CS171 Final Report

GROUP NUMBER: 1
MEMBER 1: YIHENG WU
MEMBER 2: HAIZHAO DAI
MEMBER 3: XIAOHAN WU

1 INTRODUCTION

In this project, we choose to replicate the paper *Spectral and Decomposition Tracking for Rendering Heterogeneous Volumes*. Volume rendering technique is used in rendering colloid, particals, translucent, and transparent objects. For the heterogeneous medium, two unbiased methods based on Monte Carlo methods, delta tracking and weight tracking, are often used. However, these methods are not efficient in sampling the distance. To solve the problem, the authors of the paper proposed decomposition tracking that splits the medium into control component and residual component, which accelerates sampling distance.

2 BACKGROUND

Before the spectral and decomposition tracking, we need to introduce some often-used methods that are the bases of these two methods.

2.1 Basic Concept

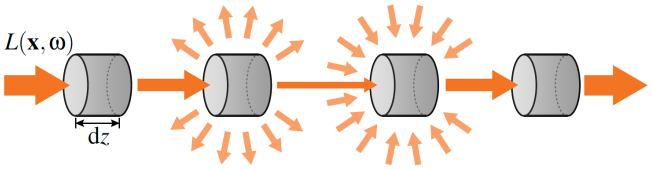
Considering a participating medium that consists of a collection of particals. When a ray $\mathbf{r}(t) = \mathbf{o} - t\mathbf{d}$ travels through the medium, it will get absorbed or scattered. We define μ_a as the *absorption coefficient* which represents the proportion of radiance that is absorbed by the medium undergoing the respective interaction per unit distance traveled. Also, μ_s for the *scattering coefficient*. The *extinction coefficient* $\mu_t = \mu_a + \mu_s$ indicates the proportion of radiance that loses per unit distance it travels. The *ratio* $\alpha = \mu_s/\mu_t$ is the proportion of losed radiance that is scattered. Generally, these properties are all spatial-varying.

Since there is radiance get out-scattered to other directions, there also is radiance get in-scattered to the direction that ray traveling. The ray loses radiance in the former event while get compensated from the latter event. The probability function of the direction that light scattered at a point $\mathbf{x}$ in the medium is the *phase function* f_p. Generally, it is a function related to the position and the incoming and the outgoing directions, that is:

$$f_p : (\mathbf{x}, \omega_i, \omega_o) \to \mathbb{R}^+ \tag{1}$$

The phase function is called isotropic if it scatters the incoming radiance uniformly, that is $f_p = 1/(4\pi)$ for all the point in the medium and for all light directions.

Apart from absorption and scattering, volumetric medium can also emit radiance. The ray traveling through the medium gains radiance from emission. The four kind of effects are show in fig.**??**.



(a) *Absorption* **(b)** *Out-scattering* **(c)** *In-scattering* **(d)** *Emission*

Fig. 1. Absorption, scattering, and emission of the volumetric medium.

If all the properties of the medium are spatial invariant, then the medium is said to be *homogeneous*. Otherwise, the medium is *heterogeneous*.

2.2 Radiative Transfer Equation (RTE)

The radiance transfer equation that describes the change in radiance that travels along direction ω through a differential volume at point $\mathbf{x}$ is:

$$(\omega \cdot \nabla)L(\mathbf{x}, \omega) = -\mu_t(\mathbf{x})L(\mathbf{x}, \omega) + \mu_s(\mathbf{x})L_s(\mathbf{x}, \omega) + \mu_a(\mathbf{x})L_e(\mathbf{x}, \omega) \tag{2}$$

where:

$$L_s(\mathbf{x}, \omega) = \int_{S^2} f_p(\mathbf{x}, \omega, \omega')L_i(\mathbf{x}, \omega')d\omega' \tag{3}$$

is the term that measures the in-scattered radiance from other directions distributed on a unit sphere modified by the phase function.

The first term in the RHS of the eq.(2) measures the loss of the radiance when travel through the volume. The second term measures the gains from incoming radiance. The third term measures the gains from self-emission.

Integrate both side of the eq.(2), we get the integral form of RTE:

$$L(\mathbf{x}, \omega) = \int_0^\infty T(\mathbf{x}, \mathbf{y}) \left[\mu_a(\mathbf{y})L_e(\mathbf{y}, \omega) + \mu_s(\mathbf{y})L_s(\mathbf{y}, \omega)\right] \mathrm{d}y \tag{4}$$

where $L(\mathbf{x}, \omega)$ is the final radiance and $\mathbf{y} = \mathbf{x} - y\omega$ is a point in the medium. The *transmittance* $T(\mathbf{x}, \mathbf{y})$ measures the radiance that is preserved after traveling between two points:

$$T(\mathbf{x}, \mathbf{y}) = \exp\left(-\int_0^y \mu_t(\mathbf{x} - s\omega)\mathrm{d}s\right) \tag{5}$$

Here we define the *optical thickness* as $\tau(t) = \int_0^t \mu_t(\mathbf{x} - s\omega)\mathrm{d}s$.

For a surface point $\mathbf{z} = \mathbf{x} - z\omega$, consider the radiance from the surface, we finally get the volume rendering equation (VRE) (fig.**??**):

$$\begin{aligned} L(\mathbf{x}, \omega) = &\int_0^z T(\mathbf{x}, \mathbf{y}) \left[\mu_a(\mathbf{y})L_e(\mathbf{y}, \omega) + \mu_s(\mathbf{y})L_s(\mathbf{y}, \omega)\right] \mathrm{d}y \\ &+ T(\mathbf{x}, \mathbf{z})L(\mathbf{z}, \omega) \end{aligned} \tag{6}$$

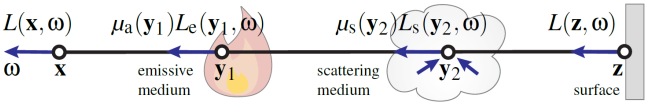


Fig. 2. Volume rendering equation

where $L(\mathbf{z},\omega)$ is the radiance from the surface rendering equation:

$$L(\mathbf{z},\omega) = L_e(\mathbf{z},\omega) + \int_{\mathcal{S}^2} f_r(\mathbf{z},\omega,\omega')L_i(\mathbf{z},\omega')(\mathbf{n}(\mathbf{x})\cdot\omega')\mathrm{d}\omega' \quad (7)$$

Here, $L_e(\mathbf{z},\omega)$ is the direct lighting from the surface point $\mathbf{z}$ along the direction ω and $f_r(\mathbf{z},\omega,\omega')$ is the bidirectional scattering distribution function (BSDF). We will discuss the VRE latter as it is the fundamental rendering equation.

2.3 Basic Tracking Methods

For simplicity, we only consider evaluating the radiative transfer equation eq.(4). When evaluating the integral-formed RTE, people usually perfer to use Monte Carlo integration. The Monte Carlo estimator of the equation is:

$$\langle L(\mathbf{x},\omega)\rangle = \frac{T(\mathbf{x},\mathbf{y})}{P(\mathbf{y})}\left[\mu_a(\mathbf{y})L_e(\mathbf{y},\omega) + \mu_s(\mathbf{y})L_s(\mathbf{y},\omega)\right] \quad (8)$$

The tracking methods, also referred to as random walk or free-path sampling, require sampling the collision point in the medium and the connect them into a light transfer path. Given a starting point $\mathbf{x}_j$ ($\mathbf{x}$ in estimator) and a traveling direction ω_j (ω in estimator) on the path segment j, we need to sample the next collision point $\mathbf{x}_{j+1}$ ($\mathbf{y}$ in estimator) and the next sample direction ω_{j+1}. Sampling direction is easy that we just need to query the phase fuction. Sampling distance is much harder and in many cases, we need to sample the volumetric medium adaptively. Here are the frequently-used methods.

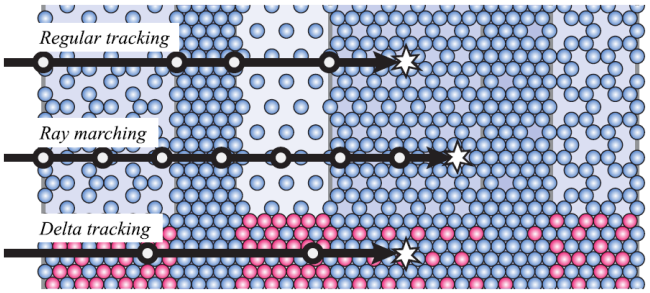


Fig. 3. Three methods for sampling distance in the medium. Regular tracking requires the boundary; ray marching ignores the boundary, constantly or linearly fitting the extinction coefficient; delta tracking adds fictitious medium to the volumetric medium.

2.3.1 Analytical Sampling. Since the transmittance $T(\mathbf{x},\mathbf{y})$ measures the proportion of the preserved radiance that traveling from $\mathbf{x}$ to $\mathbf{y}$ where $\mathbf{y} = \mathbf{x} - y\omega$, then we can also consider the transmittance as the probability of a ray that does not interact with the volumetric medium when traveling from $\mathbf{x}$ to $\mathbf{y}$. Usually we can denote $T(\mathbf{x},\mathbf{y})$ as $T_{\mathbf{x}}(y)$ which means that the transmittance from the point $\mathbf{x}$ in the distance y.

Let X be a random variable that the ray passes through a distance y, then we can get the cumulative distribution function (CDF) of X:

$$\begin{aligned} P(X > y) &= T(\mathbf{x},\mathbf{y}) \\ P(X < y) &= 1 - T(\mathbf{x},\mathbf{y}) = F_X(y) \end{aligned} \quad (9)$$

By differentiating the CDF, we obtain the probability distribution function (PDF) of X:

$$\begin{aligned} P_X(y) &= \frac{dF_X(y)}{dy} = \exp\left(-\int_0^y \mu_t(\mathbf{x}-s\omega)\mathrm{d}s\right)\mu_t(\mathbf{x}-y\omega) \\ &= T(\mathbf{x},\mathbf{y})\mu_t(\mathbf{y}) \end{aligned} \quad (10)$$

If the extinction coefficient μ_t is constant, or changing polynomially or exponentially in space, then the distance can be sampled analytically using inverse sampling. Taking constanct extinction coefficient as example. The free-path distance t can be sampled analytically: $t = -\ln(1-\xi)/\mu_t$ where ξ is a sample of the uniform distribution on interval $[0,1]$. If the inverse of the CDF has no analytically representation, we can apply Newton's method to sample the distance.

2.3.2 Regular Tracking. If the medium consists several homogeneous medium with different extinction coefficient, by finding the boundary of the volumetric medium, we can obtain the transmittance and distance sample easily. Therefore, quickly find the boundary becomes the bottleneck of the performance of regular tracking.

2.3.3 Ray Marching. By ignoring the boundary of the volumetric medium, we can sample the volumetric medium within a fix-sized stride. This method is efficient for not requiring the boundary. However, due to the Jensen's inequality, this method is biased. Though it can be reduced by smaller steps or random jittering, it also increases the cost.

2.4 Delta Tracking

Delta tracking is the most famous method used in sampling heterogeneous volumetric medium. This method is based on reject sampling by introducing the fictitious medium into the real medium. Therefore we can combine the analytical sampling into this method, and thus making it unbiased.

