Université Paris 8

Master Création Numérique

parcours : Arts et Technologies de l'Image Virtuelle

Le rendu volumétrique en temps réel

Paul Hubert

Mémoire de Master 2

2019 - 2020

Sommaire

Sommaire (suite)

Résumé

Ce mémoire traite de la synthèse d'images en temps réel, c'est à dire calculées en un temps permettant l'interaction. Le jeu vidéo en est une application typique. C'est dans ce contexte que nous nous plaçons. Plus précisément, nous étudions le problème de la représentation d'objet volumiques. Ces objets ne sont pas définis par leurs seules surfaces. Ils interviennent par exemple dans le rendu des fumées ou des effets atmosphériques.

Dans une première partie, nous introduisons la thématique à l'aide d'exemples artistiques, notamment issus du jeu vidéo. Nous présentons ensuite une formulation physique au problème. Enfin, les principes algorithmiques du temps réel sont exposés et avec eux leurs limitations au regard de notre sujet.

Une seconde partie présente une sélection d'algorithmes de rendu volumétrique répandus dans l'industrie vidéoludique. Les points notables sont illustrés par des exemples d'implémentation.

La dernière partie développe notre processus d'expérimentation à travers deux stratégies originales.

Abstract

This dissertation is about real-time rendering. This term refers to the computer synthesis of images at a speed making interaction possible. More precisely, we address the problem of volumetric objects rendering in video games. Those objects are defined not only by their surfaces. Atmospheric fog or smoke are typical ones.

In a first part, we introduce the theme using artistic examples. We then present a physical formulation of our problem. Finally, we expose the principles of real-time specific rendering algorithms and their limitations regarding our subject.

A second part presents a selection of widely used volumetric rendering algorithms. Noteworthy details are illustrated by implementation samples.

Our last part presents two original implementations.

Note préliminaire

Langue des termes techniques

Le français est préféré lorsqu'il existe un mot évocateur du concept exprimé. Dans le cas contraire, l'anglais est utilisé. On choisira par exemple le terme *d'échantillonnage* plutôt que celui de *sampling,* alors que *billboard* sera favorisé sur le français *panneau* qui pourrait perdre le lecteur.

Extraits de code

Les extraits de pseudo-code sont rédigés dans une syntaxe proche de celle du language C++. Le code GPU (processeur graphique) suit selon les cas les syntaxes GLSL ou HLSL (issues des API graphiques Direct3D et OpenGL / Vulkan).

Visuels et technologies

Tous les visuels sont originaux, en dehors des cas où un auteur, artiste ou studio est mentionné. La technologie utilisée est indiquée en légende.

1 – Contexte

1.1 – Le rendu volumétrique dans l'art

1.1.1 – Définition

On entend par rendu volumétrique l'ensemble des techniques permettant de représenter des objets semi-transparents définis par un champ dans l'espace tridimensionnel. Dans le cadre de ce mémoire, cette définition comprend aussi bien les objets localisés que les effets atmosphériques à grande échelle.

Le rendu surfacique correspond au contraire à la représentation d'objets transmettant peu ou pas la lumière.

1.1.2 – Théorie artistique

L'art de la Renaissance repose sur un développement important des techniques picturales. Ces évolutions s'accompagnent d'une certaine augmentation du réalisme des œuvres que l'on retrouvera entre autres dans les courants baroque, classique, romantique et réaliste. Dans ce contexte, on cite souvent l'anatomie, les effets de lumière, la technique de la peinture à l'huile ou les lois de la perspective.

Les lois de la perspective sont géométriques (perspectives linéaire et curviligne, anamorphoses) mais également chromatiques. On parle de perspective aérienne pour décrire les techniques permettant de représenter la profondeur à travers les effets atmosphériques. Avec la distance, les couleurs sont plus bleues et les contours moins nets. Ces lois traduisent la réalité des interactions entre la lumière et la matière.

« Ce que tu veux cinq fois plus loin, fais le cinq fois plus bleu. » - Léonard de Vinci, carnets [0]

Figure 1.1. – Paysage avec la fuite en Egypte, Pieter Brueghel l'Ancien (1563)

Plus tard, les impressionnistes joueront avec ces techniques. Dans ses *Ponts de Waterloo,* Claude Monet livre une série de paysages brumeux. Ils s'appuient sur des variations atmosphériques très convaincantes.

Figure 1.2 – Pont de Waterloo, Londres, Claude Monet (1900)

Figure 1.3 – Pont de Waterloo, Jour gris, Claude Monet (1903)

1.2.3 – Dans le jeu vidéo

Dès les premiers jeux vidéo en trois dimensions, on retrouve des représentations d'objets semi-transparents. Explosions, impacts, il servent principalement la jouabilité et mettent l'emphase sur une action. Ils sont basés sur des images pré-rendues.

Figure 1.4 – Doom (id Software, 1993)
Les explosions sont représentées par des textures opaques

Figure 1.5 – *Dans* ***Mario 64 (Nintendo, 1996)****, les explosions et effets d'impacts sont transparents.*

Les effets atmosphériques à grande échelle se popularisent progressivement au fil de la génération des consoles 32/64 bits. Ils ne sont pas texturés et se limitent à une atténuation de la couleur proportionnelle à la distance. On parle de brouillard de distance (voir 2.1.1). Il permet notamment de limiter la distance d'affichage sans provoquer d'apparition brutale des objets.

Figure 1.6 – The Legend of Zelda* : *Ocarina of Time (Nintendo, 1998). *La teinte du brouillard évolue au fil de la journée*

Figure 1.7 – *Dans* ***Silent Hill (Konami, 1999)****, le brouillard participe au gameplay en dissimulant les ennemis.*

Avec la génération suivante, les effets de gameplay se multiplient. Le plus souvent, ils reposent toujours sur des textures semi-transparentes qui ne réagissent pas avec la lumière. Les effets atmosphériques se complexifient dans des mises en scène qui restent souvent statiques.

Figure 1.8 – Far Cry (Crytek / Ubisoft, 2004)

Figure 1.9 – God of War II (Sony Computer Entertainment, 2007)
Des ombres sont projetées sur l'atmosphère.

Au fil des améliorations technologiques, les objets volumétriques pré-calculés interagissent avec la lumière de façon dynamique. On voit également apparaître des systèmes interactifs dont le rendu réagit avec l'environnement et l'influence, comme dans *Little Big Planet 2*.

Figure 1.10 – Call of Duty: Modern Warfare 2 (Infinity Ward / Activision, 2009)
Les colonnes de fumée sont éclairées de manière cohérente.

Figure 1.11 – Little Big Planet 2 (Media Molecule / Sony Compute Entertainement, 2011)
Les volumes sont non seulement éclairés, mais ils projettent des ombres sur eux-mêmes ainsi que sur l'environnement.

Aujourd'hui, le standard, chez les productions dites *triple A*, consiste généralement en une combinaison d'effets pré-calculés et de systèmes dynamiques. On voit également se développer des solutions entièrement dynamiques, rendues possibles par l'évolution des processeurs graphiques (GPU). Elles sont limitées à des équipements haut de gamme et font généralement office de vitrine technique.

Figure 1.12 – Battlefield One (Dice, 2016)
Les effets basés sur des textures précalculées et l'atmosphère dynamique se combinent de manière réaliste.

Figure 1.13 – *En 2020,* ***Nvidia et Microsoft*** *proposent sur* ***Minecraft*** *une solution de rendu volumétrique totalement dynamique pour promouvoir la technologie RTX.*

1.2 – Principes physiques du rendu volumétrique

Avant d'aborder les paramètres spécifiques au rendu volumétrique et pour mieux les comprendre, nous rappellerons en premier lieu ceux liés au rendu physique réaliste en général et au rendu de surface. Les concepts seront présentés à la manière de l'ouvrage *Physically Based Rendering: From Theory to Implementation*, qui fait office de référence, en particulier dans l'image de synthèse pré-calculée.

1.2.1 – L'équation de rendu

L'équation de rendu décrit le transport de l'énergie lumineuse (on parle de radiance ou de luminance, noté L) depuis un point en direction d'un observateur [1]. C'est de cette équation que découle l'essentiel des algorithmes de rendu physique réaliste. On utilise souvent l'acronyme anglais PBR, pour *physically based rendering*.

$$L_o(\omega_o, p) = \int_{\Omega} L_i(\omega_i, p) \cdot f_r(\omega_i, \omega_o) \cdot (\omega_i \cdot n) \cdot d\omega_i$$

avec :

ω_o	=	direction point – observateur
p	=	point d'intérêt
L_o	=	radiance dans la direction ω_o
Ω	=	domaine d'intégration : ensemble des directions dans l'hémisphère d'axe polaire n
ω_i	=	variable d'intégration dans le domaine Ω
L_i	=	radiance dans la direction ω_i
f_r	=	réflectivité bidirectionnelle (BRDF)
n	=	vecteur normal

Figure 1.14 – L'équation de rendu

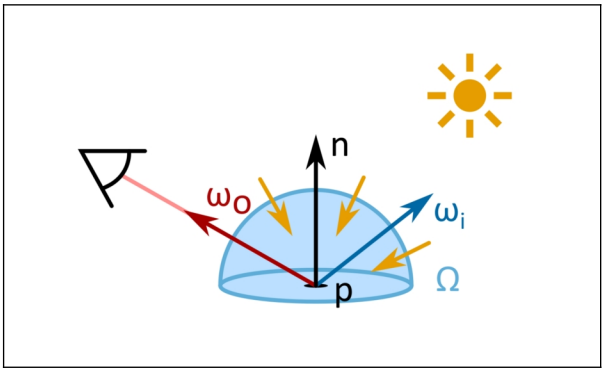


Figure 1.15 – Paramètres de l'équation de rendu : représentation graphique

$$\underbrace{L_o(\omega_o)}_{\text{(1) Radiance perçue}} = \underbrace{\int_{\Omega}}_{\text{(2) Ensemble des directions incidentes}} \underbrace{L_i(\omega_i, p)}_{\text{(3) Radiance incidente}} \cdot \underbrace{f_r(\omega_i, \omega_o)}_{\text{(4) BRDF}} \cdot \underbrace{(\omega_i \cdot n)}_{\text{(5) Atténuation géometrique}} \cdot \underbrace{d\omega_i}_{\text{Variable d'intégration}}$$

L'équation de rendu établit que la radiance réfléchie (1) par un point en direction d'un observateur correspond à l'intégrale (2) des radiances incidentes (3), multipliées par la réflectivité bidirectionnelle (4) de la surface et atténuées (5) par un facteur géométrique.

Elle peut être comprise assez simplement : la couleur perçue en un point dépend de **l'ensemble de la lumière reçue en ce point**, **d'une fonction caractérisant la surface** et de **l'orientation de la surface.**

1.2.2 – La radiance

La radiance représente un flux énergétique. Elle s'exprime en watt par mètre carré par stéradian ($W \cdot m^{-2} \cdot sr^{-1}$). C'est la puissance qui passe par unité de surface dans une direction donnée par unité d'angle solide.

On exprime souvent la radiance en un point p et le selon une direction ω :

$$L(p, \omega)$$

Un flux est sensible au sens. Ainsi :

$$L(p, \omega) = -L(p, -\omega)$$

D'une manière générale, les calculs évaluant la radiance le long d'un chemin sont équivalents dans un sens comme dans l'autre.

1.2.3 – La réflectivité bidirectionnelle ou BRDF

La réflectivité bidirectionnelle, ou BRDF, pour Bidirectionnal Reflectance Distribution Function, est une fonction qui caractérise la réflexion d'une surface. Dans l'équation de rendu, c'est elle qui définit les propriétés intrinsèques du matériau [2].

$$f_r(\omega_i, \omega_o)$$

avec :

f_r = BRDF

ω_i = direction incidente

ω_o = direction réfléchie

Figure 1.16 – BRDF : notation

Elle décrit la probabilité pour un photon incident de direction ω_i d'être réfléchi dans la direction ω_o. Autrement dit, elle décrit la portion d'énergie lumineuse transportée d'une direction ω_i vers une direction ω_o.

La BRDF est donc une fonction à deux paramètres. Pour la représenter graphiquement, on se place généralement dans la situation où le rayon incident est fixe. L'ensemble des rayons réfléchis forme un lobe. La longueur d'un rayon réfléchi est proportionnelle à sa probabilité d'occurrence.

