "integer yresolution" 1080
  "string sensor" "nikon_d850"
  "float whitebalance" 4800
  "float iso" 90
  "string filename" "bunny-cloud.exr"

Integrator "simplevolpath" "integer maxdepth" 50

WorldBegin

# uniform blue-ish illumination from all directions
```

```
LightSource "infinite" "rgb L" [ .572 .772 .921 ]

AttributeBegin
Rotate 180 0 0 1
Rotate 90 1 0 0
MakeNamedMedium "foo" "string type" "nanovdb"
   "string filename" "bunny_cloud.nvdb"
   "spectrum sigma_s" [200 10 900 10] "spectrum sigma_a" [200 .5 900 .5]
AttributeEnd

AttributeBegin
MediumInterface "foo" ""
Material "interface"
Shape "sphere" "float radius" 45
AttributeEnd
```

Explosión, sin luz

```
LookAt 00 120 20       -.5 0 30   0 0 1
#LookAt 00 400 0       -20 0 150   0 0 1

Camera "perspective"
    "float fov" 37

Film "rgb"
  "integer xresolution" 1300
  "integer yresolution" 1800
  "string sensor" "nikon_d850"
  "float whitebalance" 6000
  "float iso" 100
  "string filename" "explosion.exr"

Integrator "simplevolpath" "integer maxdepth" 5

WorldBegin

# uniform blue-ish illumination from all directions
LightSource "infinite" "rgb L" [ .572 .772 .921 ]

AttributeBegin
Scale 1 1 1.6
Rotate 90 1 0 0
MakeNamedMedium "kaboom" "string type" "nanovdb"
   "string filename" "fire.nvdb"
   "spectrum sigma_s" [200 10 900 10] "spectrum sigma_a" [200 10 900 10]
```

```
   "float Lescale" 5  "float temperaturecutoff" 1 "float temperaturescale" 100
AttributeEnd

AttributeBegin
MediumInterface "kaboom" ""
Material "interface"
Translate 0 40 0
Shape "sphere" "float radius" 80
AttributeEnd
```

A.3.4. Volume Path Integrator (PBRT)

Conejo, con luz

```
Sampler "halton"
LookAt 00 120 50       7 0 17   0 0 1

Camera "perspective"
    "float fov" 25

Film "rgb"
  "integer xresolution" 1920
  "integer yresolution" 1080
  "string sensor" "nikon_d850"
  "float whitebalance" 5000
  "float iso" 90
  "string filename" "bunny-cloud.exr"

Sampler "halton" "integer pixelsamples" 64
Integrator "volpath" "integer maxdepth" 50

WorldBegin

# uniform blue-ish illumination from all directions
LightSource "infinite" "rgb L" [ .572 .772 .921 ] "float scale" 1.0

# approximate the sun
LightSource "distant"
  "point3 from" [ 0 1 0 ]
  "point3 to" [ 0 0 0 ]
  "blackbody L" [ 3000 ] "float scale" 0.35

AttributeBegin
Rotate 180 0 0 1
Rotate 90 1 0 0
```

```
MakeNamedMedium "foo" "string type" "nanovdb"
   "string filename" "bunny_cloud.nvdb"
   "spectrum sigma_s" [200 10 900 10] "spectrum sigma_a" [200 .5 900 .5]
AttributeEnd

AttributeBegin
MediumInterface "foo" ""
Material "interface"
Shape "sphere" "float radius" 45
AttributeEnd
```

Explosión, con luz

```
LookAt 00 120 20       -.5 0 30   0 0 1
#LookAt 00 400 0       -20 0 150   0 0 1

Camera "perspective"
    "float fov" 37

Film "rgb"
  "integer xresolution" 1300
  "integer yresolution" 1800
  "string sensor" "nikon_d850"
  "float whitebalance" 6000
  "float iso" 100
  "string filename" "explosion.exr"

Sampler "halton" "integer pixelsamples" 64
Integrator "volpath" "integer maxdepth" 50

WorldBegin

# uniform blue-ish illumination from all directions
LightSource "infinite" "rgb L" [ .572 .772 .921 ]

# approximate the sun
LightSource "distant"
  "point3 from" [ 0 1 0 ]
  "point3 to" [ 0 0 0 ]
  "blackbody L" [ 3000 ] "float scale" 3.0

AttributeBegin
Scale 1 1 1.6
Rotate 90 1 0 0
```

```
MakeNamedMedium "kaboom" "string type" "nanovdb"
   "string filename" "fire.nvdb"
   "spectrum sigma_s" [200 10 900 10] "spectrum sigma_a" [200 10 900 10]
   "float Lescale" 5  "float temperaturecutoff" 1 "float temperaturescale" 100
AttributeEnd

AttributeBegin
MediumInterface "kaboom" ""
Material "interface"
Translate 0 40 0
Shape "sphere" "float radius" 80
AttributeEnd
```