The fictitious medium in the delta tracking homogenizes the real medium to a constant extinction coefficient medium. We name this coefficient as free-path-sampling coefficient and denote it as $\bar{\mu}$. It will not affect the energy of the radiance nor the traveling direction. Usually people set this bound to the upper bound of the real medium extinction coefficient, that is $\bar{\mu} = \sup_{\mathbf{x}} \mu_t(\mathbf{x})$. The density of the fictitious that fills the gap between the real and the upper bound is denoted as $\mu_n(\mathbf{x})$. By the law of the conservation of energy, we get:

$$-\mu_n(\mathbf{x})L(\mathbf{x},\omega) + \int_{\mathcal{S}^2}\delta(\omega-\omega')L(\mathbf{x},\omega')d\omega' = 0 \quad (11)$$

Plugging it into the eq.(2), and integraling it, we get:

$$L(\mathbf{x},\omega) = \int_0^\infty T(\mathbf{x},\mathbf{y}) \\ [\mu_a(\mathbf{y})L_e(\mathbf{y},\omega) + \mu_s(\mathbf{y})L_s(\mathbf{y},\omega) + \mu_n(\mathbf{y})L(\mathbf{x},\omega)]\,\mathrm{d}y \tag{12}$$

Here:

$$\begin{aligned} T(\mathbf{x},\mathbf{y}) &= -\exp\left(-\int_0^y \mu_t(\mathbf{x}-s\omega) + \mu_n(\mathbf{x}-s\omega)\mathrm{d}s\right) \\ &= -\exp\left(-\int_0^y \bar{\mu}(\mathbf{x}-s\omega)\mathrm{d}s\right) \end{aligned} \tag{13}$$

Also, the estimator is modified to:

$$\langle L(\mathbf{x},\omega)\rangle = \frac{T(\mathbf{x},\mathbf{y})}{P(\mathbf{y})}\,[\mu_a(\mathbf{y})L_e(\mathbf{y},\omega) + \mu_s(\mathbf{y})L_s(\mathbf{y},\omega) + \mu_n(\mathbf{y})L(\mathbf{y},\omega)] \tag{14}$$

For this estimator, if we substitute the probability function with $P(\mathbf{y}) = T(\mathbf{x},\mathbf{y})\bar{\mu}(\mathbf{y})$, then it can be simplfied to:

$$\langle L(\mathbf{x},\omega)\rangle = \frac{1}{\bar{\mu}(\mathbf{y})}\,[\mu_a(\mathbf{y})L_e(\mathbf{y},\omega) + \mu_s(\mathbf{y})L_s(\mathbf{y},\omega) + \mu_n(\mathbf{y})L(\mathbf{y},\omega)] \tag{15}$$

Now we introduce probabilities $P_a(\mathbf{x})$, $P_s(\mathbf{x})$ and $P_n(\mathbf{x})$ to allow probabilistically evaluating each of the emission, in-scattering, and null-collision radiance terms:

$$\langle L(\mathbf{x},\omega)\rangle = \frac{1}{\bar{\mu}(\mathbf{y})} \\ \left[P_a(\mathbf{y})\frac{\mu_a(\mathbf{y})}{P_a(\mathbf{y})}L_e(\mathbf{y},\omega) + P_s(\mathbf{y})\frac{\mu_s(\mathbf{y})}{P_s(\mathbf{y})}L_s(\mathbf{y},\omega) + P_n(\mathbf{y})\frac{\mu_n(\mathbf{y})}{P_n(\mathbf{y})}L(\mathbf{y},\omega)\right] \tag{16}$$

The Russian roulette, which is used for unbiased campling, in mathematic is:

$$\int_0^1 f(x)dx = \int_0^p \frac{f(x)}{p}dx = \int_0^1 \mathcal{H}[x<p]\frac{f(x)}{p}dx \tag{17}$$

where $\mathcal{H}[x]$ is the heavyside function:

$$\mathcal{H}[x] = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases} \tag{18}$$

Then the estimator is:

$$\begin{aligned} \langle L(\mathbf{x},\omega)\rangle = &\frac{1}{\bar{\mu}(\mathbf{y})} \\ &[\int_0^1 \mathcal{H}[\xi_a < P_a(\mathbf{y})]\frac{\mu_a(\mathbf{y})}{P_a(\mathbf{y})}L_e(\mathbf{y},\omega)\mathrm{d}\xi_a \\ &+ \int_0^1 \mathcal{H}[\xi_s < P_s(\mathbf{y})]\frac{\mu_s(\mathbf{y})}{P_s(\mathbf{y})}L_s(\mathbf{y},\omega)\mathrm{d}\xi_s \\ &+ \int_0^1 \mathcal{H}[\xi_n < P_n(\mathbf{y})]\frac{\mu_n(\mathbf{y})}{P_n(\mathbf{y})}L(\mathbf{y},\omega)\mathrm{d}\xi_n] \end{aligned} \tag{19}$$

In standard delta tracking, the probability are defined as:

$$P_a(\mathbf{x}) = \frac{\mu_a(\mathbf{x})}{\bar{\mu}(\mathbf{x})} \quad P_s(\mathbf{x}) = \frac{\mu_s(\mathbf{x})}{\bar{\mu}(\mathbf{x})} \quad P_n(\mathbf{x}) = \frac{\mu_n(\mathbf{x})}{\bar{\mu}(\mathbf{x})} \tag{20}$$

Note that $\bar{\mu} \geq \sup \mu_t(\mathbf{x})$ should be always satisfied.

Since they are sum to 1, we can simplfy the estimator to cut down the number of sampling random numbers and the branches as the form:

$$\begin{aligned} \langle L(\mathbf{x},\omega)\rangle = &\int_0^1 \mathcal{H}[\xi < P_a(\mathbf{y})]L_e(\mathbf{y},\omega) \\ &+ \mathcal{H}[P_a(\mathbf{y}) < \xi < 1 - P_n(\mathbf{y})]L_s(\mathbf{y},\omega) \\ &+ \mathcal{H}[1 - P_n(\mathbf{y}) < \xi]L(\mathbf{y},\omega_j)\mathrm{d}\xi \end{aligned} \tag{21}$$

Fig.?? (left) shows the properties of the delta tracking.

Here is the pseudo-code of delta tracking:

Algorithm 1 Pseudocode of Delta Tracking Algorithm

Require: $\mathbf{x}$: The collision point; ω: The traveling direction;
Ensure: Next collision point or next travel direction

```
while true do
    t ← −ln(1−ζ)/μ̄
    x ← x − t × ω
    if ξ < μ_a(x)/μ̄ then
        return L_e(x, ω)
    else if ξ < 1 − μ_n(x)/μ̄ then
        ω ← sample ∝ f_p(ω)
    else
        continue
    end if
end while
```

where ζ and ξ are random numbers in $[0,1]$.

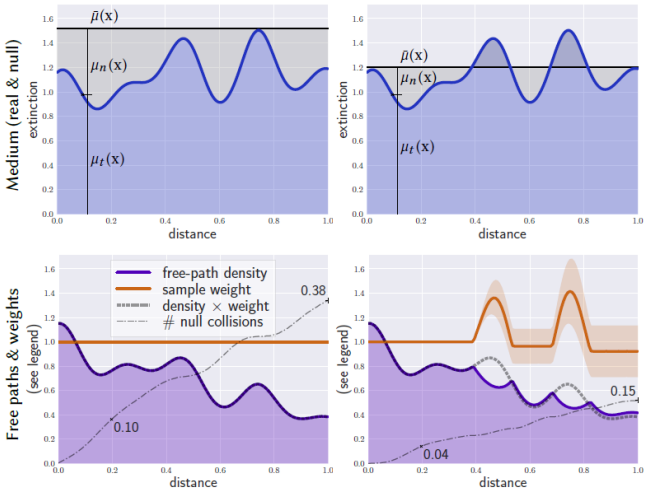


Fig. 4. The left shows delta tracking and the right shows weighted delta tracking. Free-path density is the PDF of the length of the free-path. Weight is the adjusted sample weight that used in weighted delta tracking. Number of null collsions is the cumulative number of null collsions at distance to the starting position. Density × weight compensated the effect of the negative extinction coefficient, though it leads to a higher standard deviation.

2.5 Weighted Delta Tracking

The constrain of the delta tracking is that the free-path-sampleing coefficient should be the close upper bound of the extinction coefficient. If the extinction coefficient is large at some point while low at others, then delta tracking is inefficient since most of the samples are rejected. To deal with this problem, two methods are proposed. One is to partition the space into several subspace with proper extinction coefficient in each subspace. The spatial partitioning algorithm are kd-tree or grid in practice, we will talk it later.

Another method is weight delta tracking. It looses the requirement that the free-path-sampleing coefficient can be smaller than the extinction coefficient: $\bar{\mu} < \mu_t(\mathbf{x})$. This introduces a negative fictitious coefficient which is unreasonable in real-world but efficient for computing.

So we define the probability as:

$$\begin{aligned} P_a(\mathbf{x}) &= \frac{\mu_a(\mathbf{x})}{\mu_t(\mathbf{x}) + |\mu_n(\mathbf{x})|} \\ P_s(\mathbf{x}) &= \frac{\mu_s(\mathbf{x})}{\mu_t(\mathbf{x}) + |\mu_n(\mathbf{x})|} \\ P_n(\mathbf{x}) &= \frac{|\mu_n(\mathbf{x})|}{\mu_t(\mathbf{x}) + |\mu_n(\mathbf{x})|} \end{aligned} \tag{22}$$

And the corresponding estimator is (19).

And as we can see, if we sample a distance where $\mu_n(\mathbf{x})$ is positive, then the weight will stay 1 no change. If it is negative, then it means that there are actually two event will happen, absorption or scattering. But the preset probability is not the actual probability, so we need to adjust the probability using weight. The weight is always greater equal to 1 theoretically.

Here is the pseudo-code for weighted delta tracking:

Algorithm 2 Pseudocode of Weighted Delta Tracking Algorithm

Require: $\mathbf{x}$: The collision point; ω: The traveling direction;
Ensure: Next collision point or next travel direction

$w \leftarrow 1$
while true **do**
  $t \leftarrow -\frac{\ln(1-\zeta)}{\bar{\mu}}$
  $\mathbf{x} \leftarrow \mathbf{x} - t \times \omega$
  if $\xi < P_a(\mathbf{x})$ **then**
    return $w \times \frac{\mu_a(\mathbf{x}) L_e(\mathbf{x},\omega)}{\bar{\mu} P_a(\mathbf{x})}$
  else if $\xi < 1 - P_n(\mathbf{x})$ **then**
    $\omega \leftarrow$ sample $\propto f_p(\omega)$
    $w \leftarrow w \times \frac{\mu_s(\mathbf{x})}{\bar{\mu} P_s(\mathbf{x})}$
  else
    $w \leftarrow w \times \frac{\mu_n(\mathbf{x})}{\bar{\mu} P_n(\mathbf{x})}$
  end if
end while

where ζ and ξ are random numbers in $[0, 1]$.

Symbol & Value	Description
μ_a^c	absorption coef. of control volume
μ_s^c	scattering coef. of control volume
$\mu_t^c = \mu_a^c + \mu_s^c$	extinction coef. of control volume
$\mu_a^r(\mathbf{x}) = \mu_a(\mathbf{x}) - \mu_a^c$	absorption coef. of residual volume
$\mu_s^r(\mathbf{x}) = \mu_s(\mathbf{x}) - \mu_s^c$	scattering coef. of residual volume
$\mu_t^r(\mathbf{x}) = \mu_t(\mathbf{x}) - \mu_t^c$	extinction coef. of residual volume
$\mu_n(\mathbf{x}) = \mu - \mu_t^c - \mu_t^r(\mathbf{x})$	null-collision coef.