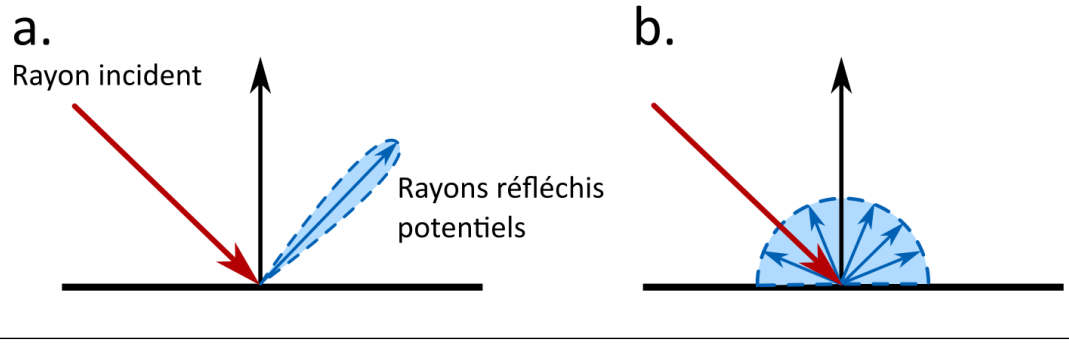


Figure 1.17 – Représentation graphique de la BRDF
Exemples d'une surface spéculaire (a.) et d'une surface diffuse (b.)

Il existe différents modèles mathématiques de BRDF [3][4][5]. Ceux-ci sont généralement paramétrisés par la rugosité et le caractère métallique du matériau.

1.2.4 – L'albédo

L'albédo décrit la fraction de radiance incidente réfléchie par une surface, le reste étant absorbé. C'est un facteur constant multiplicatif de la BRDF. En informatique graphique, en dehors du cas particulier du rendu spectral, l'albédo est une variable à trois dimensions (R, V et B) qui détermine la couleur perçue du matériau. Elle est généralement notée ρ.

1.2.5 – Les milieux participatifs

Les concepts de rendu physique réaliste que nous venons de présenter permettent de décrire un monde dans lequel les rayons lumineux parcourent le vide, en ligne droite et sans perte d'énergie, et rencontrent des surfaces avec lesquelles ils interagissent. Dans la réalité, qui n'est pas faite que de vide et de surfaces réfléchissantes, la lumière interagit également avec la matière qu'elle traverse.

Il est possible d'étendre les modèles de rendu de surface pour prendre en compte ce comportement.

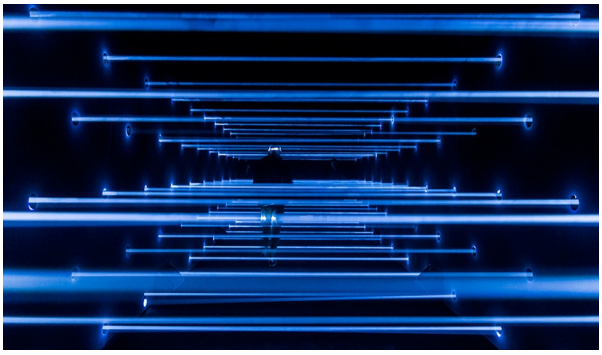


Figure 1.18 – Licht, mehr Licht!
Installation, Guillaume Marmin. L'interaction entre la lumière et le milieu rend visible les faisceaux.

On appelle **milieu participatif** un volume constitué de particules interagissant avec la lumière. L'air, l'eau, mais aussi la peau ou le plastique sont des milieux participatifs. On peut même considérer tous les matériaux comme des milieux participatifs, le modèle surfacique de la BRDF en étant une approximation acceptable pour les matériaux peu réfractifs, comme les métaux. Pour prendre en compte cette réalité, il convient de rajouter à notre modèle une interaction sur le trajet du rayon ω_o en direction de l'observateur.

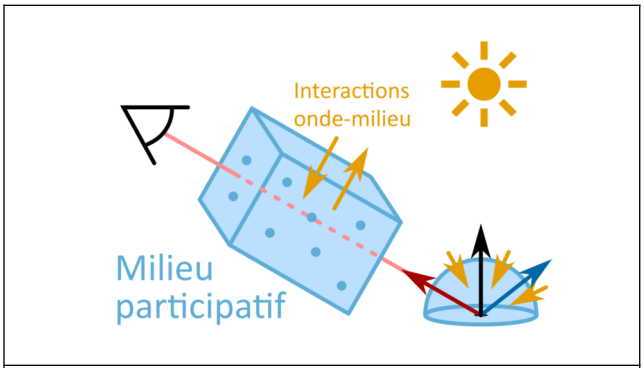


Figure 1.19 – Représentation d'un modèle physique surfacique et volumétrique

Nous allons voir par quels mécanismes les photons interagissent avec la matière au niveau atomique, quelles lois et paramètres macroscopiques décrivent ces mécanismes et comment ils s'intègrent dans le modèle du rendu PBR. La description d'un milieu participatif correspond à un champ tridimensionnel de ces paramètres.

1.2.6 – La diffusion des ondes

Lorsque la lumière traverse un milieu participatif, elle est susceptible d'interagir avec la matière selon quatre phénomènes :

- **L'absorption :** Les photons sont absorbés et leur énergie est transformée en chaleur.
- **La diffusion réfléchie ou out-scattering :** Des photons sont réfléchis dans différentes directions à l'intérieur du milieu et quittent le rayon.
- **La diffusion incidente ou in-scattering :** Des photons sont réfléchis par le milieu environnant et contribuent au rayon.
- **L'émission :** Des photons sont émis par la matière. C'est le cas notamment lorsqu'elle atteint de hautes températures.

On peut les représenter sous la forme d'une interaction ponctuelle le long d'un rayon lumineux.

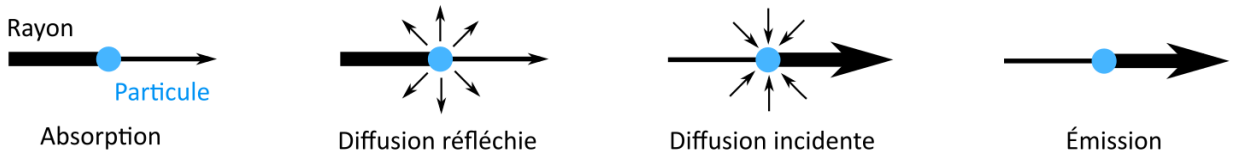


Figure 1.20 – Interaction lumière matière
L'épaisseur du trait est proportionnelle à la radiance du rayon.

Pour décrire les variations de luminance, on se place dans l'espace construit sur le rayon lumineux.

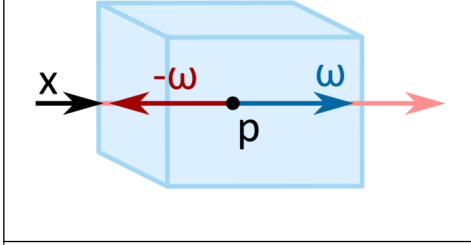


Figure 1.21 – Espace du rayon
La direction ω correspond à la propagation de la lumière.

On définit la variation de la luminance le long d'un rayon, en un point p et dans la direction de propagation de la lumière ω, $\frac{dL(p,\omega)}{dx}$, comme la somme des variations de luminance selon ces différents phénomènes :

$$\frac{dL(p,\omega)}{dx} = \frac{dL_{\text{out-scattering}}(p,\omega)}{dx} + \frac{dL_{\text{absorption}}(p,\omega)}{dx} + \frac{dL_{\text{in-scattering}}(p,\omega)}{dx} + \frac{dL_{\text{emission}}(p,\omega)}{dx}$$

On peut regrouper ces interactions deux à deux. Ainsi, un rayon qui traverse un milieu participatif :

- voit sa luminance **baisser** sous l'effet de l'**absorption** et de la **diffusion réfléchie**. On parle d'atténuation.
- voit sa luminance **augmenter** sous l'effet de la **diffusion incidente** et de l'**émission**.

1.2.7 – L'atténuation

L'atténuation désigne le processus selon lequel la radiance d'un rayon diminue. Elle se produit via les phénomènes d'absorption et de diffusion réfléchie. On définit généralement les coefficients d'absorption et de diffusion comme suit :

$$\frac{dL_{absorption}}{dx} = \sigma_a \cdot L_i \quad \text{et} \quad \frac{dL_{scattering}}{dx} = \sigma_s \cdot L_i$$

avec :

$\frac{dL_{absorption}}{dx}$ = variation de radiance le long d'un rayon, due à l'absorption

σ_a = coefficient d'absorption

L_i = radiance incidente

$\frac{dL_{scattering}}{dx}$ = variation de radiance le long d'un rayon, due à la diffusion réfléchie

σ_s = coefficient de diffusion

Figure 1.22 – Coefficients d'absorption et de diffusion
Les radiances et coefficients sont fonctions de la position p et de la direction ω. Nous ne l'exprimons pas ici par souci de synthèse.

Ces définitions peuvent être comprises ainsi :

- σ_a représente la propension du milieu à absorber l'énergie lumineuse. C'est la probabilité de d'absorption d'un photon par unité de distance.
- σ_s représente la propension du milieu à réfléchir l'énergie lumineuse. C'est la probabilité de déviation d'un photon par unité de distance.

La somme de ces deux coefficients est désignée coefficient d'atténuation, σ_t.

$$\sigma_t = \sigma_a + \sigma_s$$

Figure 1.23 – Coefficient d'atténuation : notation

On peut également définir l'albédo :

$$\rho = \frac{\sigma_s}{\sigma_t}$$

Figure 1.24 – Albédo : notation

De la même manière que dans le modèle surfacique, l'albédo représente la fraction de radiance réfléchie par le volume.

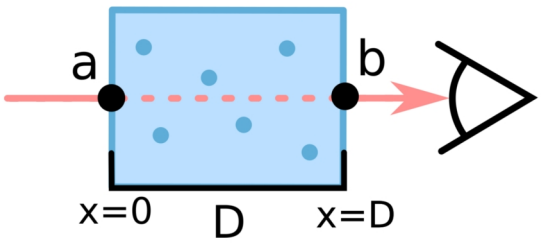
1.2.7.1 – Intégration et loi de Beer-Lambert

Le coefficient d'atténuation défini, on peut décrire l'atténuation globale de la radiance sous forme de l'équation différentielle :

$$\frac{dL_o(p,\omega)}{dx} = -\sigma_t(p,\omega) \cdot L_i(p,-\omega)$$

Figure 1.25 – Atténuation : équation différentielle

En pratique, on cherche à déterminer l'absorption sur un intervalle de matière. On appelle transmittance le rapport entre les radiances à l'entrée et à la sortie de l'intervalle. Elle nous permet de calculer la radiance perçue par l'observateur.



$$Tr(a \rightarrow b) = \frac{L(b)}{L(a)}$$

Figure 1.26 – Transmittance : notation et représentation graphique

La transmittance entre deux points s'obtient en intégrant l'équation différentielle de l'atténuation.

Les termes sont réorganisés. La direction ω, constante, est omise :

$$\frac{1}{L_i(p)} \cdot dL_o(p) = -\sigma_t(p) \cdot dx$$

On intègre de chaque côté, sur l'intervalle [a, b] :

$$\int_{p=a}^{b} \frac{1}{L_i(p)} \cdot dL_o(p) = \int_{p=a}^{b} -\sigma_t(p) \cdot dx$$

$$\ln\left(\frac{L(b)}{L(a)}\right) = -\int_{p=a}^{b} \sigma_t(p) \cdot dx$$

$$Tr(a \rightarrow b) = \frac{L(b)}{L(a)} = e^{-\int_{p=a}^{b} \sigma_t(p) \cdot dx}$$

Figure 1.27 – Transmittance : intégration

Dans le cas d'un milieu homogène, le coefficient d'atténuation σ_t est constant et la transmittance s'évalue analytiquement. La relation qui en résulte est connue sous le nom de loi de Beer-Lambert.

$$Tr(a \rightarrow b) = e^{-\int_{p=a}^{b} \sigma_t(p) \cdot dx}$$

$$Tr(a \rightarrow b) = e^{-\sigma_t \cdot (b-a)}$$

Figure 1.28 – Transmittance : Loi de Beer-Lambert

En informatique graphique, une des applications les plus simples de la loi de Beer-Lambert est le brouillard de distance (voir 2.1.1).

Dans le cas d'un milieu hétérogène, la transmittance doit être évaluée par échantillonnage. Elle reflète un rapport de radiance entre deux points. Ainsi, elle est toujours comprise entre 0 et 1, et :

$$Tr(a \rightarrow b) = Tr(a \rightarrow a_1) \cdot Tr(a_1 \rightarrow b)$$

Plus généralement :

$$Tr(a \rightarrow b) = Tr(a \rightarrow a_1) \cdot Tr(a_1 \rightarrow a_2) \cdot \ldots \cdot Tr(a_n \rightarrow b)$$

Si n est grand, la distance D entre deux points adjacents, est faible : on peut considérer le milieu comme localement homogène et donc l'évaluer analytiquement.

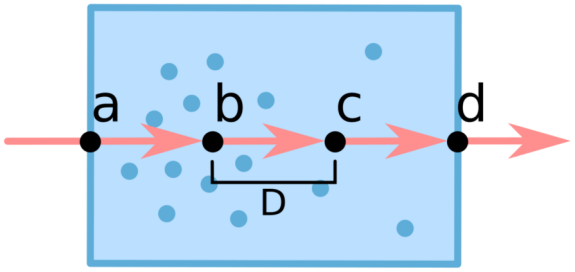


Figure 1.29 – Milieu hétérogène : échantillonnage de la transmittance
$Tr(a \rightarrow d)$ *est estimé par*
$Tr(a \rightarrow b) \cdot Tr(b \rightarrow c) \cdot Tr(c \rightarrow d)$ *(n = 3).*
Plus D est petit, plus n est grand et meilleure est l'estimation.

1.2.8 – La diffusion incidente

Nous avons décrit l'atténuation. Seule, elle rend compte de la densité du volume dans l'axe d'observation, mais ne reflète pas la complexité du trajet lumineux à travers la matière.

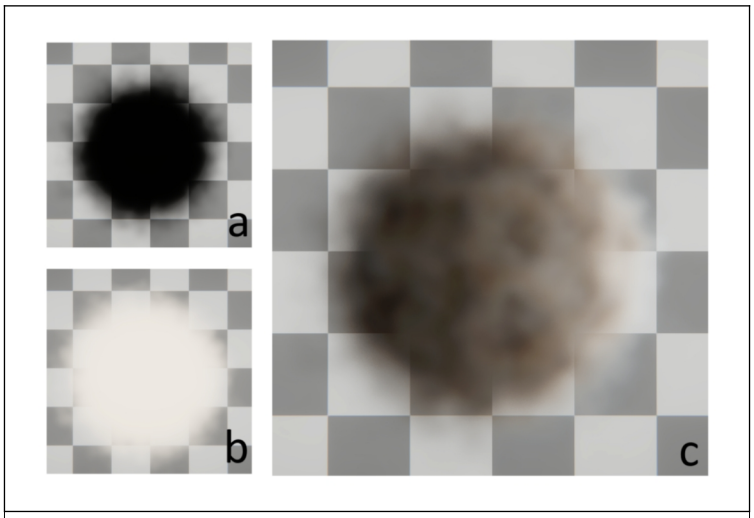


Figure 1.30 – Milieu participatif hétérogène
Absence de diffusion incidente : atténuation pure (a.), émission et atténuation (b.)
Présence de diffusion incidente (c.).
Images obtenues avec la méthode présentée en 3.2.

La diffusion incidente peut être comprise comme la réflexion d'un rayon lumineux sur une particule du milieu participatif.

On peut décrire la diffusion incidente comme suit :

$$L_s = \sigma_s \cdot \int_{\Omega} ph(\omega_i, \omega_o) \cdot L_i \cdot d\omega_i$$

avec :

L_s = variation de radiance le long du rayon d'observation, due à la diffusion incidente

σ_s = coefficient de diffusion

Ω = domaine d'intégration : ensemble des directions

ph = fonction de phase du milieu participatif

L_i = radiance incidente dans la direction ω_i

ω_i = direction incidente, variable d'intégration

ω_o = direction d'observation

Figure 1.31 – Diffusion incidente : définition physique

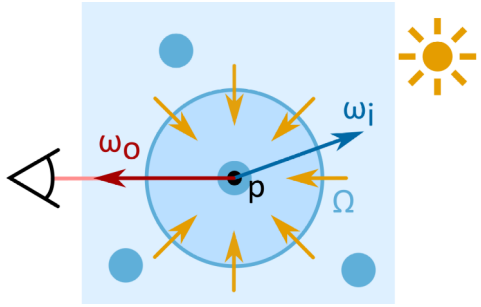


Figure 1.32 – Diffusion incidente : représentation graphique

$$\underbrace{L_s}_{\substack{\text{(1) Variation de radiance par diffusion incidente}}} = \underbrace{\sigma_s}_{\substack{\text{(2) Coefficient de diffusion}}} \cdot \underbrace{\int_{\Omega}}_{\substack{\text{(3) Ensemble des directions incidentes}}} \underbrace{ph(\omega_i, \omega_o)}_{\substack{\text{(4) Fonction de phase}}} \cdot \underbrace{L_i}_{\substack{\text{(5) Radiance incidente}}} \cdot d\omega_i$$

Cette égalité peut être comprise de manière similaire à l'équation de rendu présentée dans le cadre surfacique (voir 1.2.1). Elle établit que la **diffusion incidente (1)** dépend de **la probabilité de diffusion, représentée par le coefficient de diffusion (2), d'une fonction caractérisant le milieu participatif (4) et de la radiance incidente (5).**

1.2.9 – La fonction de phase

Dans le rendu surfacique, la réflexion de la lumière est caractérisée par la réflectivité bidirectionnelle du matériau (voir 1.2.3). De manière analogue, dans le rendu volumétrique, la diffusion de la lumière est caractérisée par la fonction de phase du milieu participatif.

$$ph(\omega_i, \omega_o)$$

avec :

ph = fonction de phase

ω_i = direction incidente

ω_o = direction réfléchie

Figure 1.33 – Fonction de phase : notation

La fonction de phase a la même signification que la BRDF : elle décrit la probabilité pour un photon incident de direction ω_i d'être réfléchi dans la direction ω_o. Autrement dit, elle décrit la portion d'énergie lumineuse transportée d'une direction ω_i vers une direction ω_o.

On peut représenter la fonction de phase dans un repère construit autour du rayon incident. Comme pour la BRDF, l'ensemble des rayons réfléchis forme un lobe.

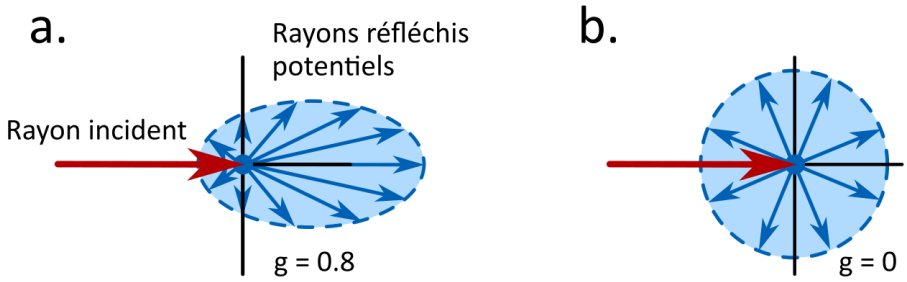


Figure 1.34 – Fonction de phase : représentation graphique
En a., la diffusion prédomine vers l'avant (antédiffusion ou forward scattering). En b, la diffusion est équiprobable dans toutes les directions : la fonction de phase est isotropique.

Comme pour la BRDF, il existe plusieurs modèles mathématiques de fonctions de phase. Le plus utilisé est celui d'Henyey-Greenstein [7]. Il est paramétrisé par g (antédiffusion si $g > 0$, rétrodiffusion si $g < 0$, isotropisme si $g = 0$).

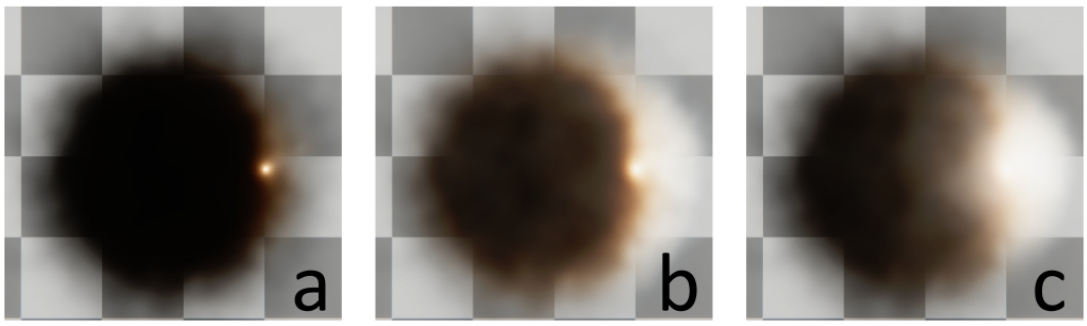


Figure 1.35 – Fonction de phase : rendus
Une source lumineuse est placée en arrière du volume.
a. Rétrodiffusion (g = -0.8). Peu de lumière parvient à l'observateur.
b. Diffusion isotropique (g = 0). La lumière est diffusée dans tout le volume.
c. Antédiffusion (g = 0.8). La lumière est peu déviée.
Images obtenues avec la méthode présentée en 3.2.

1.2.10 – La diffusion incidente : intégration

En pratique, on cherche à estimer la radiance due à la diffusion incidente transmise vers l'observateur. Elle correspond à la radiance par diffusion incidente telle que présentée en 1.2.8, multipliée par la transmittance et intégrée le long du rayon d'observation.

$$(1)\quad L_o = \int_{x=0}^{D} L_s \cdot Tr(0 \rightarrow x) \cdot dx$$

Comme pour la transmission, on peut estimer cette intégrale par échantillonnage, en considérant L_s comme constant localement.

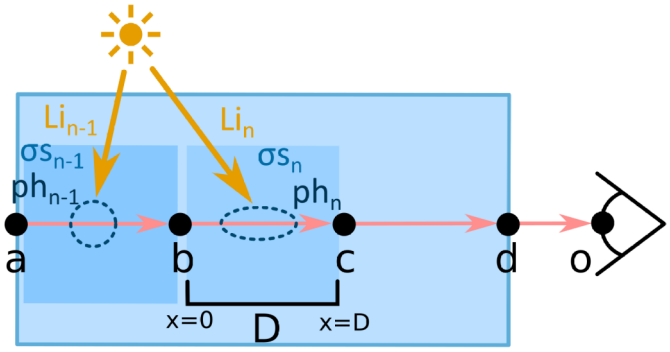


Figure 1.36 – Diffusion incidence : échantillonnage
Les 3 paramètres intervenant dans le calcul de la diffusion incidente Ls (1.2.8) sont considérés comme constants sur chaque intervalle.
On estime $L_o(a \rightarrow v)$ *par :*

$$L_o(a \rightarrow b) \cdot Tr(b \rightarrow v) + L_o(b \rightarrow c) \cdot Tr(c \rightarrow v) + L_o(c \rightarrow d) \cdot Tr(d \rightarrow v)$$

Sur chaque intervalle de paramètres constants et d'épaisseur D, l'équation (1) admet alors une solution [10] :

$$L_o = \int_{x=0}^{D} L_s \cdot Tr(0 \rightarrow x) \cdot dx$$

$$L_o = \int_{x=0}^{D} L_s \cdot e^{-\sigma_t x} \cdot dx$$

$$L_o = \frac{L_s - L_s \cdot Tr(0 \rightarrow D)}{\sigma_t}$$

Figure 1.37 – Intégration de la radiance par diffusion incidente sur un intervalle d'épaisseur D

1.2.11 – L'émission

La modélisation des volumes émissifs n'est pas développée. Elle est triviale : un paramètre décrivant l'intensité de l'émission est ajouté au terme L_s.

1.2.12 – Modèle physique : conclusion

Nous avons présenté le modèle physique dans lequel un objet volumétrique est représenté par un champ tridimensionnel de paramètres : les coefficients de diffusion et d'absorption, la fonction de phase.

Pour modéliser le comportement de la lumière, nous avons établi des relations entre ces paramètres sous la forme d'équations intégro-différentielles définies dans l'espace d'un rayon traversant le volume.

Dans la majorité des cas, l'hétérogénéité du champ de paramètres rend impossible la résolution analytique de ces équations. Nous avons présenté une solution numérique fondée sur l'échantillonnage.

Les algorithmes présentés dans les parties 2 et 3 de ce mémoire reposent sur ces principes.

1.3 – Principe de fonctionnement d’un moteur de rendu et cas du temps réel

Pour comprendre le défi que représente le rendu d’objets semi-transparents, il est nécessaire d’évoquer le fonctionnement général d’un moteur de rendu 3D.

Fondamentalement, sa fonction est la suivante : générer une représentation en deux dimensions, l'image, à partir d'une représentation en trois dimensions d'une collection d'objets, la scène. Un objet est, typiquement, représenté par un ensemble de triangles.

Note préliminaire : pixel / texel / voxel

Le pixel correspond à l’unité de base d’un écran, ou d’une image. Cette confusion n’existe pas lorsqu’on utilise le terme texel, qui désigne toujours le plus petit élément d’une texture, c’est à dire d’une image en mémoire. Pas souci de synthèse, nous parlerons dans cette partie de pixels, les concepts étant équivalents dans le cas du rendu direct sur un écran. Le voxel, abordé dans la partie 2, est un texel en trois dimensions, et fait toujours référence à l’élément de la texture.