Table 1. the symbol table for control volume and residual volume

3 DECOMPOSITION TRACKING

3.1 Analog Decomposition Tracking

Algorithm 3 Pseudocode of Weighted Decomposition Tracking Algorithm

Require: $\mathbf{x}$: The collision point; ω: The traveling direction;
Ensure: Next collision point or next travel direction

$w \leftarrow 1$
while true **do**
  $t \leftarrow -\frac{\ln(1-\zeta)}{\bar{\mu}}$
  $\mathbf{x} \leftarrow \mathbf{x} - t \times \omega$
  $F \leftarrow 0$
  if $\xi < (F \leftarrow F + P_a^c(\mathbf{x}))$ **then**
    return $w \times \frac{\mu_a^c(\mathbf{x}) L_e(\mathbf{x},\omega)}{\bar{\mu} P_a^c(\mathbf{x})}$
  else if $\xi < (F \leftarrow F + P_s^c(\mathbf{x}))$ **then**
    $\omega \leftarrow$ sample $\propto f_p(\omega)$
    $w \leftarrow w \times \frac{\mu_s^c(\mathbf{x})}{\bar{\mu} P_s^c(\mathbf{x})}$
  else if $\xi < (F \leftarrow F + P_a^r(\mathbf{x}))$ **then**
    return $w \times \frac{\mu_a^r(\mathbf{x}) L_e(\mathbf{x},\omega)}{\bar{\mu} P_a^r(\mathbf{x})}$
  else if $\xi < (F \leftarrow F + P_s^r(\mathbf{x}))$ **then**
    $\omega \leftarrow$ sample $\propto f_p(\omega)$
    $w \leftarrow w \times \frac{\mu_s^r(\mathbf{x})}{\bar{\mu} P_s^r(\mathbf{x})}$
  else
    $w \leftarrow w \times \frac{\mu_n(\mathbf{x})}{\bar{\mu} P_n(\mathbf{x})}$
  end if
end while

where ζ, ψ and ξ are random numbers in $[0, 1]$. Delta tracking and weight delta tracking are both useful. But decomposition tracking is able to reduce the number of queries of the volumetric medium by decomposition the medium into control component and residual component by handling the control volume analytically.

Before introducing the decomposition tracking, let us look a theorem first:

Theorem 3.1. *Let non-negative extinction coefficient $\mu_A(\mathbf{x}), \mu_B(\mathbf{x})$ be combined to $\mu_C(\mathbf{x}) = \mu_A(\mathbf{x}) + \mu_B(\mathbf{x})$. Let A, B, and C be independent random variables distributed according to the following CDF $F_X(t) = 1 - \exp\left(-\int_0^t \mu_X(\mathbf{x}_s)\mathrm{d}s\right)$ with $X \in \{A, B, C\}$. The CDFs of C and $\min(A, B)$ are identical.*

Proof. Let a random variable $D = \min(A, B)$, then:

$$\begin{aligned} F_D(t) &= P(D \le t) = P(\min(A,B) \le t) = 1 - P(\min(A,B) > t) \\ &= 1 - P(A > t)P(B > t) = 1 - T_A(t)T_B(t) \\ &= 1 - \exp(-\int_0^t \mu_A(\mathbf{x}_s)ds)\exp(-\int_0^t \mu_B(\mathbf{x}_s)ds) \\ &= 1 - \exp(-\int_0^t \mu_C(\mathbf{x}_s)ds) \\ &= F_C(t) \end{aligned} \tag{23}$$

which means that the CDFs of C and $\min(A, B)$ are identical. □

Algorithm 4 Pseudocode of Decomposition Tracking Algorithm

Require: $\mathbf{x}$: The collision point; ω: The traveling direction;
Ensure: Next collision point or next travel direction

```
while true do
    t^c ← −ln(1−ζ) / μ_t^c
    t^r ← 0
    while true do
        t^r ← t^r − ln(1−ψ) / (μ̄ − μ_t^c)
        if t^r > t^c then
            x ← x − t^c × ω
            if ξ < μ_a^c(x) / μ_t^c then
                return L_e(x, ω)
            else
                ω ← sample ∝ f_p(ω)
                break
            end if
        else
            if ξ < μ_a^r(x) / (μ̄ − μ_t^c) then
                x ← x − t^r × ω
                return L_e(x, ω)
            else if ξ < 1 − μ_n(x) / (μ̄ − μ_t^c) then
                x ← x − t^r × ω
                ω ← sample ∝ f_p(ω)
                break
            end if
        end if
    end while
end while
```

where ζ, ψ and ξ are random numbers in $[0, 1]$.

Then we can sample the free-path in both control volume and residual volume. And find the smaller distance for the sampling. Here we can combine the analytical sampling and delta tracking to form the decomposition tracking (see the algorithm.4).

3.2 Weighted Decomposition Tracking

The estimator now is:

$$\begin{aligned} &\langle L(\mathbf{x},\omega)\rangle = \frac{1}{\bar{\mu}(\mathbf{y})} \\ &[\int_0^1 \mathcal{H}[\xi_a^c < P_a^c(\mathbf{y})]\frac{\mu_a^c(\mathbf{y})}{P_a^c(\mathbf{y})}L_e(\mathbf{y},\omega)\mathrm{d}\xi_a^c \\ &+ \int_0^1 \mathcal{H}[\xi_a^r < P_a^r(\mathbf{y})]\frac{\mu_a^c(\mathbf{y})}{P_a^r(\mathbf{y})}L_e(\mathbf{y},\omega)\mathrm{d}\xi_a^r \\ &+ \int_0^1 \mathcal{H}[\xi_s^c < P_s^c(\mathbf{y})]\frac{\mu_s^c(\mathbf{y})}{P_s^c(\mathbf{y})}L_s(\mathbf{y},\omega)\mathrm{d}\xi_s^c \\ &+ \int_0^1 \mathcal{H}[\xi_s^r < P_s^r(\mathbf{y})]\frac{\mu_s^c(\mathbf{y})}{P_s^r(\mathbf{y})}L_s(\mathbf{y},\omega)\mathrm{d}\xi_s^r \\ &+ \int_0^1 \mathcal{H}[\xi_n < P_n(\mathbf{y})]\frac{\mu_n(\mathbf{y})}{P_n(\mathbf{y})}L(\mathbf{y},\omega)\mathrm{d}\xi_n] \end{aligned} \tag{24}$$

Similar to what we did in weighted delta tracking, we define the probability of each term:

$$\begin{aligned} P_a^c &= P_c P_{a|c} = \frac{\mu_t^c}{\bar{\mu}}\frac{\mu_a^c}{\mu_t^c} = \frac{\mu_a^c}{\bar{\mu}} \\ P_s^c &= P_c P_{s|c} = \frac{\mu_t^c}{\bar{\mu}}\frac{\mu_s^c}{\mu_t^c} = \frac{\mu_s^c}{\bar{\mu}} \end{aligned} \tag{25}$$

And for the sake of the sum of the probability to be one:

$$\begin{aligned} P_a^r(\mathbf{x}) &= \left(1 - \frac{\mu_t^c}{\bar{\mu}}\right)\frac{|\mu_a^r(\mathbf{x})|}{|\mu_a^r(\mathbf{x})| + |\mu_s^r(\mathbf{x})| + |\mu_n(\mathbf{x})|} \\ P_s^r(\mathbf{x}) &= \left(1 - \frac{\mu_t^c}{\bar{\mu}}\right)\frac{|\mu_s^r(\mathbf{x})|}{|\mu_a^r(\mathbf{x})| + |\mu_s^r(\mathbf{x})| + |\mu_n(\mathbf{x})|} \\ P_n(\mathbf{x}) &= \left(1 - \frac{\mu_t^c}{\bar{\mu}}\right)\frac{|\mu_n(\mathbf{x})|}{|\mu_a^r(\mathbf{x})| + |\mu_s^r(\mathbf{x})| + |\mu_n(\mathbf{x})|} \end{aligned} \tag{26}$$

So by combining the weighted delta tracking and decomposition tracking, we get the weighted decomposition tracking algorithm (algorithm.3).

3.3 Analysis

The decomposition tracking is efficient if the upper bound of the control component is high. If the control component is too low, it is not efficient compared to the original delta tracking since it need to compute one more sample. Furthermore, since the analog decomposition relies on delta tracking, it can handle only a single wavelength. We will deal with this problem by spectral tracking.

4 SPECTRAL TRACKING

Usually, when we talk about the radiance, we have accepted the fact that:

$$L(\mathbf{x},\omega) = \int_\lambda L(\mathbf{x},\omega,\lambda)\mathrm{d}\lambda \tag{27}$$

where λ is the wavelength of the ray.

In practice and for the photo-realistic, the properties of the volumetric medium are wavelength-dependent. Obviously, we can not cover all the wavelength of the ray. But the mixture of the red, green, and blue (RGB) color can recover all the color. So using a vector of N_λ wavelength is enough.

We define a single distribution for sampling collisions and counteract the discrepancy between this distribution and the true, per wavelength free-path distribution by reweighting just like weighted delta tracking. Ideally, to keep the variance low and reduce the number of sampling, we should set the free-path-sampling coefficient $\bar{\mu} = \sup_\lambda \mu_t(\mathbf{x}, \lambda)$.

Obviously, we can not set a uniform probabilities to cancel out the $\mu_\star(\mathbf{x}, \lambda)/\bar{\mu}$. But the author proposed to use the maximum and the average-based coefficient for the probability.

The maximum probability is:

$$\begin{aligned} P_a(\mathbf{x}) &= \max_\lambda(|\mu_a(\mathbf{x})|)c^{-1} \\ P_s(\mathbf{x}) &= \max_\lambda(|\mu_s(\mathbf{x})|)c^{-1} \\ P_n(\mathbf{x}) &= \max_\lambda(|\mu_n(\mathbf{x})|)c^{-1} \\ c &= \max_\lambda(|\mu_a(\mathbf{x})|) + \max_\lambda(|\mu_s(\mathbf{x})|) + \max_\lambda(|\mu_n(\mathbf{x})|) \end{aligned} \tag{28}$$

This will allow the number of local collision to be no more than 3 no matter how many wavelength we trace.

Next is average-based probability:

$$\begin{aligned} P_a(\mathbf{x}) &= \text{avg}_\lambda(|\mu_a(\mathbf{x})w(\mathbf{X}, \lambda)|)c^{-1} \\ P_s(\mathbf{x}) &= \text{avg}_\lambda(|\mu_s(\mathbf{x})w(\mathbf{X}, \lambda)|)c^{-1} \\ P_n(\mathbf{x}) &= \text{avg}_\lambda(|\mu_n(\mathbf{x})w(\mathbf{X}, \lambda)|)c^{-1} \\ c &= \sum_\star \text{avg}_\lambda(|\mu_\star(\mathbf{y})w(\mathbf{X}, \lambda)|) \end{aligned} \tag{29}$$

where $\mathbf{X}$ is the weight from previous path segment. We call this histroy-aware method. This will allow the number of local collision to be no more than N_λ.

By modifying the weight delta tracking algorithm, we can get:

Algorithm 5 Pseudocode of Spectral Tracking Algorithm

Require: $\mathbf{x}$: The collision point; ω: The traveling direction;
Ensure: Next collision point or next travel direction

$\hat{w} \leftarrow (1, \ldots, 1)_{N_\lambda}$
while true **do**
  $t \leftarrow -\frac{\ln(1-\zeta)}{\bar{\mu}}$
  $\mathbf{x} \leftarrow \mathbf{x} - t \times \omega$
  if $\xi < P_a(\mathbf{x})$ **then**
    return $\hat{w} \times \frac{\hat{\mu}_a(\mathbf{x})\hat{L}_e(\mathbf{x},\omega)}{\bar{\mu}P_a(\mathbf{x})}$
  else if $\xi < 1 - P_n(\mathbf{x})$ **then**
    $\omega \leftarrow$ sample $\propto f_p(\omega)$
    $\hat{w} \leftarrow \hat{w} \times \frac{\hat{\mu}_s(\mathbf{x})}{\bar{\mu}P_s(\mathbf{x})}$
  else
    $\hat{w} \leftarrow \hat{w} \times \frac{\hat{\mu}_n(\mathbf{x})}{\bar{\mu}P_n(\mathbf{x})}$
  end if
end while

In practice, these probabilities reduce the magnitude of undesired color noise and avoid bright chromatic outliers.