1.3.1 – Exemple du ray tracing

Un algorithme naïf de rendu 3D est celui du ray tracing. Nous allons dans un premier temps le présenter en tant que méthode de référence pour illustrer, à l'opposé, les techniques de rendu propres au temps réel et les limites qu'elles impliquent.

```
Color[] trace_scene(camera, triangles)
{
    Color[] pixels;

    for (each pixel)
    {
        Ray camera_ray = compute_camera_ray(pixel, camera);
        float closest_distance = infinite;
        Hit closest_hit = null;

        for (each triangle)
        {
            float distance;
            Hit hit;
            bool has_hit = false;
            intersect(camera_ray, triangle, &distance, &has_hit, &hit);
            if (has_hit && distance < closestDistance)
            {
                closest_hit = hit;
            }
            pixel_color = black
        }

        if (closest_hit != null)
        {
            pixel = shade(closest_hit);
        }
    }

    return pixels;
}
```

Figure 1.38 – Algorithme de ray tracing (pseudo-code)
entrée : *description de la scène (triangles, caméra)*
sortie : *couleur du pixel*

L'algorithme de ray tracing peut être qualifié de pixel-centré. Il itère sur chaque pixel de l'écran puis sur chaque objet de la scène.

Par sa proximité avec le comportement réel de la lumière, un algorithme de ray tracing est particulièrement versatile : il est simple d'y implémenter une grande gamme d'effets physiques en traçant un nombre plus important d'intersections. En voici quelques exemples :

- Générer plusieurs rayons primaires par pixel permet de produire un filtre antialiasing.

- Générer des rayons secondaires au point d'intersection permet d'évaluer les ombres ou les réflexions.
- La transparence d'une surface peut être gérée de manière stochastique.
- La profondeur de champ peut être reproduite en modifiant l'orientation des rayons primaires, comme à travers une lentille.

1.3.2 – La rastérisation

Pour atteindre des fréquences de rafraîchissement permettant l'interaction, la 3D en temps réel se base sur des techniques de rastérisation.

L'algorithme est cette fois-ci objet-centré. Il itère sur les objets puis sur les pixels. Ce design permet une première optimisation majeure : un grand nombre de pixels ne sont pas évalués. Il s'agit de ceux se trouvant hors du rectangle englobant le triangle (*bounding box*).

Les cartes graphiques sont spécialement conçues pour réaliser de manière parallèle les opérations nécessaires au pipeline de rastérisation (transformations, test d'inclusion).

On appelle fragment l'unité du triangle rastérisé en un pixel.

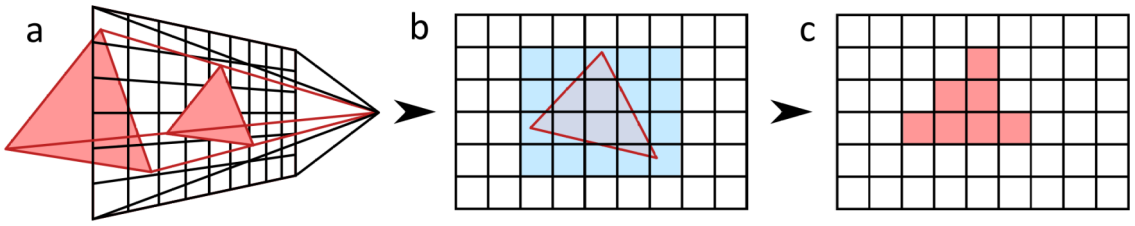


Figure 1.39 – Rastérisation
a. Transformation : projection d'un triangle sur le plan de la caméra
b. Seule la zone bleue est évaluée
c. Un fragment est généré pour chaque pixel dont le centre est situé dans le triangle.

```
Color[] rasterize_scene(camera, triangles)
{
    Color[] pixels ;

    for (each triangle)
    {
        float2 vertex_0 = project_to_camera_plane(vertex_0);
        float2 vertex_1 = project_to_camera_plane(vertex_1);
        float2 vertex_2 = project_to_camera_plane(vertex_2);

        float4 triangle_bounding_box =
            compute_bounding_box(vertex_0, vertex_1, vertex_2)

        for (each pixel within triangle_bounding_box)
        {
            if (is_in_triangle(pixel, vertex_0, vertex_1, vertex_2))
            {
                pixel = shade();
            }
        }
    }

    return pixels;
}
```

Figure 1.40 – Algorithme de rastérisation (pseudo-code)
entrée : *description de la scène (triangles, caméra)*
sortie : *couleur du pixel*

1.3.2.1 – Gestion de la profondeur

Par profondeur, on désigne la composante vecteur caméra « avant » du vecteur caméra-objet.

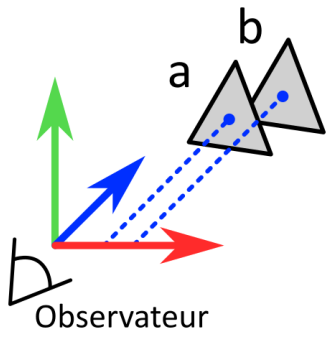


Figure 1.41 – Profondeur
La profondeur de l'objet correspond à sa coordonnée avant (en bleu) dans l'espace de la caméra. La profondeur du triangle a. est inférieure à celle du triangle b. : le triangle a. occlut le triangle b.

Dans le cas du ray tracing, la profondeur est évaluée à chaque test d'intersection, et seule l'intersection la plus proche sera préservée.

La nature objet-centrée de la rastérisation rend impossible cette comparaison directe. L'organisation des objets se fait via une mémoire tampon dédiée : le depth buffer. Quand un fragment est rastérisé en une coordonnée précise de l'écran, sa profondeur est comparée à la valeur du depth buffer à cette coordonnée, et le fragment est rejeté ou non. S'il est conservé, il écrit sa profondeur dans le buffer. Généralement, le test utilisé est :

```
bool write_fragment = fragment_depth < depth_buffer[pixel_coords]
```

Figure 1.42 – Rendu opaque : depth test

1.3.3 – Le problème de la transparence

La méthode du depth buffer est très efficace car elle permet de traiter les objets de manière parallèle et dans un ordre quelconque. Cependant, elle est tout à fait inadaptée aux objets semi-transparents. Nous allons présenter ici le cas général des surfaces. La partie 2 est consacrée au cas des volumes.

L'occlusion générée par un objet semi-transparent sur ce qui se trouve au-delà de lui-même est partielle et dépend de son opacité.

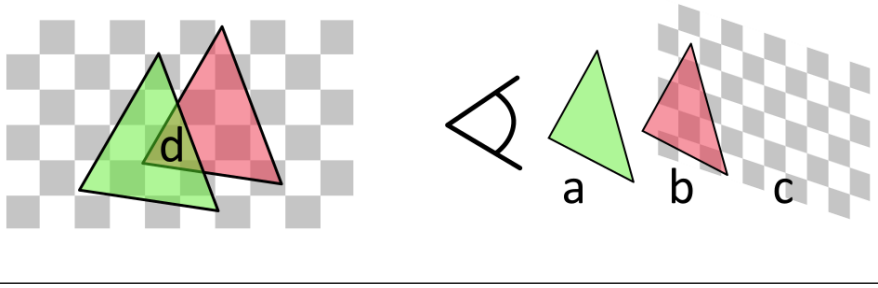


Figure 1.43 – Fusion
Dans la zone centrale d, la couleur de l'objet c. est atténuée par celles de l'objet b. puis de l'objet a.

On calcule la combinaison d'une surface transparente avec son arrière-plan en utilisant son opacité (ou composante alpha de la couleur) comme paramètre d'interpolation entre sa teinte et la teinte se trouvant au-delà de la surface, dans l'espace caméra. On parle d'*alpha blending*.

La fusion d'une surface A semi-transparente avec son arrière plan de couleur B s'évalue comme suit :

$$c = c_A \cdot a_A + c_B \cdot (1 - a_A)$$

avec :

a_A = opacité de la surface A

c_A = couleur de la surface A

c_B = couleur de l'arrière plan B

Figure 1.44 – Mode de fusion : alpha blend

Pour déterminer la couleur d'un pixel dans lequel sont rastérisés de multiples fragments transparents, il convient d'appliquer ce mode de fusion dans un ordre précis, du fragment le plus éloigné au fragment le plus proche. Ce type de situation nous incite à trier les objets et à évaluer l'ensemble des fragments. Ces opérations peuvent vite impacter les performances d'un moteur temps réel. Par ailleurs, en l'absence d'écriture dans le *depth buffer*, l'information de profondeur est perdue, et avec elle certaines possibilités de post-traitement, comme la profondeur de champ.

1.3.4 – Rendu deferred

Le rendu *deferred* (différé) consiste, par opposition au rendu *forward* (direct), à réaliser les calculs d'éclairage après avoir évalué une première fois tous les fragments. Seuls ceux effectivement visibles sont traités, à un coût fixe dépendant de la résolution de l'écran. Cette technique repose sur des textures intermédiaires dont le *depth buffer :* elle est inaccessible aux objets non opaques.

1.3.5 – Conclusion

Le problème de la transparence est, par design, inhérent aux pipelines de rastérisation et ce même pour le cas simple des surfaces. Des solutions existent mais impliquent toujours des compromis en termes de performance ou d'apparence.

2 – Stratégies

Cette partie illustre une sélection d'algorithmes de rendu volumétrique. On choisit de séparer ces techniques en trois catégories : les méthodes analytiques, limitées aux effets atmosphériques, celles basées sur des images pré-calculées et enfin celles reposant sur une représentation tridimensionnelle discrète.

2.1 – Méthodes analytiques, effets à grande échelle

En faisant certaines simplifications sur la nature du milieu participatif, il est possible d'évaluer de façon analytique sa contribution le long du rayon d'observation. Ce type de stratégie est ancien mais reste très utilisé : il est peu coûteux en terme computationnel tout en étant visuellement efficace.

2.1.1 – Distance fog / Brouillard de distance

On considère l'atmosphère comme homogène. On peut ainsi calculer la transmittance entre la surface d'un objet et la caméra. Le brouillard de distance, (en anglais *distance fog* ou parfois *depth fog*) est l'application la plus simple de la loi de Beer-Lambert (1.2.8).

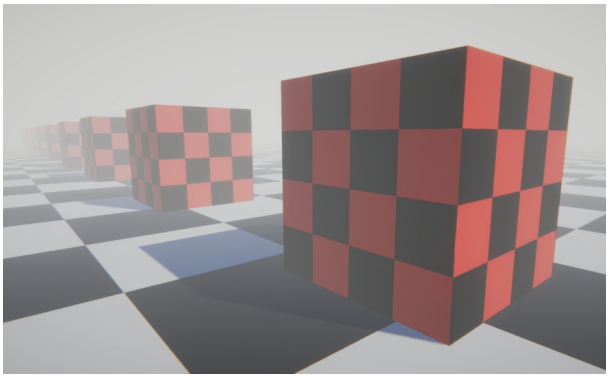


Figure 2.1 – Distance fog
Implémentation dans Unity

Dans le fragment shader, on évalue la transmittance et donc l'atténuation à appliquer à la couleur de la surface. Cette stratégie s'implémente facilement en *forward* comme en *deferred rendering*. La distance objet-observateur doit être accessible. Typiquement, en *deferred rendering*, on obtient la position de la surface à partir du *depth buffer* (voir 2.1.3 – notions géométriques).

```
float3 apply_distance_fog(  float3 surface_color,
                            float distance_to_surface,
                            float3 fog_color,
                            float fog_sigmaT )
{
    float transmittance = exp(-fog_sigmaT * distance_to_surface );
    return lerp(fog_color, surface_color, transmittance);
}
```

Figure 2.2 – Distance fog : algorithme (HLSL)
entrées : *couleur de la surface d'arrière-plan, distance surface-caméra, couleur du brouillard, coefficient d'atténuation (ou densité)*
sortie : *couleur de la surface atténuée par le brouillard*

2.1.2 – Exponential height fog / Brouillard de hauteur

Il est également possible d'évaluer analytiquement la transmittance dans le cas d'un milieu non plus homogène, mais de densité exponentiellement décroissante avec l'altitude.

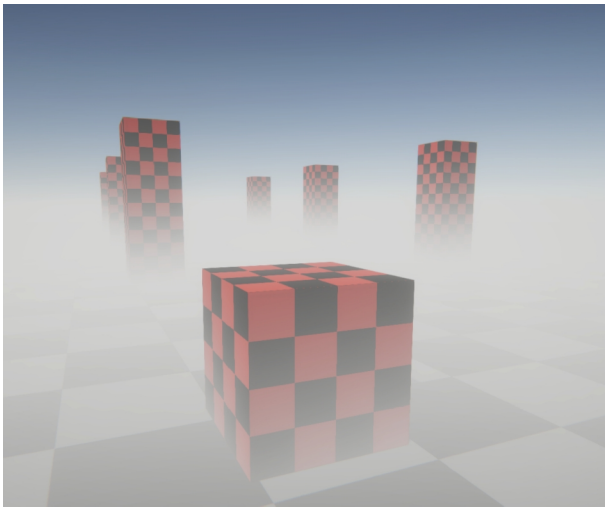


Figure 2.3 – Exponential height fog
Implémentation dans Unity

Cette solution apporte plusieurs améliorations notables au brouillard de distance simple. Elle permet de reproduire le comportement d'une brume s'accumulant près du sol. De plus, elle ne masque pas totalement l'environnement à l'infini (*skybox*) et produit avec celui-ci une transition douce au niveau de l'horizon.

$$(1)\quad Tr(c \rightarrow s) = e^{-\int_{p=c}^{s} \sigma_t(p) \cdot dx}$$

avec :

$$\sigma_t(p) = k \cdot e^{-p.y \cdot h}$$

et :

Tr	=	transmittance de la camera c à la surface s
σ_t	=	coefficient d'atténuation / densité
h	=	contrôle de la décroissance exponentielle
k	=	facteur multiplicatif global de σ_t
$p.y$	=	hauteur de caméra

Figure 2.4 – Exponential height fog : définition mathématique
Le coefficient d'atténuation décroît avec l'altitude. L'intégration analytique reste possible.