4.1 Residual Ratio Tracking

Ratio tracking and residual tracking,a way that is able to handle media with wavelength dependent extinction, can be combined to compute unbiased,low-variance transmittance estimates in general heterogeneous media.To be more specfic,ratio tracking is to leverage the information discovered during the tracking more efficiently instead of deducing "just" a binary answer,finally provides a piecewise constant approximation to the transmittance function.Residual tracking,in the other hand,is complementary to delta tracking and ratio tracking and combines numerical estimation with an analytic approximation, yielding a piecewise exponential solution.

4.1.1 Ratio Tracking.

Definition. Ratio tracking is defined by following rules: 1.Replacing the Russian roulette by a weight that is equal to the probability of colliding with a fictitious particle. 2.Continuing the random walk until it reaches the end-point of the ray instead of probabilistically terminating the random walk at one of tentative collision points. 3.keeping track of the joint probability of colliding points with fictitious particles at all the preceding tentative collisions during the construction.By following the above rules, we can define the transmittance when reading d as:

$$T(d) = \prod_{i=1}^{K}(1 - \frac{\mu(x_i)}{\bar{\mu}}) \tag{30}$$

The multiplicand in the product represents the ratio of fictitious to all particles at collision point x_i.
Algorithm 1 is the pseudo-code of ratio tracking:

Algorithm 6 Pseudocode of the ratio tracking estimator of transmittance along a ray with origin o, direction ω, and length d.

function RatioTrackingEstimator(o, ω, d)
  $t = 0$
  $T = 1$
  do:
    $\zeta = rand()$
    $t = t - \frac{log(1-\zeta)}{\bar{\mu}}$
    if $t \geq d$: **break**
    $T = T * (1 - \frac{\mu(o+t*\omega)}{\bar{\mu}})$
  while true
  return T
end function

Unbiasedness of Ratio Tracking. The unbiasedness of Ratio Tracking can be demonstrated if it follows the following equation:

$$E[T] = exp(-\int_0^d \mu(x)dx) = T(d) \tag{31}$$

Now we try to prove that conclusion. Denoting tau_i realizations of T_i,the expected value of T can be written as:

$$E[T] = E[\sum_{i=0}^{\infty} T_i] = \sum_{i=0}^{\infty} E[T_i] = \sum_{i=0}^{\infty} \int_{i=0}^{\infty} \tau_i dP(\tau_i) \tag{32}$$

To list T_0 and T_1 as example,T_0 represents all realizations with the first tentative free flight distance S_1 already exceeding d.Since the value of T_0 equals to 1,the expected value reduces to the probability of S_1 taking on values greater than d:

$$E[T_0] = P(S_1 > d) = \int_d^\infty \tau_i dP(\tau_i) p_s(x) dx = exp(-\bar{\mu}d) \tag{33}$$

In the case of T_1, which represents events where $S_1 \leq d$ and $S_2 \geq -S_1$,then the expectancy of T_1 reads:

$$\begin{aligned} E[T_1] &= \int_0^d \iota(x) P(S_2 > d - x) dP(S_1 \leq x) \\ &= \int_0^d \iota(x_1) P_s(x_1) \int_{d-x1}^\infty p_s(x_2) dx_2 dx_1 \\ &= \int_0^d \iota(x_1) \bar{\mu} exp(-\mu \bar{x}_1) exp(-\bar{\mu}(d - x_1)) dx_1 \\ &= \bar{\mu} exp(-\bar{\mu}d) \int_0^d \iota(x) dx \end{aligned} \tag{34}$$

Then we can analogously express the expected value of T_k, where represents realizations that satisfy $C_k \leq d$ and $C_k + 1 > d - C_k$,as:

$$\begin{aligned} E[T_k] = &\int_0^d \iota(x_1) P_s(x_1) \int_0^{d-x_1} \iota(x_1 + x_2) P_s(x_2) ... \\ &... \int_0^{d-\sum_{j=1}^{k-1} x_j} \iota(\sum_{j=1}^{k} x_j) p_s(x_k) \\ &\times \int_{d-\sum_{j=1}^{k-1} x_j}^\infty p_s(x_{k+1}) dx_{k+1} dx_k ... dx_2 dx_1 \\ = &\int_0^d \iota(x_1) \int_0^{d-x_1} \iota(x_1 + x_2) ... \int_0^{d-\sum_{j=1}^{k-1} x_j} \iota(\sum_{j=1}^{k} x_j) \\ &\times \int_{d-\sum_{j=1}^{k-1} x_j}^\infty \bar{\mu}^k exp(-\bar{\mu}(x_1 + ...x_k + (d - \sum_{j=1}^{k} x_j))) \\ &dx_{k+1} dx_k ... dx_2 dx_1 \\ = &\bar{\mu}^k exp(-\bar{\mu}d) \int_0^d \iota(x_1) \int_0^{d-x_1} \iota(x_1 + x_2) ... \\ &... \int_0^{d-\sum_{j=1}^{k-1} x_j} \iota(\sum_{j=1}^{k} x_j) dx_k ... dx_2 dx_1 \end{aligned} \tag{35}$$

Then we replace z_i for $\sum_{j=0}^{i} x_j$ to rewrite the equation as:

$$\begin{aligned} E[T_k] = \bar{\mu}^k exp(-\bar{\mu}d) &\int_0^d \iota(z_1) \int_0^{d-z_1} \iota(z_2) ... \\ &... \int_0^{d-z_{k-1}} \iota(z_k) dz_k ... dz_2 dz_1 \end{aligned} \tag{36}$$

The multiple integrals integrate $\prod_{j=1}^{k} \iota(z_j)$ over a k-dimensional simplex, which can be written concisely as:

$$E[T_k] = \bar{\mu}^k exp(-\bar{\mu}d) \frac{(\int_0^d \iota(x) dx)^k}{k!} \tag{37}$$

Finally,we express the expected value of T as following:]

$$\begin{aligned} E[T] &= \sum_{k=0}^{\infty} E[T_k] \\ &= \sum_{k=0}^{\infty} \bar{\mu}^k exp(-\bar{\mu}d) \frac{(\int_0^d \iota(x) dx)^k}{k!} \\ &= exp(-\bar{\mu}d) \sum_{k=0}^{\infty} \frac{(\bar{\mu} \int_0^d \iota(x) dx)^k}{k!} \\ &= exp(-\bar{\mu}d) exp(\bar{\mu} \int_0^d 1 - \frac{\mu(x)}{\bar{\mu}} dx) \\ &= exp(-\int_0^d \bar{\mu} dx) exp(\int_0^d \bar{\mu} - \mu(x) dx) \\ &= exp(-\int_0^d \mu(x) dx) \end{aligned} \tag{38}$$

Though ratio tracking is unbiased, its weakness is also obvious:the expected number of steps required to reach the endpoint of the ray is high,which directly depends on the value of the majorant extinction coefficient.That's why we introduce residual tracking.

4.1.2 *Residual Tracking.*

Definition. Before introducing the definition of residual trakcing, we first analyize the transmittion function,where $\mu_c(x)$ is the extinction coefficient which is a simplified version of the original $\mu(x)$ that enables expressing the optical thickness up to distance d in closed form $\tau_c(d)$:]

$$\begin{aligned} T(d) &= exp(-\int_0^d \mu(x) dx) \\ &= exp(-\int_0^d \mu_c(x) + \mu(x) - \mu_c(x) dx) \\ &= exp(-\tau_c(d)) exp(-\int_0^d \mu(x) - \mu_c(x) dx) \end{aligned} \tag{39}$$

The first exponential term, referred to as the control transmittance $T_c(d)$, represents a coarse approximation of $T(d)$, which is computed using the simplified extinction coefficient μ_c. The second exponential then serves as a correction that accounts for the difference between the control and the actual transmittance. We denote this exponential as the residual transmittanceT_r and refer to the integrand as the residual extinction coefficient $\mu_r(x) = \mu(x) - \mu_c(x)$. There is also one thing worth noticing—- in certain situations—depending on the value of $\mu_c(x)$—the residual extinction may be negative. Algorithm 2 is the pseudo-code of residual ratio tracking:

Expression. According to the equation metioned before, we can obtain the residual ratio tracking estimator as below:]

$$\langle T_r(d) \rangle_{RR} = \prod_{i=1}^{K} (1 - \frac{\mu_r(x_i)}{\bar{\mu_r}}) = \prod_{i=1}^{K} (1 - \frac{\mu(x_i) - \mu_c}{\bar{\mu_r}}) \tag{40}$$

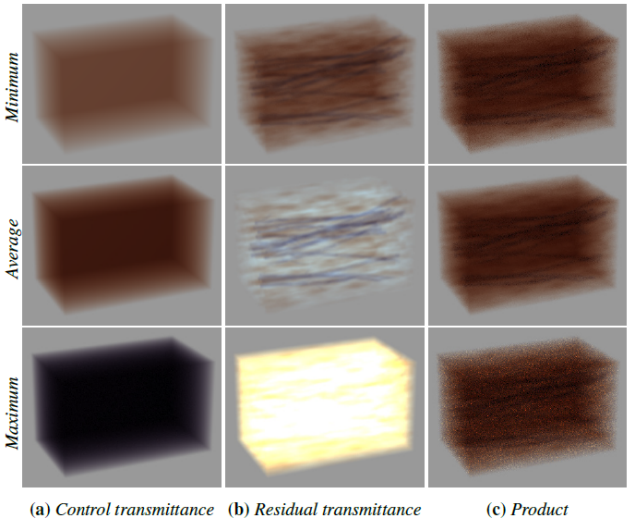
As our goal is to minimize the tracking cost by prolonging the tentative free-flight distances, it yields to finding the smallest $\bar{\mu_r}$ that

Algorithm 7 Pseudocode of the residual ratio tracking estimator for sampling transmittance along a ray with origin o, direction ω, and length d.

function ResidualRatioTrackingEstimator(o, ω, d)
  $t = 0$
  $T_c = exp(-\mu_c * d)$
  $T_r = 1$
  do:
    $\zeta = rand()$
    $t = t - \frac{log(1-\zeta)}{\bar{\mu_r}}$
    if $t \geq d$: **break**
    $T_r = T_r * (1 - \frac{\mu(o+t*\omega)-\mu_c}{\bar{\mu_r}})$
  while true
  return $T_c * T_r$
end function

still avoids negative multiplicands,which would prevent convergence of the above estimator.To ensure that $\mu_r(x)/\bar{\mu_r} \leq 1$, we set $\bar{\mu_r} = max(|\mu_r(x)|; 0 \leq x \leq d)$,i.e. the maximum absolute difference between $\mu(x)$ and μ_c along the ray.

For different values of μ_c, the visiulization of them is totally different,shown in the following figure, where the bright regions correspond to cases when the residual transmittance takes on values higher than 1 to correct for the overly low control transmittance.



(a) *Control transmittance* (b) *Residual transmittance* (c) *Product*

Fig. 5. Residual ratio tracking in a simple procedural volume with the control transmittance (a) computed using the minimum (top), the average (middle), and the maximum (bottom) extinction coefficient in the volume. The residual transmittance (b) used to compute the product (c) was estimated using 4 trackings only to emphasize the noise typical for each control extinction

- if $\mu_c = \mu_{min} = min(\mu(x)); 0 \leq x \leq d)$,i.e. the control extinction is underestimating, the control transmittance systematically overestimates the real transmittance
- if $\mu_c = \mu_{avg} = avg(\mu(x)); 0 \leq x \leq d)$,the control extinction along the ray matches the real extinction on average. The residual tracking thus corrects only for the local over- and underestimation of the control transmittance w.r.t. the real transmittance along the ray.
- if $\mu_c = \mu_{max} = max(\mu(x)); 0 \leq x \leq d)$,the control transmittance systematically underestimates the real transmittance and the residual tracking thus needs to produce values greater than 1 to scale the control up.

Unbiasedness of Residual Ratio Tracking. The unbiasedness of Residual Ratio Tracking can be demonstrated if it follows the following equation:

$$E[T] = exp(-\int_0^d \mu(x) - \mu_c dx) = \frac{T(d)}{exp(-\mu_c d)} \tag{41}$$

Now we try to prove that conclusion.As tge proof is very similar, we can directly mimic the above proof:

$$\begin{aligned} E[T] &= exp(-\bar{\mu_r} d) \sum_{k=0}^{\infty} \frac{(\bar{\mu_r} \int_0^d \iota(x) dx)^k}{k!} \\ &= exp(-\bar{\mu_r} d) exp(\bar{\mu_r} \int_0^d 1 - \frac{\mu(x) - \mu_c}{\bar{\mu_r}} dx) \\ &= exp(-\int_0^d \bar{\mu_r}) exp(\int_0^d \bar{\mu_r} - (\mu(x) - \mu_c) dx) \\ &= exp(-\int_0^d \mu dx(x) - \mu_c dx) \end{aligned} \tag{42}$$

4.2 Adaptive and unbiased sampling using kd-tree based space partitioning

As it's a time-consuming process to find a new scattering event in inhomogeneous participating meadia,we use an adaptive and unbiased sampling technique using kd-tree based space. partitioning. During preprocessing, it partitions the analytical space (the bounding box of the medium) into sub-spaces (partitions,presented in kd-tree) according to the spatial variation of the mean free path in the medium. During rendering, the locations of the scattering events are determined adaptively using the kd-tree.
Here is the pseudo-code of WoodcockTracking and kdTreeFreePath-Sampling:

Algorithm 8 Pseudocode of WoodcockTracking Algorithm

Require: $x_0, \vec{\omega}$:The ray starting at x_0 in direction $\vec{\omega}$
  k_M:The majorant extinction coefficient
  $(d_{min}, d_{max}]$:The interval of the ray to evaluate
Ensure: The free path d to the next scattering event
  $d \leftarrow d_{min}$-ln(1-rand())/k_M
  while $d \leq d_{max} \wedge k(x_0 + d\vec{\omega})/k_M < rand()$ **do**
    $d \leftarrow d$-ln(1-rand())/k_M
  end while
  return d

4.2.1 *Unbiased Sampling and Space Partitioning.*

Algorithm 9 Pseudocode of kdTreeFreePathSampling Algorithm

Require: $x_0, \vec{\omega}$:The ray starting at x_0 in direction $\vec{\omega}$
Ensure: The free path d to the next scattering event
while true **do**
  $kdTreeNode\ p \leftarrow findNextLeafNode()$
  if $p = nil$ **then**
    return INFINITY
  end if
  $(d_{min}, d_{max}) \leftarrow intersectionBetweenRayAndNode(p)$
  $k_M \leftarrow majorantExtinctionCoefficientStoredIn(p)$
  $d \leftarrow WoodcockTracking(x_o, \vec{\omega}, k_M, d_{min}, d_{max})$
  $d_{isect} \leftarrow INFINITY$
  if $intersectionWithObjectSurfaces(x_0, \vec{\omega}, p)$ **then**
    $d_{isect} \leftarrow distanceToTheIntersection()$
  end if
  if $d \geq d_{isect}$ **then**
    return IntersectionEvent
  end if
  if $d < d_{max}$ **then**
    return d
  end if
end while

Strategy. Suppose the ray starting at x_0 in direction $\vec{\omega}$ passes through two adjacent partitions i and j, with $k_M^{(i)}$ and $k_M^{(j)}$ being their majorant extinction coefficients. Let the distances from x_0 to the intersections with those partitions be s, q, and t.
First,to sample the free path in the interval $(s, t]$,we first deal with the $(s, q]$) by using WoodcockTracking Algorithm,and if the free pathexceeds q, we rewind the free path back to q and proceed to interval $(q, t]$ and use that algorithm again.By rewinding the free path back to q, we can ensure the unbiasedness of the sampling.

Proof of Unbiasedness. To prove unbiasedness, we need to show taht we can obtain the free path d with probability:

$$\begin{aligned} &P(x' = x_0 + d\vec{\omega} \wedge s < d \leq t) = e^{-\tau(x_0+s\vec{\omega},x')}k(x') \\ &P(d > t) = e^{-\tau(x_0+s\vec{\omega},x_0+t\vec{\omega})} \end{aligned} \tag{43}$$

First, we obtain d_1 during the sampling for the first interval (s,q] with the probability:

$$\begin{aligned} &P(x' = x_0 + d_1\vec{\omega} \wedge s < d_1 \leq q) = e^{-\tau(x_0+s\vec{\omega},x')}k(x') \\ &P_1(d_1 > q) = e^{-\tau(x_0+s\vec{\omega},x_0+q\vec{\omega})} \end{aligned} \tag{44}$$

Similarily, we obtain d_2 during the sampling for the second interval (q,t]

$$\begin{aligned} &P(x' = x_0 + d_2\vec{\omega} \wedge q < d_2 \leq t) = e^{-\tau(x_0+q\vec{\omega},x')}k(x') \\ &P_1(d_2 > t) = e^{-\tau(x_0+q\vec{\omega},x_0+t\vec{\omega})} \end{aligned} \tag{45}$$

Since the sampling for the second interval is executed if and only if d_1 exceeds q during the sampling for the first interval, we obtain

$$\begin{aligned} P(x' = x_0 + d\vec{\omega} \wedge s < d \leq q) &= P_1(x' = x_0 + d\vec{\omega} \wedge s < d \leq q) \\ &= e^{-\tau(x_0+s\vec{\omega},x')}k(x') \\ P(x' = x_0 + d\vec{\omega} \wedge q < d \leq t) &= P_1(d_1 > q)P_2(x' = x_0 + d\vec{\omega} \wedge q < d \leq t) \\ &= e^{-\tau(x_0+s\vec{\omega},x_0+q\vec{\omega})}e^{-\tau(x_0+q\vec{\omega},x')}k(x') \\ &= e^{-\tau(x_0+s\vec{\omega},x')}k(x') \\ P(d > t) = P_1(d > q)P_2(d > t) &= e^{-\tau(x_0+s\vec{\omega},x_0+q\vec{\omega})}e^{-\tau(x_0+q\vec{\omega},x_0+t\vec{\omega})} \\ &= e^{-\tau(x_0+s\vec{\omega},x_0+t\vec{\omega})} \end{aligned} \tag{46}$$

4.2.2 Unbiased Sampling and Space Partitioning. The performance of the sampling technique shown in previous section highly depends on the partitioning. If we partition a nearly homogeneous interval, an additional iteration is required to rewind the free path to q when proceeding to the adjacent interval, and this would be a waste compared to the case with no partitioning. On the other hand, we can have more benefit, if $k_M^{(i)}$ in a partition is much smaller than kM for the entire space. These observations would give us a stopping criterion for the partitioning process, as well as a decision criterion for the partitioning location q.

5 IMPLEMENTATION

The implementation is based on HW3 and HW4 of YiHengWu, to which our group members add a lot more functions and classes so that it can fit the condition of volume rendering. It follows the PBRT naming style for variables and functions so that those who are familiar with PBRT will find it comfortable to read the code.

The PBRT-style framework splits the tracer into independent parts: shape, primitive, aggregate, sampler, distribution, scattering, material, texture, light, medium. These information accompanied with the camera are recorded in the scene settings. The scene is finally rendered by the integrator and an image is outputted.

We assume that the readers are familiar with PBRT so we will only focus on medium, integrator and aggregate.

5.1 Volume

See 'medium/volume.h' and 'medium/volume.cpp' for more details.

The volume framework is based on OpenVDB, thus a volume is simply a OpenVDB grid with some interfaces to access the internal value. A smoke can be mostly described by its density with a VDB float grid. A problem is that the grid is described in an unsigned integer coordinate (grid coordinate) so that rays should be transformed from the world coordinate to the grid coordinate before tracking. That is why 'LocalToGrid()' is a necessary virtual function of 'class Volume'.

```
class Volume:
public:
virtual Transform LocalToGrid() const = 0;
```

We will focus on the child class 'class VolumeFloatGrid' and its child class 'class VolumeFloatKDTree'.

The reason why we also need a KDTree is that simply using the uniform grid for decomposition tracking will always result in bad performance because the minimum value of grid, which is chosen as μ_t^c, is always very small. So we use a KDTree to split the grid into blocks where the minimum value inside each block is expected to be higher. Our implementation of the KDTree is based on the article introduced above.

```
class VolumeFloatGrid : public Volume;
class VolumeFloatKDTree : public VolumeFloatGrid;
```

5.1.1 Grid. The implementation of 'class VolumeFloatGrid' is quite simple. It simply provides interfaces between our framework and OpenVDB data structure. Information about the grid including the max and min value and the boundary are recorded as fields to enable fast accessing.

5.1.2 KDTree. The KDTree is the child class of the grid, with node-level fields including the max and min value, split axis and split position. Information about a node is compressed and recorded into 64 bits. The format of the tree is shown here.

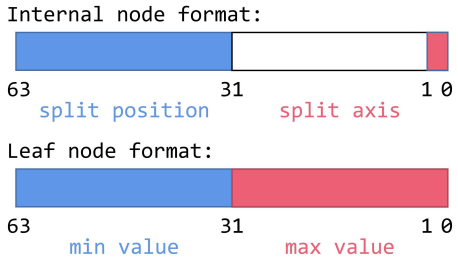


Fig. 6. KDTree node format.

The lowest two bits (split axis) is used to indicate whether a node is a internal or leaf node. For internal nodes, split axis $\in 0, 1, 2$ indicating x, y, z axis. For leaf nodes, the lowest two bits are fixed to 3.

You may have noticed that split axis shares two bits with max value so that for leaf nodes, the lowest two bits of the max value is always '0b11', which results in some acceptable errors but achieves great speedup.

You may also have noticed that parent nodes do not record the index of their child nodes. This makes our kdTree different from PBRT. We compressed the kdTree into a heap where children of node at array index i are located at $i << 1$ and $(i << 1) + 1$. We chose this design to compress the node into 64-bits, making it more cache-friendly for 64-bit-cacheline systems.

Following this format, 'class VolumeFloatKDTree' provides 4 inline functions to get split axis, split position of internal nodes and max, min value of leaf nodes. Here shows the interfaces, where 'idx' is the index of the kdTree array with 64-bit stride.

```
class VolumeFloatKDTree : public VolumeFloatGrid:
public:
inline int KDTreeSplitAxis(int idx) const;
inline int KDTreeSplitPos(int idx) const;
inline int KDTreeMaxValue(int idx) const;
inline int KDTreeMinValue(int idx) const;
private:
void* kdTree;
```

Building of the KDTree is based on the article introduced above. Since the implementation is simple, it will not be discussed here.