Wenzel [6] propose une solution à l'équation (1). Nous en présentons directement l'intégration :

```
float3 apply_exponential_height_fog(    float3 surface_color,
                                        float3 camera_position,
                                        float3 surface_position,
                                        float3 fog_color,
                                        float fog_height_falloff,
                                        float fog_sigmaT )
{
    float3 camera_to_surface = surface_position - camera_position;
    float distance_to_surface = length(camera_to_surface);
    float density_at_view = exp(- fog_height_falloff * camera_position.y);
    float fog_integral = distance_to_surface * density_at_view;
    float t = (heightFalloff * camera_to_surface.y);
    fog_integral *= (1.0 - exp(-t)) / t;

    float transmittance = exp(-fog_sigmaT * fog_integral);

    return lerp(fog_color, surface_color, transmittance);
}
```

Figure 2.5 – Exponential height fog : algorithme (HLSL)
entrées : *couleur de la surface, positions de la caméra et de la surface, couleur du brouillard, paramètre de décroissante exponentielle, coefficient d'atténuation*
sortie : *couleur de la surface atténuée par le brouillard*

Cette solution nécessite d'avoir accès à la position de la surface. Comme pour la distance caméra-surface, celle-ci peut être reconstruite à partir du *depth buffer* (voir 2.1.3 – notions géométriques impliquées).

2.2.6 – Notions géométriques

Généralement, une implémentation de brouillard de distance utilise la projection inverse pour reconstruire la distance surface-caméra dans un *pass* écran dédié (ou effet de *post-processing screen space*).

```
// In vertex shader
float4 far_plane_position_vs =
    mul(inverse_projection, float4(clip_coordinates.xy, 1, 1));
far_plane_position_vs.xyz /=  far_plane_position_vs.w;
vertex_output.far_plane_position_vs = view_space_position.xyz;

// In fragment shader
float depth = tex2D(camera_depth_texture, screen_uv);
depth = to_linear_depth(depth);
float distance_to_surface = depth * length(fragment_input.far_plane_position_vs);
```

Figure 2.6 – Reconstruction de la distance caméra-surface dans un pass de post-processing
La position du plan de clipping distant est calculée en espace caméra (view space «_vs»).
La fonction to_linear_depth() linéarise la valeur du depth buffer (0 : camera, 1 : plan de clipping distant). Son détail est implémentation-dépendant.

Avec une logique similaire, on peut également reconstruire la position de la surface :

```
// In vertex shader
float4 far_plane_position_vs =
    mul(inverse_projection, float4(clip_coordinates.xy, 1, 1));
far_plane_position_vs.xyz /=  far_plane_position_vs.w;
vertex_output.camera_to_far_plane =
    mul(inverse_view, float4(far_plane_position_vs.xyz, 0)).xyz;

// In fragment shader
float depth = tex2D(camera_depth_texture, screen_uv);
depth = to_linear_depth(depth);
float3 surface_position = camera_position + depth * fragment_input.camera_to_far_plane;
```

Figure 2.7 – Reconstruction de la position de la surface dans un pass de post-processing
Dans le vertex shader, le vecteur allant de la camera au coin du plan de clipping distant est reconstruit.
Dans le fragment shader, ce vecteur est multiplié par la depth linéaire pour obtenir le vecteur allant de la camera à la surface la plus proche.

2.1.4 – Couleur du brouillard

Les techniques de *distance/exponential height fog* font intervenir une couleur constante. Elle correspond à une radiance incidente uniforme en tout point du rayon d'observation, ou à un volume émissif.

Une solution pour estimer la radiance transmise le long du rayon d'observation est d'associer **une contribution directe** faisant intervenir une source lumineuse à l'infini ainsi que la fonction de phase du milieu participatif à **une contribution indirecte** construite à partir de l'environnement (ciel procédural, *environment map*).

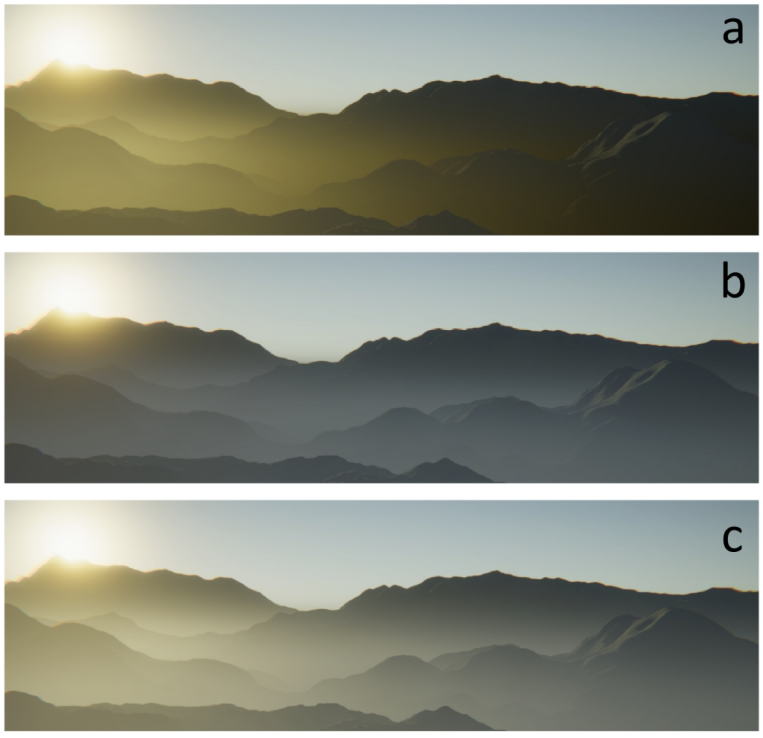


Figure 2.8 – Exponential height fog : couleur
Implémentation dans Unity
a. Contribution directe. On remarque la prédominance d'antédiffusion (forward scattering). g = 0.85
b. Contribution indirecte : couleur unie obtenue par échantillonnage du ciel.
c. Combinaison des deux contributions.

2.2 – Textures bidimensionnelles, effets localisés

A l'opposé des méthodes analytiques limitées aux effets atmosphériques à grande échelle, l'utilisation de textures 2D permet de représenter des objets volumétriques localisés avec un grand contrôle dans le rendu comme dans le comportement. Ce type de stratégie repose parfois plus sur la validation artistique que sur la théorie physique. En 2020, il s'agit encore de la méthode principale d'affichage d'effets de *gameplay*.

2.2.1 – Splatting

Le *splatting* (éclaboussure en français) consiste à accumuler la contribution d'objets semi-transparents par dessus le reste de la scène opaque. On utilise des *billboards :* des plans, souvent texturés, orientés en direction de la caméra. Ceux-ci sont triés et rendus dans l'ordre allant du plus distant au plus proche par rapport à l'observateur (voir 1.3.3).

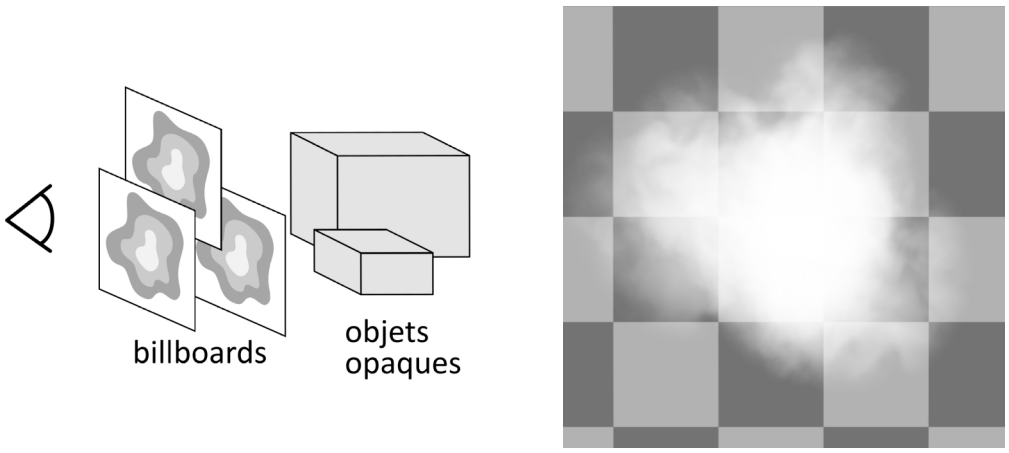


Figure 2.9 – Splatting
Principe (gauche)
Image rendue avec Vulkan (droite).
Le volume possède une couleur constante (pas de calcul d'éclairage).

2.2.2 – Éclairage : utilisation de textures

Il est souvent préférable de reproduire le comportement de la lumière à travers le volume.

La solution la plus simple est d'utiliser une texture RGB d'un rendu au préalable éclairé. Cette méthode est limitée au effets statiques dont le comportement et la position sont connues (cinématiques, objets distants).

Pour obtenir un rendu dynamique, on peut traiter le volume comme une surface, à l'aide d'une texture de normales et d'un modèle de BRDF. Cette technique s'éloigne de la réalité physique : un milieu participatif n'a ni surface, ni normale. On peut cependant obtenir des résultats acceptables en

assimilant cette dernière à une texture RGB représentant le gradient de densité du volume rendu. Le gradient correspond à la direction vers laquelle le volume augmente le plus en densité.

Il est également possible d'utiliser une combinaison de rendus précalculés, éclairés sous différentes incidences. Ces rendus sont rassemblés dans une ou plusieurs textures. On parle de *lightmaps.* Les *lightmaps* sont interpolées selon la direction de la lumière à l'intérieur du fragment *shader*.

Il existe une variété d'algorithmes modélisant l'éclairage par diffusion incidente[12][13]. Macklin[14] propose une solution pour intégrer analytiquement celle-ci depuis une lumière ponctuelle et dans un milieu homogène sur un intervalle. Celle-ci ne prend pas en compte l'atténuation le long du rayon d'observation ni entre la source lumineuse et le point de diffusion. Le milieu participatif ne produit par conséquent pas d'ombre sur lui-même. Le détail de l'intégration est disponible sur la page web de l'auteur.

Macklin propose notamment d'évaluer la diffusion dans le *fragment shader* de *billboards.* Ceux-ci sont considérés comme ayant une certaine épaisseur. La densité est échantillonnée depuis la texture, et donc variable entre les fragments.

Nous proposons une variante de cette stratégie. Les *billboards* sont rendus dans un *pass* séparé. La densité y est accumulée. Les distances fragment-caméra maximales et minimale sont également écrites. Un *pass* écran est dédié à l'évaluation de la diffusion incidente ainsi qu'à la fusion avec le reste de la scène. Cette technique permet de ne pas trier les *billboards* et d'évaluer l'éclairage moins souvent (une seule fois par pixel, même si plusieurs *billboards* s'y superposent).

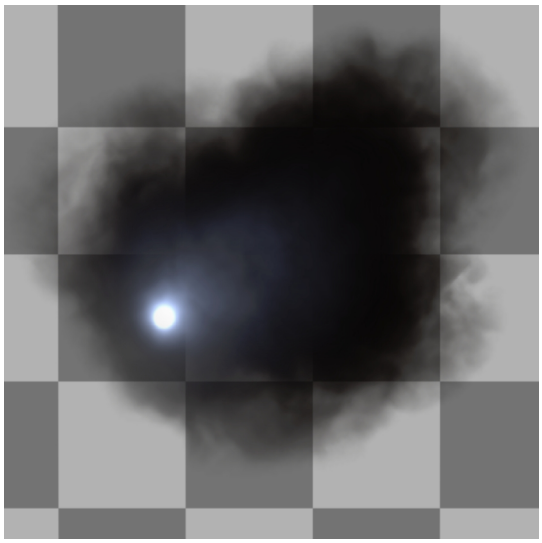


Figure 2.10 – Diffusion incidente, lumière ponctuelle
Implémentation en Vulkan.

Dans la partie 3.1, nous présentons une technique basée également sur *pass* dédié. L'implémentation y est davantage détaillée.

2.2.3 – Éclairage : ombrage volumétrique

Les techniques présentées en 2.2.2 ne prennent pas en compte l'atténuation de la lumière entre une source et un point de diffusion sur le rayon d'observation. C'est ce phénomène qui est responsable des ombres du milieu participatif sur lui-même.