5.2 Medium

See 'medium/medium.h', 'medium/medium.cpp' for more details.

This section will focus on the interface design of the medium system. Readers can skip this part if they are familiar with PBRT.

As we have fully explained, the volume path tracing procedure primarily needs 3 building blocks: sampling scattering distance, evaluating transmittance and sampling scattering direction.

5.2.1 Sample Distance. Weighed delta tracking and weighed decomposition tracking are used. Since these two solutions are based on reject sampling, a random number generator (or a sampler in our framework) is needed. The current weight is also given as a parameter, which will be updated during samples. Sampling results including the position is recorded in a 'MediumInteraction'. Here is the interface.

```
class Medium:
public:
virtual Spectrum Sample(
const Ray& rayWorld,   // World-coord ray
Sampler& sampler,      // Random generator
Spectrum* weigh,       // Weighed tracking
MediumInteraction* mi // Sampling results
) const = 0;
```

The return value is a spectrum used to update β in integrators. It will be explained in detail later.

5.2.2 Evaluate Transmittance. Our implementation of evaluating transmittance is based on radio tracking and residual radio tracking, thus a sampler is also needed. We follow the rule that the transmittance to evaluate is between $t = 0$ and $t =$ rayWorld.tMax. Here is the interface.

```
class Medium:
public:
virtual Spectrum Tr(
const Ray& rayWorld, // World-coord ray
Sampler& sampler,    // Random generator
) const = 0;
```

5.2.3 Sample Direction. We simply sample the HenyeyGreenstein phase function so that the interface is similar to PBRT.

```
class PhaseFunction:
public:
virtual Float P(
const Vector3f& wo, // Omega out
const Vector3f& wi  // Omega in
) const = 0;
virtual Float Sample_P(
const Vector3f& wo,
Vector3f* wi,
```

```
const Point2f& samples
) const = 0;
class HenyeyGreenstein : public PhaseFunction;
```

5.3 Surface

See 'medium/medium.h', 'medium/medium.cpp' for more details.

To combine surface rendering and volume rendering, it is necessary to know whether the ray is travelling through air or inside some medium such as smoke. The simplest approach is assigning every primitive a surface, which records the medium outside the primitive surface and inside the primitive surface.

```
class Surface:
public:
const Medium* mediumIn; // Medium inside
const Medium* mediumOu; // Medium outside
```

Then every geometric primitive should be constructed with a surface instance.

```
class GeometricPrimitive : public Primitive:
public:
const Surface surface;
```

When 'GeometricPrimitive::Intersect()' is called, surface information should be recorded in a 'SurfaceInteraction' instance. So 'class SurfaceInteraction' should also contain a 'Surface' instance as its field.

```
class SurfaceInteraction : public Interaction:
public:
Surface surface; // Surface of hit primitive
```

It should be noticed that it is difficult for the framework to handle overlaps of mediums. Take this picture as an example.

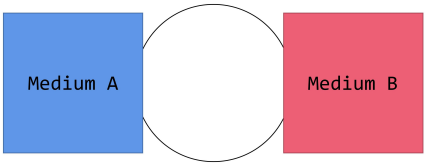


Fig. 7. Weakness of the surface interface.

Since 'class Surface' contains only one pointer for the internal and external medium, it is impossible to handle this two-medium case. So the middle sphere has to be split into two hemispheres.

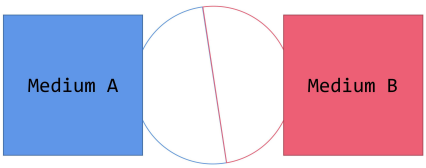


Fig. 8. Solution to the weakness of the surface interface.

Now that the data structures of the framework is clear, we will continue to the volume path tracing integrator.

5.4 Integrator

See 'volumePathSamplerIntegrator.h', 'volumePathSamplerIntegrator.cpp' for more details.

While 'class SamplerIntegrator' implements 'void Render()' to handle the rendering of an image, every child classes derived form 'class SamplerIntegrator' should implement 'Spectrum Integrate()' to evaluate the radiance of a given ray casted to the scene. The method 'Spectrum Integrate()' shares the similar meaning with 'Spectrum Li()' in PBRT.

To extend surface path tracing to volume path tracing, we first introduce the assumptions (restrictions) of our implementation. Then, we will mathematically prove the correctness of our integrator. Finally, we will explain the details of the implementation.

5.4.1 Assumptions. It is difficult to implement emissive mediums because of the complicity to randomly sample a volume light. Even the mathematical proof will be much more complicated, so we assume that all mediums are non-emissive.

5.4.2 Mathematics. We start from volume rendering equation (VRE) with surface contributions.

$$L(\mathbf{x},\omega) = \int_0^z T(\mathbf{x},\mathbf{y})[\mu_a(\mathbf{y})L_e(\mathbf{y},\omega) + \mu_s(\mathbf{y})L_s(\mathbf{y},\omega)]dy + T(\mathbf{x},\mathbf{z})L(\mathbf{z},\omega) \tag{47}$$

For non-emissive mediums, $L_e(\mathbf{y},\omega) = 0$, so

$$L(\mathbf{x},\omega) = \int_0^z T(\mathbf{x},\mathbf{y})\mu_s(\mathbf{y})L_s(\mathbf{y},\omega)dy + T(\mathbf{x},\mathbf{z})L(\mathbf{z},\omega) \tag{48}$$

where

$$L_s(\mathbf{y},\omega) = \int_{S^2} f_p(\omega,\bar{\omega})L(\mathbf{y},\bar{\omega})d\bar{\omega} \tag{49}$$

Suppose camera is located at position p_0, p_1 is the first sampled position. Similarly define $\mathrm{p}_2, \mathrm{p}_3, \cdots$.

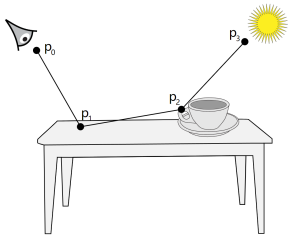


Fig. 9. Path tracing.

For path tracing, we firstly consider only the two simplest cases: both $\mathrm{p}_i, \mathrm{p}_{i+1}$ are in the air (no medium) and both $\mathrm{p}_i, \mathrm{p}_{i+1}$ are in some medium. Similar to the proof of surface path tracking in PBRT, we will derive the iteration of β and 'UniformSampleOneLight()' of these two cases. These two situations will be extended to more complicated cases later.

For the first case where both p_i, p_{i+1} are in the air, taking p_0, p_1 as an example.

$$\begin{aligned} L(p_1 \to p_0) &= L_e(p_1 \to p_0) + L_r(p_1 \to p_0) \\ &= L_e(p_1 \to p_0) \\ &+ \int_{H^2(n)} L(\bar{p} \to p_1) f_r(\bar{p} \to p_1 \to p_0) |\cos\bar{\theta}| d\bar{\omega} \end{aligned} \tag{50}$$

where L_e is the direct illumination from position p_1 to position p_0, L_r is the indirect illumination from position p_1 to position p_0. It should be noticed that $\bar{p}$ is not ensured to be in air or in some medium so that domain of $\bar{p}$ should be split to those in air (on another surface), donated as $\bar{p}^s$ and those in some medium, donated as $\bar{p}^m$.

$$\begin{aligned} L(p_1 \to p_0) &= L_e(p_1 \to p_0) \\ &+ \int_{H^2(n)} L(\bar{p}^s \to p_1) f_r(\bar{p}^s \to p_1 \to p_0) |\cos\bar{\theta}| d\bar{\omega} \\ &+ \int_{H^2(n)} L(\bar{p}^m \to p_1) f_r(\bar{p}^m \to p_1 \to p_0) |\cos\bar{\theta}| d\bar{\omega} \end{aligned} \tag{51}$$

where $L(\bar{p}^s \to p_1)$ can be evaluated using the surface contribution part of VRE and $L(\bar{p}^m \to p_1)$ can be evaluated using the volume contribution part of VRE.

$$\begin{aligned} L(\bar{p}^s \to p_1) &= T(\bar{p}^s, p_1) L_e(\bar{p}^s \to p_1) + T(\bar{p}^s, p_1) L_r(\bar{p}^s \to p_1) \\ L(\bar{p}^m \to p_1) &= \int_{p_1}^{\bar{p}^m} T(p_1, p) \mu_s(p) L_s(p, \omega_{\bar{p}^m \to p_1}) ds \end{aligned} \tag{52}$$

Substitute $L(\bar{p}^s \to p_1)$ into the path tracing equation.

$$\begin{aligned} & L(p_1 \to p_0) \\ &= L_e(p_1 \to p_0) \\ &+ \int_{H^2(n)} T(\bar{p}^s, p_1) L_e(\bar{p}^s \to p_1) f_r(\bar{p}^s \to p_1 \to p_0) |\cos\bar{\theta}| d\bar{\omega} \\ &+ \int_{H^2(n)} T(\bar{p}^s, p_1) L_r(\bar{p}^s \to p_1) f_r(\bar{p}^s \to p_1 \to p_0) |\cos\bar{\theta}| d\bar{\omega} \\ &+ \int_{H^2(n)} L(\bar{p}^m \to p_1) f_r(\bar{p}^m \to p_1 \to p_0) |\cos\bar{\theta}| d\bar{\omega} \end{aligned} \tag{53}$$

Apply Monte Carlo to the above equation.

$$\begin{aligned} & L(p_1 \to p_0) \\ &= L_e(p_1 \to p_0) \\ &+ \frac{T(\bar{p}^s_{light}, p_1) L_e(\bar{p}^s_{light} \to p_1) f_r(\bar{p}^s_{light} \to p_1 \to p_0) |\cos\bar{\theta}|}{p(\omega_{light})} \\ &+ \begin{cases} \dfrac{L_r(\bar{p}^s_{bsdf} \to p_1) f_r(\bar{p}^s_{bsdf} \to p_1 \to p_0) |\cos\bar{\theta}|}{p(\omega_{bsdf})} & \text{sample in air} \\ \dfrac{L(\bar{p}^m_{bsdf} \to p_1) f_r(\bar{p}^m_{bsdf} \to p_1 \to p_0) |\cos\bar{\theta}|}{p(\omega_{bsdf})} & \text{sample in some medium} \end{cases} \end{aligned} \tag{54}$$

Define function

$$\begin{aligned} & \text{UniformSampleOneLight}(p_1 \to p_0) \\ &= \frac{T(\bar{p}^s_{light}, p_1) L_e(\bar{p}^s_{light} \to p_1) f_r(\bar{p}^s_{light} \to p_1 \to p_0) |\cos\bar{\theta}|}{p(\omega_{light})} \end{aligned} \tag{55}$$

There is still a problem that it is difficult to evaluate $L(\bar{p}^m_{bsdf} \to p_1)$ because p_1 is in the air while $\bar{p}^m_{bsdf}$ is in the medium. This problem can be solved by applying energy conservation while radiance is travelling in the air.

Since $\bar{p}^m_{bsdf}$ is in the medium and p_1 is in the air, there exists a position where radiance travels exactly from medium to the air. Take this picture as an example.

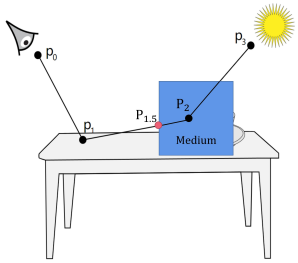


Fig. 10. Path tracing.

Let $p_{1.5}$ be the boundary between volume and air, apply energy conservation.

$$L(\bar{p}^m_{bsdf} \to p_1) = L(p_{1.5} \to p_1) \tag{56}$$

Then the problem is converted from evaluating radiance from medium to air, to evaluating radiance from medium to medium because the boundary can be both considered as an in-air position and an in-medium position.

The mathematics about evaluating radiance from medium to medium will be discussed then, which is exactly the second case discussed above where both p_i, p_{i+1} are in the air. Again take p_0, p_1 as an example. For in-medium case, p_1 is got by sampling distance, so we donate $p = p_1$ below.

$$\begin{aligned} & L(p \to p_0) \\ &= \int_{p_0}^{p} T(p_0, p) \mu_s(p) L_s(p \to p_0) ds \\ &= \int_{p_0}^{p} T(p_0, p) \mu_s(p) \int_{S^2} f_p(\bar{p} \to p \to p_0) L(\bar{p} \to p) d\bar{\omega} ds \end{aligned} \tag{57}$$

Similar to the in-air case, $\bar{\mathrm{p}}$ is not ensured to be in air or in some medium.

$$\begin{aligned}
&L(\mathrm{p}\to\mathrm{p}_0)\\
&=\int_{\mathrm{p}_0}^{\mathrm{p}} T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})\int_{S^2} f_p(\bar{\mathrm{p}}^s\to\mathrm{p}\to\mathrm{p}_0)L(\bar{\mathrm{p}}^s\to\mathrm{p})d\bar{\omega}ds\\
&+\int_{\mathrm{p}_0}^{\mathrm{p}} T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})\int_{S^2} f_p(\bar{\mathrm{p}}^m\to\mathrm{p}\to\mathrm{p}_0)L(\bar{\mathrm{p}}^m\to\mathrm{p})d\bar{\omega}ds\\
&=\int_{\mathrm{p}_0}^{\mathrm{p}} T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})\int_{S^2} f_p(\bar{\mathrm{p}}^s\to\mathrm{p}\to\mathrm{p}_0)\\
&\qquad\left(T(\bar{\mathrm{p}}^s,\mathrm{p})L_e(\bar{\mathrm{p}}^s\to\mathrm{p})+T(\bar{\mathrm{p}}^s,\mathrm{p})L_r(\bar{\mathrm{p}}^s\to\mathrm{p})\right)d\bar{\omega}ds\\
&+\int_{\mathrm{p}_0}^{\mathrm{p}} T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})\int_{S^2} f_p(\bar{\mathrm{p}}^m\to\mathrm{p}\to\mathrm{p}_0)L(\bar{\mathrm{p}}^m\to\mathrm{p})d\bar{\omega}ds
\end{aligned}\tag{58}$$

Apply Monte Carlo for sampling scattering distance.

$$\begin{aligned}
&L(\mathrm{p}\to\mathrm{p}_0)\\
&=\frac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}\int_{S^2} f_p(\bar{\mathrm{p}}^s\to\mathrm{p}\to\mathrm{p}_0)T(\bar{\mathrm{p}}^s,\mathrm{p})L_e(\bar{\mathrm{p}}^s\to\mathrm{p})d\bar{\omega}\\
&+\begin{cases}
\dfrac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}\displaystyle\int_{S^2} f_p(\bar{\mathrm{p}}^s\to\mathrm{p}\to\mathrm{p}_0)T(\bar{\mathrm{p}}^s,\mathrm{p})L_r(\bar{\mathrm{p}}^s\to\mathrm{p})d\bar{\omega} & \text{sample in air}\\
\dfrac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}\displaystyle\int_{S^2} f_p(\bar{\mathrm{p}}^m\to\mathrm{p}\to\mathrm{p}_0)L(\bar{\mathrm{p}}^m\to\mathrm{p})d\bar{\omega} & \text{sample in some medium}
\end{cases}
\end{aligned}\tag{59}$$

It should be noticed that for those tracking methods derived from delta tracking, $p(\mathrm{p})=T(\mathrm{p}_0,\mathrm{p})\mu_t(\mathrm{p})$. Then

$$\frac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}=\frac{\mu_s(\mathrm{p})}{\mu_t(\mathrm{p})}.\tag{60}$$

Apply Monte Carlo for sampling scattering direction.

$$\begin{aligned}
&L(\mathrm{p}\to\mathrm{p}_0)\\
&=\frac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}\frac{f_p(\bar{\mathrm{p}}^s_{light}\to\mathrm{p}\to\mathrm{p}_0)T(\bar{\mathrm{p}}^s_{light},\mathrm{p})L_e(\bar{\mathrm{p}}^s_{light}\to\mathrm{p})}{p(\omega_{light})}\\
&+\begin{cases}
\dfrac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}\dfrac{f_p(\bar{\mathrm{p}}^s_{phase}\to\mathrm{p}\to\mathrm{p}_0)T(\bar{\mathrm{p}}^s_{phase},\mathrm{p})L_r(\bar{\mathrm{p}}^s_{phase}\to\mathrm{p})}{p(\omega_{phase})} & \text{sample in air}\\
\dfrac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}\dfrac{f_p(\bar{\mathrm{p}}^m_{phase}\to\mathrm{p}\to\mathrm{p}_0)L(\bar{\mathrm{p}}^m_{phase}\to\mathrm{p})}{p(\omega_{phase})} & \text{sample in some medium}
\end{cases}\\
&=\frac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}\frac{f_p(\bar{\mathrm{p}}^s_{light}\to\mathrm{p}\to\mathrm{p}_0)T(\bar{\mathrm{p}}^s_{light},\mathrm{p})L_e(\bar{\mathrm{p}}^s_{light}\to\mathrm{p})}{p(\omega_{light})}\\
&+\begin{cases}
\dfrac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}T(\bar{\mathrm{p}}^s_{phase},\mathrm{p})L_r(\bar{\mathrm{p}}^s_{phase}\to\mathrm{p}) & \text{sample in air}\\
\dfrac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}L(\bar{\mathrm{p}}^m_{phase}\to\mathrm{p}) & \text{sample in some medium}
\end{cases}
\end{aligned}\tag{61}$$

Define function

UniformSampleOneLight($\mathrm{p}\to\mathrm{p}_0$)

$$=\frac{T(\mathrm{p}_0,\mathrm{p})\mu_s(\mathrm{p})}{p(\mathrm{p})}\frac{f_p(\bar{\mathrm{p}}^s_{light}\to\mathrm{p}\to\mathrm{p}_0)T(\bar{\mathrm{p}}^s_{light},\mathrm{p})L_e(\bar{\mathrm{p}}^s_{light}\to\mathrm{p})}{p(\omega_{light})}\tag{62}$$

There is also a similar problem that it is difficult to evaluate $L_r(\bar{\mathrm{p}}^s_{phase}\to\mathrm{p})$ because p is in some medium while $\bar{\mathrm{p}}^s_{phase}$ is in the air. This problem can also be solved similarly by energy conservation.

For now, the two simplest cases (both $\mathrm{p}_i,\mathrm{p}_{i+1}$ are in the air or in some medium) has been solved. Readers may have noticed that it is quite easy to derive them into more complicated cases by using an empty BSDF for the wrapping surface of the medium. That is, ray simply travels through the wrapping surface without scattering. The following picture shows the idea.

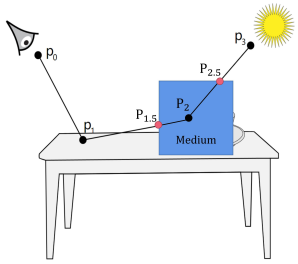


Fig. 11. Ultimate volume path tracing.

Path $\mathrm{p}_3\to\mathrm{p}_2\to\mathrm{p}_1\to\mathrm{p}_0$ is extended to $\mathrm{p}_3\to\mathrm{p}_{2.5}\to\mathrm{p}_2\to\mathrm{p}_{1.5}\to\mathrm{p}_1\to\mathrm{p}_0$, then nodes of each single path are located both in the air or in some medium. It is valid to combine them together, which derives our integrator for volume rendering.

Now we will derive the iteration of β and 'UniformSampleOneLight()', which is clearly proved above.

$$\beta(i)=\prod_{j=1}^{i}\begin{cases}
\dfrac{f_r(\mathrm{p}_{j+1}\to\mathrm{p}_j\to\mathrm{p}_{j-1})|\cos\theta_j|}{p_\omega(\mathrm{p}_{j+1}\to\mathrm{p}_j)} & \text{sample in air}\\
\dfrac{T(\mathrm{p}_j,\mathrm{p}_{j+1})\mu_s(\mathrm{p}_{j+1})}{p(\mathrm{p}_{j+1})} & \text{sample in some medium}
\end{cases}\tag{63}$$

UniformSampleOneLight($\mathrm{p}_j\to\mathrm{p}_{j-1}$)

$$=\begin{cases}
\dfrac{T(\bar{\mathrm{p}}^s_{light},\mathrm{p}_j)L_e(\bar{\mathrm{p}}^s_{light}\to\mathrm{p}_j)f_r(\bar{\mathrm{p}}^s_{light}\to\mathrm{p}_j\to\mathrm{p}_{j-1})|\cos\bar{\theta}|}{p(\omega_{light})} & \text{sample in air}\\
\dfrac{T(\mathrm{p}_{j-1},\mathrm{p}_j)\mu_s(\mathrm{p}_j)}{p(\mathrm{p}_j)}\cdot\dfrac{f_p(\bar{\mathrm{p}}^s_{light}\to\mathrm{p}_j\to\mathrm{p}_{j-1})T(\bar{\mathrm{p}}^s_{light},\mathrm{p}_j)L_e(\bar{\mathrm{p}}^s_{light}\to\mathrm{p}_j)}{p(\omega_{light})} & \text{sample in some medium}
\end{cases}\tag{64}$$

Then the radiance of a pixel can be determined. For surface path tracing, the radiance of a pixel can be rewritten to the following

iterative equation.

$$\begin{aligned}\text{radiance} = & L_e(\mathrm{p}_1 \to \mathrm{p}_0) \\ & + \beta_0 \text{UniformSampleOneLight}(\mathrm{p}_1 \to \mathrm{p}_0) \\ & + \beta_1 \text{UniformSampleOneLight}(\mathrm{p}_2 \to \mathrm{p}_1) \\ & + \cdots\end{aligned} \tag{65}$$

For volume path tracing, the iterative equation is too complicated to be described by mathematics so it will not be shown here.

5.4.3 Implementation. The implementation simply translate the above mathematics into code. Readers may find that our implementation is quite similar to PBRT. That is right because we accidentally chose the similar implementation approach with PBRT (but PBRT does not prove the correctness of their implementation). We will further discuss the restriction of PBRT and show the better implementation in the next section while introducing spectral weighed delta tracking and spectral weighed decomposition tracking.

Here is the pseudo code written with PBRT literate programming.

```
Spectrum
VolumePathSamplerIntegrator::Integrate(const Ray& r, const Scene& scene) const:
Spectrum radiance(Float(0));
Ray ray(r);
bool specularBounce = false;
Spectrum beta(Float(1));
Spectrum mediumScatteringWeigh(Float(1));
const Medium* medium = camera->medium;
for (int bounces = 0; ; ++bounces):
<<Intersect ray with scene and store intersection in si>>
<<Sample the participating medium and store intersection in mi, if present>>
// Handle an interaction with a medium or a surface
if (mi.IsValid()):
// Handle scattering at point in medium for volumetric path tracer
radiance += beta * UniformSampleOneLight(...);
<<Spawn ray from mi>>
else:
// Handle scattering at point on surface for volumetric path tracer
<<Possibly add emitted light at intersection>>
<<Terminate path if ray escaped or max depth was reached>>
// Compute scattering functions and skip over medium boundaries>>
si.ComputeScatteringFunctions(ray);
if (!si.bsdf):
<<Spawn ray from si>>
--bounces;
continue;
// Sample illumination from lights to find attenuated path contribution
radiance += beta * UniformSampleOneLight(...);
<<Sample BSDF to get new path direction>>
<<Possibly terminate the path with Russian roulette>>
return radiance;
```

For the integrator to run successfully, child classes of 'class Medium' should be implemented then, which will be introduced in the next section.