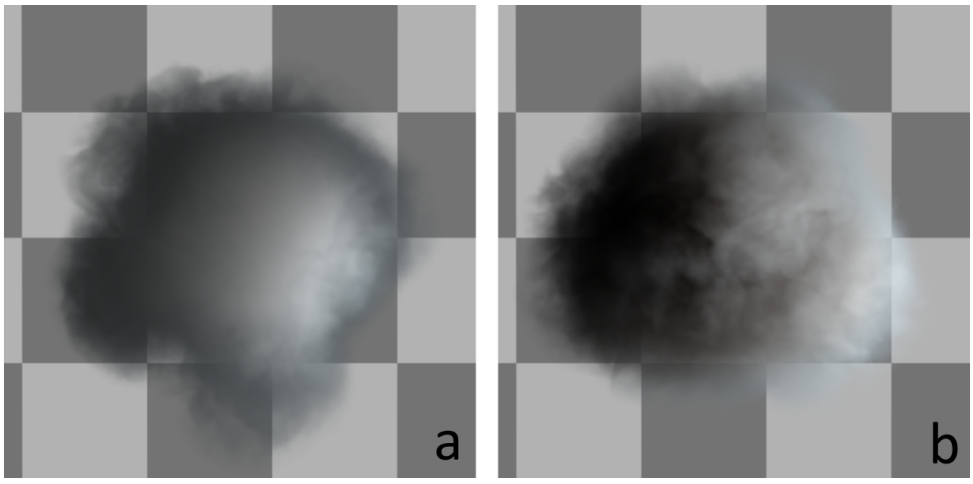


Figure 2.11 – Billboards sans (a) et avec (b) ombrage volumétrique
Rendus avec la méthode présentée en partie 3.1.

Dans le rendu surfacique, l'évaluation des ombres passe par la génération de *shadow maps.* Une *shadow map* est une texture 2D correspondant à un rendu du point de vue d'une source lumineuse. Seule l'information de profondeur y est enregistrée. Celle-ci est utilisée pour déterminer si un fragment est dans l'ombre ou la lumière.

Cette stratégie est, par design, inadaptée aux objets semi-transparents qui n'écrivent pas dans le *depth buffer* (voir 1.3.3).

Lokovic et Veach [15] introduisent le concept de *deep shadow map*. Chaque texel contient une description de l'atténuation de la lumière depuis sa source. Cette texture est évaluée pour déterminer la transmittance entre un point de la scène et la source lumineuse et donc, la radiance atténuée.

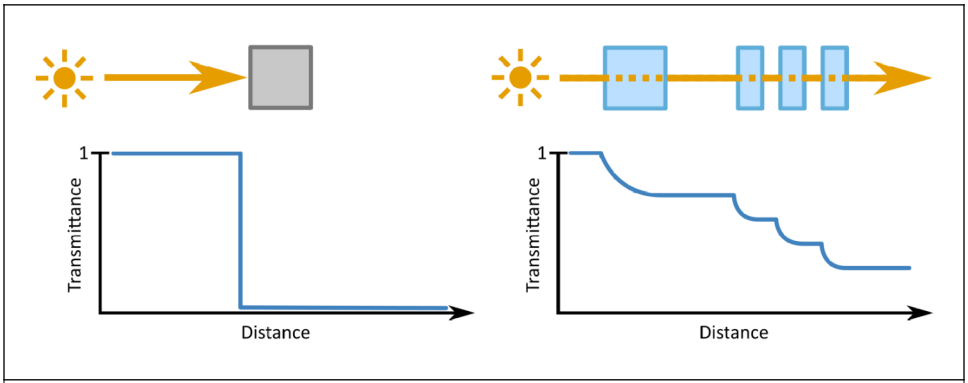


Figure 2.12 – Fonctions d'atténuation
Objet opaque (à gauche) et objets transparents (à droite).

La fonction d'atténuation est construite en combinant l'opacité de plusieurs rendus (depuis la lumière), produits à des profondeurs différentes. La principale difficulté de ce type de stratégie est de construire et d'encoder cette fonction en temps réel. Une grande variété de publications proposent de résoudre ce problème [16][17]. L'Unreal Engine 4 implémente un encodage basé sur les coefficients de Fourier [16]. Cette représentation est particulièrement efficace dans le cas d'objets diffus peu denses mais ne permet pas de représenter les détails à haute fréquence.

2.3 – Description tridimensionnelle discrète

Les textures 2D offrent un contrôle artistique important. Cependant, elles s'intègrent mal dans la théorie physique et dépendent parfois de processus de création de ressources rigides. De plus, la modélisation des phénomènes complexes de diffusion ou d'atténuation repose sur des algorithmes spécifiques et vite coûteux en termes computationnels. Avec le développement des GPU, une solution unifiant les effets atmosphériques et les effets localisés devient envisageable.

L'idée générale est d'évaluer une représentation du milieu participatif ambiant, hétérogène. Une texture tridimensionnelle enregistre une description discrète de ses paramètres.

2.3.1 – Raymarching

Le *raymarching* désigne l'échantillonnage d'une description de volume 3D (généralement une texture) le long de rayons d'observation.

L'atténuation et la radiance incidente sont intégrées numériquement selon les principes présentés respectivement en 1.2.7.1 et en 1.2.10.

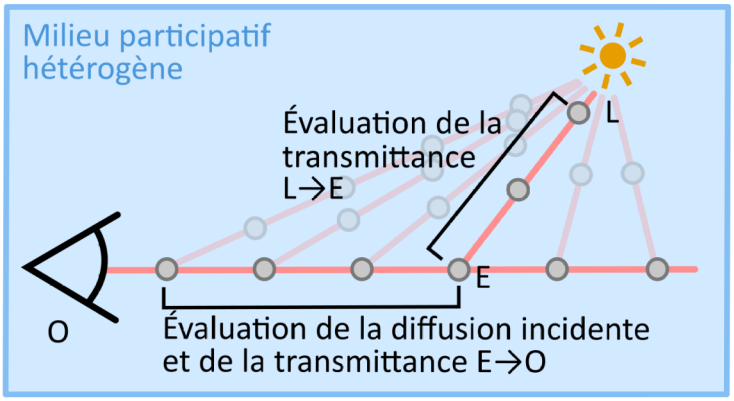


Figure 2.13 – Raymarching : principe de l'algorithme
Le volume est échantillonné le long du rayon d'observation et entre chaque échantillon primaire et la source lumineuse.
La transmittance L⟶E nous permet de calculer la radiance incidente en E. La transmittance E⟶O nous permet de calculer la contribution de l'échantillon E.

Le coût computationnel du *raymarching* naïf dépend fortement de la fréquence d'échantillonnage, de la résolution de rendu, de la méthode d'évaluation du volume (texture ou volume procédural). Par ailleurs, l'algorithme étant basé sur des boucles, il n'est pas adapté à l'architecture moderne des GPU. En pratique, une implémentation brute est rarement envisageable dans une production.

Figure 2.14 – Raymarching : scène procédurale
Fragment shader WebGL.
La densité est générée à partir d'une fonction de bruit, ce qui est généralement incompatible avec les simulations propres aux effets de gameplay.

2.3.2 Texture 3D – espace frustum

Dans cette solution basée sur une texture 3D de dimensions fixes, les calculs de radiance s'effectuent par voxel. Le coût computationnel est ainsi mieux maîtrisé. La texture est réécrite en temps réel, ce qui permet d'y incorporer des simulations dynamiques.

Wronski[17] et Hillaire [5] présentent en 2014 et 2015 deux stratégies similaires, implémentées respectivement dans AnvilNext (Ubisoft) et Frostbite (EA). On peut résumer l'algorithme en quatre grandes étapes.

2.3.2.1 – Rastérisation des objets volumétriques

Les objets volumétriques de la scène (système de particules [voir partir 3.2], atmosphère ambiante) sont voxélisés dans une texture tridimensionnelle alignée sur le *frustum* de la caméra (ou cône de vision). Cette étape s'apparente à une rastérisation 3D. Elle est réalisée par un *compute shader*. Un algorithme de rastérisation tridimensionnelle est détaillé dans la partie 3.2.

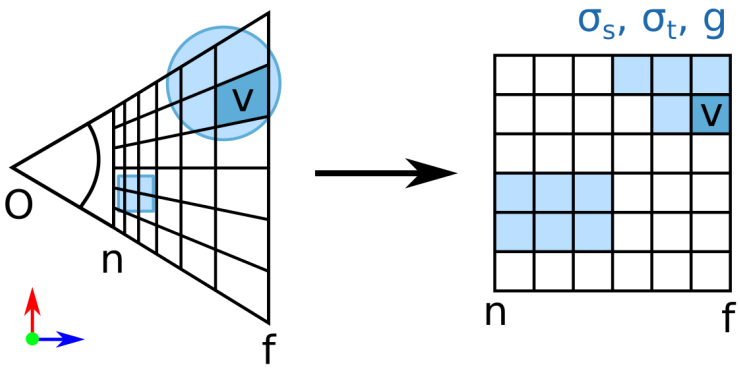


Figure 2.15 – Voxélisation des paramètres du milieu participatif
La densité de voxels décroît avec la distance. A volume égal, un objet au niveau du plan de clipping proche (n) est représenté par un plus grand nombre de voxels qu'un objet au niveau du plan de clipping distant (f).

En pratique, plusieurs textures peuvent être nécessaires en fonction du nombre de paramètres décrivant le volume. La transformation vers l'espace texture n'est pas affine (les distances ne sont pas préservées) : la précision est supérieure à proximité de l'observateur.

C'est la seule étape dont la complexité est proportionnelle au nombre d'objets volumétriques à évaluer.

2.3.2.2 – Évaluation de l'éclairage

La diffusion incidente est évaluée (voir 1.2.10) à partir de la fonction de phase (g), des coefficients d'atténuation (σ_t) et de diffusion (σ_s) et des informations de lumière de la scène (description des sources, *shadow* et/ou *deep shadow map*).

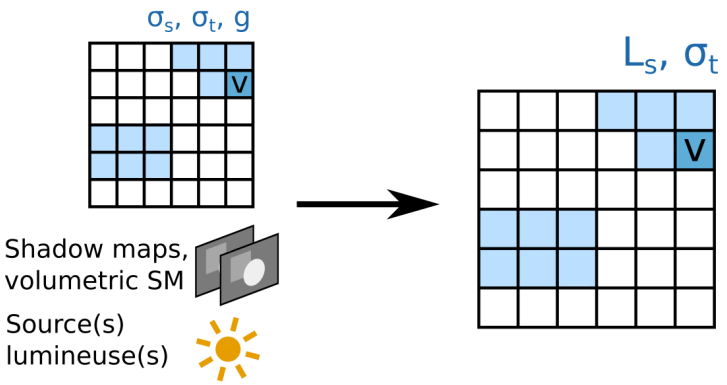


Figure 2.16 – Évaluation par voxel de la diffusion incidente et de l'atténuation

Le coût computationnel de cette étape est fonction de la résolution choisie de la texture et du nombre de sources lumineuses à évaluer.

2.3.2.3 – Intégration

L'étape d'intégration permet de transformer les données locales en données relatives au trajet voxel→observateur et donc directement exploitables par un *fragment shader*. L'atténuation et la radiance sont intégrées, encore une fois selon les principes présentés respectivement en 1.2.7.1 et en 1.2.10.

L'évaluation se fait nécessairement des voxels proches aux voxels distants. Ainsi, contrairement à l'étape précédente, un *thread* n'évalue non pas un voxel mais une colonne de voxels dans l'axe Z.

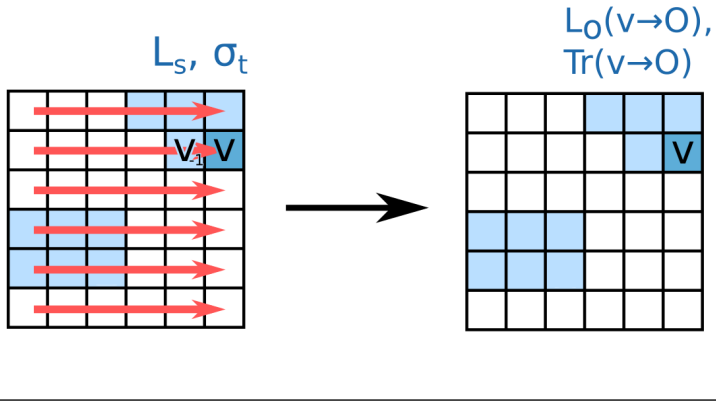


Figure 2.17 – Intégration de la radiance et de la transmittance
Les valeurs au voxel v dépendent de celles calculées au voxel v-1.

Le coût de cette évaluation est également uniquement dépendant de la résolution choisie pour la texture 3D.