5.5 Density Medium

See 'densityMedium.h', 'densityMedium.cpp' for more details.

Information of volumes can be described by a floating point grid called density, accompanied by some mega information to translate density into μ_a, μ_s, μ_t.

It should be noticed that even density at a given position is simply a floating point, μ_a, μ_s, μ_t are spectrums. That is, they are wavelength-interdependent. So that spectral tracking (weighed delta tracking and weighed decomposition tracking) should be used. Readers familiar with PBRT may be confused since PBRT implements non-weighed delta tracking. That is because PBRT does not support wavelength-interdependent μ_t. See the source code of PBRT for more details.

In this section, we are proud to extend PBRT by implementing wavelength-interdependent mediums. We will first introduce the fields of the classes. We will then explain how to sample scattering distance and evaluate transmittance for mediums based on grid and KDTree.

5.5.1 Fields of Classes. Density is interpreted as μ_t in our implementation so the first step is to transform density to μ_t. This is done by providing a spectrum called 'toMut', then μ_t = density · toMut. Albedo = $\frac{\mu_s}{\mu_t}$ should also be provided to compute μ_a and μ_s from μ_t. Such an idea derives the interfaces of density mediums.

```
class HeteroMedium : public Medium;
class DensityGridMedium : public HeteroMedium:
protected:
const VolumeFloatGrid* density;
const Spectrum toMut;
const Spectrum albedo;
const PhaseFunction* phase;
const Transform worldToMedium;
class DensityKDTreeMedium : public HeteroMedium:
... // The same
```

For delta tracking, $\bar{\mu}$ should be determined during the construction. Since density is given, we choose $\bar{\mu} = \max(\text{density})$. $\frac{1}{\bar{\mu}}$ is also computed to make the sampling process faster.

```
class DensityGridDeltaMedium :
public DensityGridMedium:
private:
Float mubar;
Float mubar_inv;
```

For decomposition tracking, besides $\bar{\mu}$ and $\frac{1}{\bar{\mu}}$, μ_a^c, μ_s^c, μ_t^c, P_a^c, P_s^c, $P_{residual}$ should also be computed. Since it is not that easy to evaluate transmittance wavelength-interdependently by residual ratio tracking, a homogeneous choice of μ_t^c should also be computed, as well as the corresponding $\frac{1}{\mu^r}$.

```
class DensityGridDecompositionMedium :
public DensityGridMedium:
private:
Float mubar;
Float mubar_inv;
Spectrum muaC;
Spectrum musC;
Spectrum mutC;
Float paC;
Float psC;
Float pResidual;
Float mutC_homoC;
Float mubarR_homoC_inv;
```

5.5.2 Sampling Scattering Distance. As has mentioned above, we follow the similar interface for 'Medium::Sample()'. That is, the return value is the updating multiplier for β, which is exactly $\frac{T(\mathrm{p}_{j-1},\mathrm{p}_j)\mu_s(\mathrm{p}_j)}{p(\mathrm{p}_j)}$. For tracking methods based on delta tracking including decomposition tracking, it can be simplified to $\frac{\mu_s(\mathrm{p}_j)}{\mu_t(\mathrm{p}_j)}$. Thus

the return value is always the albedo. Our implementation assumes fixed albedo for a medium, different from PBRT.

The code simply translates weighed delta tracking and weighed decomposition tracking of Disney pseudo code into C++, except that 'weigh' is a parameter. When a ray is scattering inside a medium weigh is updated during each scattering process, until the ray finally travels out of the medium, then the weigh is reset to 'Spectrum(Float(1))'. The weigh is kept in the integrator as a variable called 'Spectrum mediumScatteringWeigh'.

Problem occurs when there are solid shapes inside a medium. Delta tracking and transmittance evaluation should never overreach these shapes. So a ray casting always has to be performed before calls to functions 'Sample()' and 'Tr()' in order to record the maximum sampling distance inside 'rayWorld.tMax'. 'tMax' should also be recomputed when ray is transformed from world coordinate to grid coordinate, as readers will see in our source code.

Pseudo code of weighed delta tracking for grid is shown here.

```
Spectrum
DensityGridDeltaMedium::Sample(
const Ray& rayWorld, Sampler& sampler,
Spectrum* weigh, MediumInteraction* mi) const:
<<Transform ray from world coordinate to grid coordinate>>
<<Ray casting to the bounding box and compute tMin, tMin>>
// Weighed delta tracking
Float t = tMin;
while (true):
// Step forward
t -= std::log(1 - sampler.Get1D()) * mubar_inv;
if (t > tMax):
break;
<<Compute sampling variables such as Pa, Ps, Pn>>
if (kexi < pa):
<<Set mi->emission>>
return Spectrum(Float(0)); // Terminate path
else if (kexi < 1 - pn):
<<Update weigh>>
<<Set mi>>
return albedo; // Return albedo to update beta
else:
<<Update weigh>>
return Spectrum(Float(1)); // Sampling failed
```

Code of weighed decomposition tracking for grid is shown in 'DensityGridDecompositionMedium::Sample()', which is quite similar so it will be shown here.

Tracking for KDTree is also straightforward. Readers can simply replace the intersection testing of traversal of PBRT KDTree with node-level delta tracking or decomposition tracking. But it should be noticed that if a real collision is sampled inside one node, other nodes should no longer be sampled, which is different from a KDTree containing primitives because a primitive may be located inside many nodes but a medium-grid block is located inside exactly one node.

The codes of 'DensityKDTreeDeltaMedium::Sample()' and 'DensityKDTreeDecompositionMedium::Sample()' is so simple that they will also not be included here.

5.5.3 Evaluating transmittance. Since ratio tracking is related to weighed delta tracking and residual ratio tracking is related to weighed decomposition tracking. We use ratio tracking for "delta" medium and residual ratio tracking for "decomposition" medium.

The following code shows the evaluation of transmittance by ratio tracking for grid.

```
Spectrum
DensityGridDeltaMedium::Tr(const Ray& rayWorld, Sampler& sampler) const:
<<Transform ray from world coordinate to grid coordinate>>
<<Ray casting to the bounding box and compute tMin, tMin>>
// Ratio tracking
while (true):
// Step forward
t -= std::log(1 - sampler.Get1D()) * mubar_inv;
if (t >= tMax):
break;
Float dense = density->Sample(rayMedium(t));
tr *= Spectrum(Float(1))
- Spectrum::Max(Spectrum(), dense * mubar_inv * toMut);
return tr;
```

However, for decomposition tracking, μ^c which is independent among wavelengths have to be used because those with interdependence is not supported by residual ratio tracking.

The code for the other three cases are not shown since they share too much similarities.

There is still one interface to implement for the integrator to comfortably evaluate transmittance. A ray may travel through many mediums before hitting a solid surface. For these cases, the accumulated transmittance should be evaluated. This is done by evaluating and multiplying the transmittance of all the mediums along the ray. So 'IntersectTr()' is used instead of 'Intersect()' in volume path tracing, which evaluates the transmittance along the ray and returns whether the ray hits a solid surface. See 'scene/scene.h', 'scene/scene.cpp' for the details of the interface.

```
Scene:IntersectTr(
const Ray& ray,          // World-coord ray
Sampler& sampler,        // Random generator
const Medium* medium,    // Medium with ray.o
Float* tHit,             // Hit time
SurfaceInteraction* si, // Interaction info
Spectrum* tr             // Transmittance
);
```

Now we are done.

5.6 Aggregate

See 'embreeAccel.h', 'embreeAccel.cpp' for more details.

Since volume global illumination is quite slow, we provide an accelerator based on Intel Embree. The scenes are rendered with 'embreeAccel' in default for performance, but we also provide interfaces for our self-written accelerators including KDTree and BVH.

```
#ifdef INTEL
shared_ptr<EmbreeAccel> aggregate =
make_shared<EmbreeAccel>(primitives);
#else
shared_ptr<BVH> aggregate =
make_shared<BVH>(primitives);
#endif
```

Readers can enable or disable Embree by setting 'INTEL' in cmake.

```
set(INTEL true)
```

6 TODO

We failed to implement everything because time is limited. Features not yet implemented are introduced in this section.

6.1 Emissive Medium

Has been proved above, our integrator does not support emissive medium. A better integrator should be mathematically proved.

6.2 Faster IntersectTr()

The function 'IntersectTr()' is quite slow because it may calls 'Intersect()' multiple times when there are multiple mediums in the scene. This can be solved by using Embree interfaces about masking.

6.3 MIS

We haven't updated the MIS for surface rendering to volume rendering because some interfaces of the framework have been updated to support volume rendering, so that MIS has to be written.

6.4 Better Interface for Surface

Has also been mentioned above, it is difficult for our interface to handle overlaps of solid shapes and mediums. A special pointer texture may be a solution, but storing and looking up such a texture may be expensive.

6.5 Packet Tracking

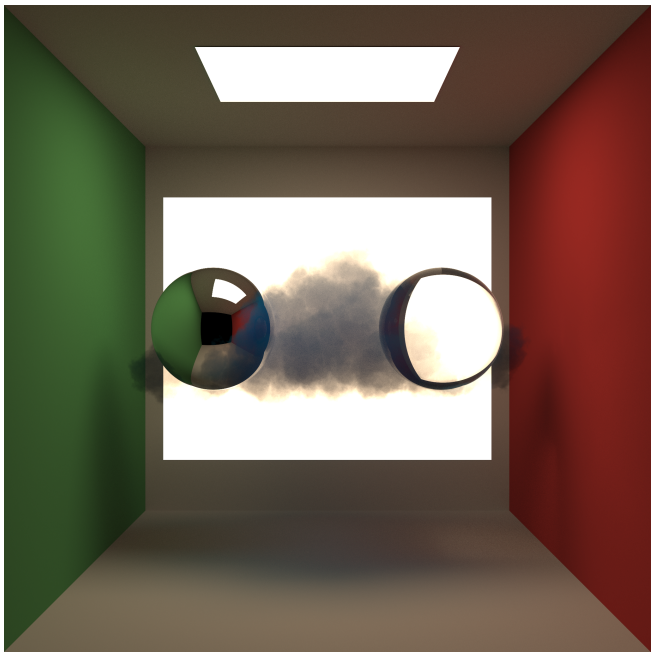
Since our implementation is originally not based on Embree, it does not support packet tracking very well. To implementation this feature, the whole framework should be rewritten.

6.6 Handling Floating Point Errors

PBRT restrictedly handle floating point errors especially nans. It is extremely important when rays are frequently bounced from surfaces because infinite loops will occur if rays are not correctly bounced.

7 RESULTS

On the right are the results of our projects.Though we have set up many many scenes in this project, finally we select two scenes to demonstrate our result.



8 CONTRIBUTION

Yiheng Wu. The realization of the main part of the project:integrator, kdtree structure for volume, delta tracking and decomposition tracking. The implementation part of the report.

Haizhao Dai. Setting up many scenes for the project.Build up thread-safe random number generator. The learning and test experiment of Embree repo. The introduction part and first two sections of the theoretical part of the report.

Xiaohan Wu. Setting up many scenes for the project.The implementation of several API,including the loader of openvdb, surface-related interfaces. The arrangement of past and present codes(eg.light etc). The collection of openvdb assets.The last two sections of the theoretical part of the report.