2.3.2.4 – Exploitation de la texture, rendu

La texture finale décrit l'atténuation et la radiance parvenant à la caméra depuis tout point visible de la scène. Lors du rendu d'une surface, en *deferred* comme en *forward,* la texture est échantillonnée à la position du fragment. La transmittance atténue la couleur de la surface (par multiplication), la radiance s'y ajoute. Cette fusion s'apparente au mode alpha prémultiplié.

$$c = c_{surf} \cdot Tr(p \rightarrow o) + L_o(p \rightarrow o)$$

avec :

c	=	couleur perçue (surface + milieu participatif)
c_{surf}	=	couleur de la surface seule
$Tr(p \rightarrow o)$	=	transmittance de la position du fragment à l'observateur
$L_o(p \rightarrow o)$	=	radiance ajoutée (par le milieu participatif)

Figure 2.18 – Exploitation des données volumétriques, fusion avec la couleur de la surface

2.4 – Conclusion

Le rendu volumétrique par voxélisation de l'espace du frustum est une solution robuste. Le coût computationnel de cette méthode est en grande partie contrôlable. La discrétisation par rastérisation et l'évaluation de la lumière de façon parallèle la rendent particulièrement adaptée au hardware moderne.

Les dimensions de la texture sont encore limitantes pour représenter certains détails.

Les effets localisés et diffus sont traités conjointement et de façon fidèle à la physique. Comme pour le rendu surfacique, on peut s'attendre à ce que la standardisation des définitions facilite le travail des artistes.

Aujourd'hui, les moteurs utilisés en production intègrent généralement des *shadow maps* volumétriques ainsi qu'un système de voxélisation. La combinaison de ces deux solutions permet de représenter une grande variété d'effets, détaillés ou diffus, dépendant de ressources recalculées ou basées sur la physique.

3 – Expérimentations

Nous présentons deux méthodes originales de rendu volumétrique. Elles traitent essentiellement des systèmes de particules, plus que de l'atmosphère ambiante. Les systèmes de particules sont des ensembles de points soumis à des forces. Dans le jeu vidéo, ils modélisent le comportement d'une grande partie des effets visuels.

3.1 – Billboards volumétriques screen space

Comme nous l'avons vu, l'utilisation de *billboards* permet d'afficher des motifs détaillés avec un contrôle important (voir 2.2). Cependant, la modélisation de l'ombrage volumétrique repose sur des algorithmes complexes (voir 1.3.3) et la gestion correcte de la transparence nécessite un rendu ordonné (voir 2.2.3). Nous proposons une solution exempte de ces contraintes, basée sur un *pass* dédié.

L'idée centrale est de construire une représentation volumétrique à partir d'un ensemble de *billboards*. Au lieu d'évaluer l'éclairage pour chaque *billboard* semi-transparent, donc potentiellement un grand nombre de fois par pixel, on choisit d'accumuler trois valeurs : la transmittance le long du rayon d'observation ainsi que les distances caméra-fragment maximale et minimale. Ces paramètres nous permettent ensuite d'évaluer globalement le volume, une fois par pixel. Notre méthode s'apparente à un rendu *deferred* dédié à la transparence.

Cet algorithme s'appuie sur deux *pass* de rendu.

Le prototype présenté est développé en C++ avec l'API graphique Vulkan.

3.1.1 – Pass transparent dédié

Le pass dédié aux billboards semi-transparents permet l'évaluation de la transmittance et des deux distances.

La texture des *billboards* représente un rendu non éclairé monochromatique.

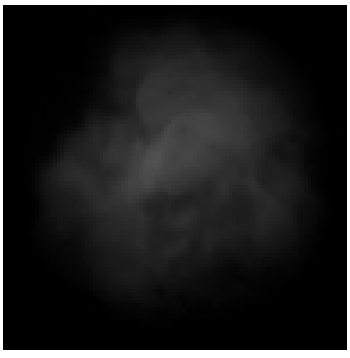


Figure 3.1 – Billboards volumétriques : texture

Cette texture peut être interprétée comme l'inverse de la transmittance. Les zones claires sont denses et/ou épaisses, les zones foncées sont peu denses et/ou fines.

Une propriété de la transmittance entre deux points est qu'elle est le produit de transmittances intermédiaires (voir 1.2.7.1). Ainsi, la transmittance globale le long du rayon d'observation correspond au produit de la transmittance de tous les fragments présents le long de ce rayon, et ce, **dans n'importe quel ordre**.

Le mode de fusion du *pass* de transparence est paramétré en « produit » et initialisé à 1. Les distances fragment-observateur minimales et maximales peuvent être toutes les deux sélectionnées via le seul mode de fusion « minimum ». En effet :

$$max(a,b) = -min(-a,-b)$$

Vulkan permet de paramétrer plusieurs sorties et plusieurs modes de fusion dans un seul *pass* de rendu.

Les fragments sont déplacés le long de l'axe d'observation en fonction de la transmittance. On fait la supposition qu'un fragment dense, donc avec une faible transmittance, correspond à un volume plus plus épais, dans l'axe d'observation.

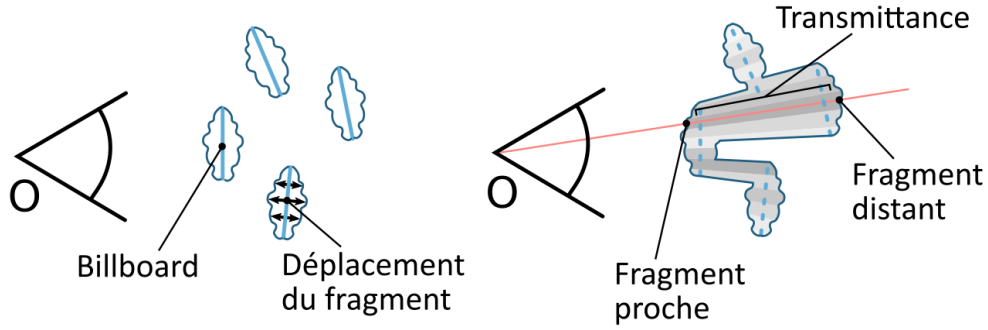


Figure 3.2 – Billboards volumétriques : principe
Trois valeurs sont évaluées par le pass transparent : la transmittance le long de l'axe d'observation, par produits successifs des transmittances des fragments, et les distances fragment-observateur extrêmes.

```
    float oneMinusTransmittance = texture(billboardTexture,uv).r;

    if (oneMinusTransmittance <= 0) discard;

    float transmittance = 1 - oneMinusTransmittance;
    float distToFragment = length(cameraPosition - fragmentPosition);

    float displace = displace(oneMinusTransmittance);

    outputTransmittance = transmittance;
    outputDistances = vec2(distToFragment - displace, -(distToFragment + displace));
```

Figure 3.3 – Billboards volumétriques : fragment shader, pass transparent (GLSL)
La fonction displace() détermine l'intensité du déplacement en fonction de la valeur échantillonnée. Dans notre implémentation, elle est définie par l'expérimentation à $f(x) = x^{0.25}$.

3.1.2 – Pass de rendu volumétrique

Le second *pass* évalue le rendu du volume à partir des données du premier.

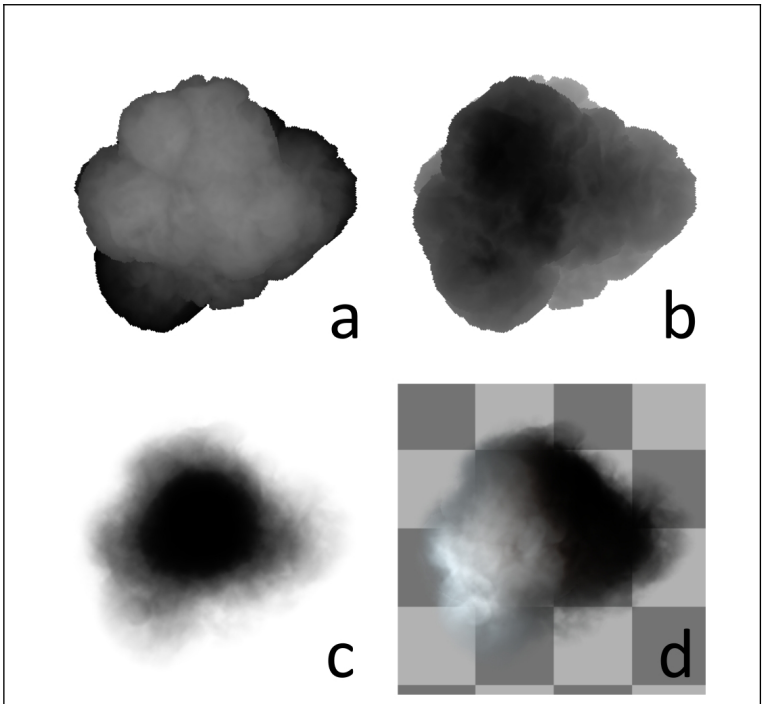


Figure 3.4 – *Billboards volumétriques*
Sorties du pass transparent (a, b, c) et rendu (d)
a. Distance du fragment de plus éloigné
b. Distance du fragment le plus proche
c. Transmittance

Pour simplifier cette présentation, on considère l'albédo à 1. Ainsi, les coefficients de diffusion et d'atténuation sont équivalents (voir 1.2.7 figure 1.24) et s'apparentent à la densité locale du milieu participatif.

L'éclairage est évalué par *raymarching*. Le coefficient de diffusion est reconstruit à partir de la transmittance et des deux distances :

$$(1)\quad Tr(a \rightarrow b) = e^{-\sigma_t \cdot (b-a)}$$

$$(2)\quad \sigma_t = \ln \frac{Tr(a \rightarrow b)}{-(b-a)}$$

Figure 3.5 – *Calcul du coefficient d'atténuation à partir de la transmittance Tr et de la distance b-a*
(1) Loi de Beer-Lambert
(2) Réorganisation des termes

Les échantillonnages indirects (en dehors du rayon correspondant au pixel évalué) sont évalués par projection d'une position dans l'espace de la texture.

```
vec3 getLocalSigmaT(vec3 position, vec3 color)
{
    vec2 screenUV = worldPosToScreen(position);
    vec2 distances = texture(distancesTexture, sceenUV).rg;
    float transmittance = texture(transmittanceTexture, sceenUV).r;
    float nearDistance = distances.r;
    float farDistance = -distances.g; // encoded negated in transparent pass
    float thickness =  farDistance -  nearDistance;
    float distanceFromCamera = length(cameraPosition – position);
    float sigmaT = -log(transmittance) / thickness; // equation (2)

    return color * sigmaT * remap((distanceFromCamera – nearDistance) / thickness);
}
```

Figure 3.6 – Fonction getLocalSigma() (GLSL)
Entrée : *position, couleur*
Sortie : *coefficient d'atténuation*

La couleur est utilisée comme facteur multiplicatif du coefficient d'atténuation calculé.

Notre description du volume est par design incomplète : le coefficient d'atténuation augmente brutalement au niveau de la distance minimale, puis reste constant pour redescendre brutalement en quittant le milieu participatif (voir figure 3.2). Nous proposons de remapper le coefficient d'atténuation reconstruit pour créer une transition douce aux limites.

Concernant la fonction de remappage r() recherchée :

- Elle est définie sur [0-1]
- Elle tend vers 0 en se rapprochant de ses limites
- Son intégrale est de 1, pour préserver la transmittance totale

Nous proposons la fonction suivante :

$$(1)\quad f(x) = min(\frac{x}{T} \cdot \frac{1}{1-0.5 \cdot T}, \frac{1}{1-0.5 \cdot T})$$
$$(2)\quad r(x) = f(1 - 2 \cdot |x-0.5|)$$

Figure 3.7 – Fonction de remappage r(x) (2)
(1) Fonction intermédiaire
Le paramètre T contrôle la pente de la fonction

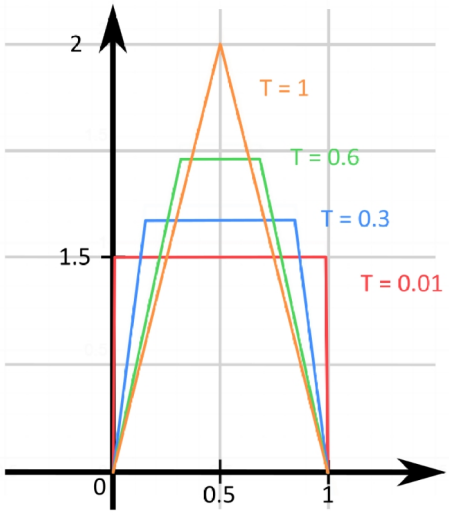


Figure 3.8 – *Fonction de remappage r(x) : visualisation*

Cette approche nous permet de diminuer fortement les artefacts sans augmenter la fréquence d'échantillonnage.

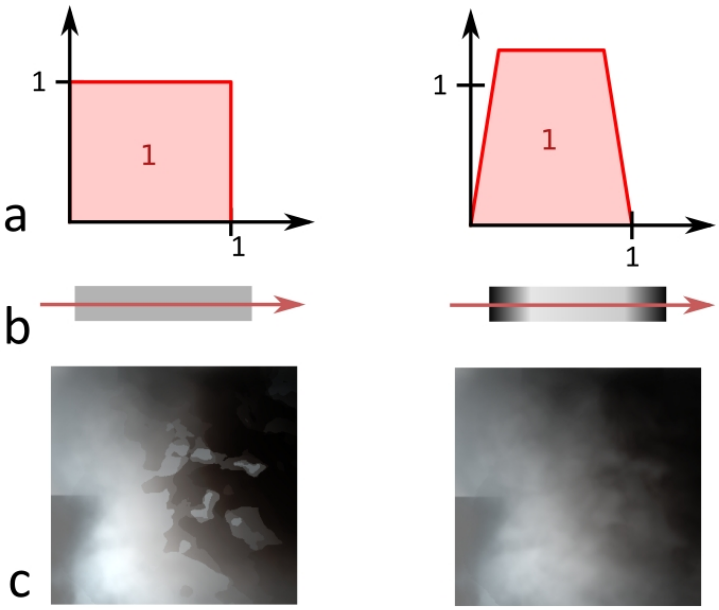


Figure 3.9 – ***Coefficient d'atténuation***
à gauche : situation naïve
à droite : remappage
a. Fonction r(x) sur 0-1 (T = 0.4)
b. Segment de volume le long du rayon.
c. Rendus.

3.1.3 – Résultats

Notre stratégie nous permet d'obtenir un ombrage volumétrique de *billboards* sans rendu ordonné ni *shadow map*. Le nombre d'échantillons du *raymarching* n'est pas nécessairement élevé : les détails de haute fréquence apportés par la texture sont préservés et les artefacts de sous-échantillonnage réduits par le remappage.

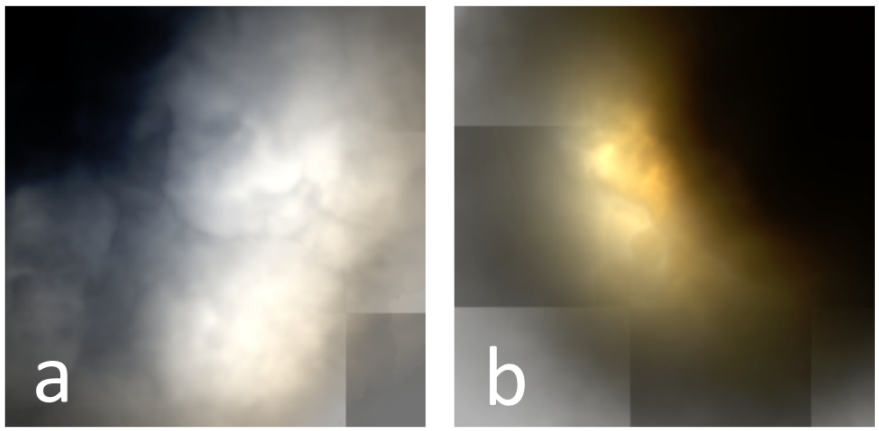


Figure 3.10 – *Billboards volumétriques screen-space : résultats*
a. La source lumineuse est placée sur la droite. 100 échantillons/pixel. La teinte varie au cours du trajet de la lumière, selon le coefficient d'absorption.
b. La source lumineuse est placée à l'intérieur du volume. 25 échantillons/pixel.

Notre méthode de reconstruction reste malgré tout inadaptée aux effets à grande échelle ou comportant d'importantes zones de vide.

Son aspect repose en partie sur des ajustements heuristiques (fonctions de déplacement et de remappage) inconsistants. Enfin des artefacts persistent. Les limites des *billboards* sont parfois visibles. Elle trahissent le changement brutal de profondeur inhérent à notre design.

3.2 – Système de particules voxélisées

Nous présentons une implémentation d'une méthode de discrétisation proche de celle présentée en 2.3. Dans cette solution moins générale, l'essentiel des ressources est consacré à la représentation volumétrique d'un système de particules dense.

Le prototype est développé dans le moteur Unity. La dynamique des particules est gérée par la technologie PopcornFX.

Chaque particule correspond à un point doté d'un rayon d'influence. L'ensemble des particules est évalué comme un champ de densité. L'algorithme est divisé en deux étapes : la discrétisation et l'évaluation du champ.

3.2.1 – Discrétisation

Un *compute shader* rastérise les particules dans l'espace du *frustum* de vue.

Les positions sont tout d'abord transformées vers l'espace de la caméra. Cette transformation préserve les distances, ce qui permet de calculer une *bounding box* valide.

La *bounding box* est ensuite transformée dans l'espace de la texture 3D. Elle permet de déterminer quels texels évaluer.

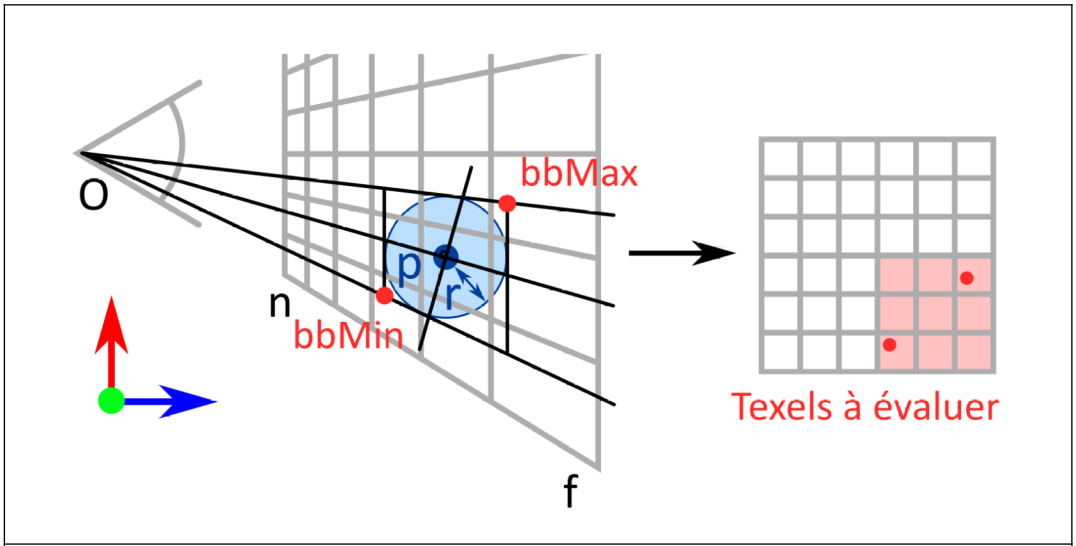


Figure 3.11 – Calcul d'une bounding box alignée avec le frustum : principe géométrique
p : position de la particule
r : rayon de la particule
bbMin, bbMax : limites minimale et maximale de la bounding box

Chaque texel de la *bounding box* est évalué et sa valeur est incrémentée en fonction de sa distance par rapport au centre de la particule.

```
// Gets data from the particle buffer
float radius = getSize(particleID);
float3 position = position(particleID);

// To view space
float3 positionViewSpace = mul(CameraView), float4(center, 1)).xyz;

float3 viewDirection = normalize(positionViewSpace);

// Builds a vector which is orthogonal to the (camera -> particle center) vector
float3 rightUp = normalize(float3(-viewDirection.z, 0, viewDirection.x))
                + normalize(float3(0, -viewDirection.z, viewDirection.y));

// Computes bounding box
float4 bbMax;
bbMax.xyz = centerViewSpace + (rightUp) * radius;
bbMax.xyz = (centerViewSpace.z - radius) * (bbMax.xyz / bbMax.z);

float4 bbMin;
bbMin.xyz = centerViewSpace - (rightUp) * radius;
bbMin.xyz = (centerViewSpace.z + radius) * (bbMin.xyz / bbMin.z);

// Transforms bounding box to texture space (uvw coords)
bbMax = mul(TextureProjection, float4(bbMax.xyz, 1));
bbMax.xyz /= bbMax.w;

bbMin = mul(TextureProjection, float4(bbMin.xyz, 1));
bbMin.xyz /= bbMin.w;

bbMax.xyz = bbMax.xyz * 0.5f + 0.5f;
bbMin.xyz = bbMin.xyz * 0.5f + 0.5f;

bbMin.xyz = saturate(bbMin.xyz);
bbMax.xyz = saturate(bbMax.xyz);

// Transforms to texel ID
uint3 bbMaxID = (bbMax.xyz) * uint3(_SDFTextureSize) + uint3(1, 1, 1);
uint3 bbMinID = (bbMin.xyz) * uint3(_SDFTextureSize);

// Extent of the bounding box
uint3 delta = clamp((bbMaxId - bbMinId), uint3(0,0,0), TextureSize);
```

Figure 3.12 – Rastérisation 3D : compute shader, transformation (HLSL)

Cette étape se fait de manière parallèle : l'écriture doit reposer sur des opérations atomiques. L'API graphique Direct3D (sur laquelle repose Unity) fournit un ensemble de fonctions atomiques opérant sur les entiers.

```
// Iterates over texels of the bounding box
for (uint i = 0; i < (delta.x*delta.y*delta.z); i++)
{
    // Computes local ID
    uint3 localID;
    localID.z = floor(i / (delta.x * delta.y));
    localID.y = floor((i - localID.z * delta.x * delta.y) / delta.x);
    localID.x = floor(i - localID.z * delta.x * delta.y - localID.y * delta.x);

    uint3 texelID = localID + bbMinID;
    float4 position = float4((float3(texelID) + 0.5f) / TextureSize, 1);

    // Transforms position to view space
    position.xyz = position.xyz * 2.0f - 1.0f;
    position = mul(textureInverseProjection, float4(position.xyz, 1.0f));
    position.xyz /= position.w;

    // Evaluates distance to center and writes density
    float distToCenter = length(centerViewSpace - position.xyz);
    if (distToCenter <= radius)
    {
        float density = saturate(distToCenter / radius);
        // Atomic add
        InterlockedAdd(texture[texelID], uint(density * INT_BIG));
    }
}
```

Figure 3.13 – Rastérisation 3D : compute shader, évaluation / écriture (HLSL)
INT_BIG définit un entier élevé constant. On l'utilise pour encoder un flottant positif en entier non signé.

3.2.2 – Exploitation de la texture, rendu

Contrairement aux méthodes par voxelisation présentées en partie 2.3, dans lesquelles l'éclairage est calculé par voxel, l'évaluation de la texture 3D se fait directement par *raymarching*. Sur cet aspect, elle se révèle plus coûteuse mais elle offre une finesse importante dans notre cas, celui d'un effet relativement localisé. Par ailleurs, le *raymarching* nous permet également de tracer une isosurface (voir figue 3.15).

3.2.3 – Résultats

Comme pour les autres techniques basées sur la voxélisation, le coût variable réside dans la première étape.

La diffusion de la lumière est modélisée selon la théorie physique uniquement. La rastérisation permet de traiter un grand nombre de particules. Les figures 1.30 et 1.35 illustrant les paramètres physiques des milieux participatifs sont réalisées avec cette méthode. Elle permet de créer par accumulation de densité des effets visuels variés avec une grande consistance (figure 3.14).

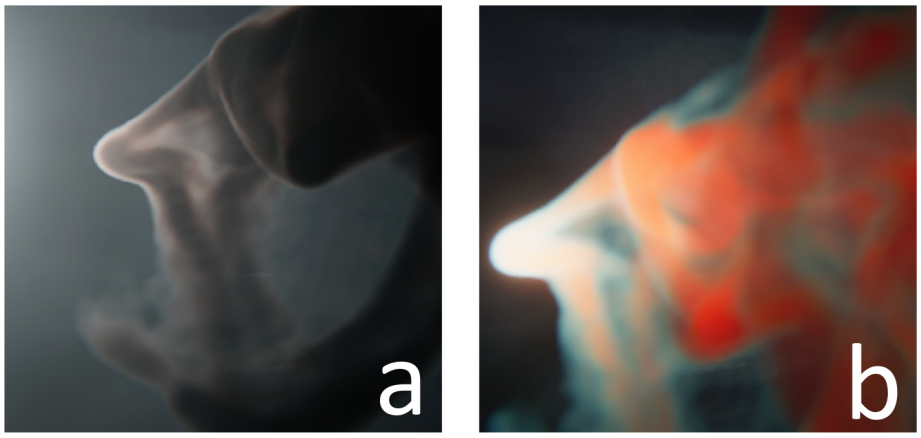


Figure 3.14 – Système de particules voxélisées
Variations des paramètres physiques

La précision dépend de la résolution de la texture. Les motifs sont généralement plus diffus que lors de l'utilisation de textures 2D.

Avec une densité élevée et en traçant une surface, on peut reproduire le phénomène de transluminescence (ou *subsurface scattering*).

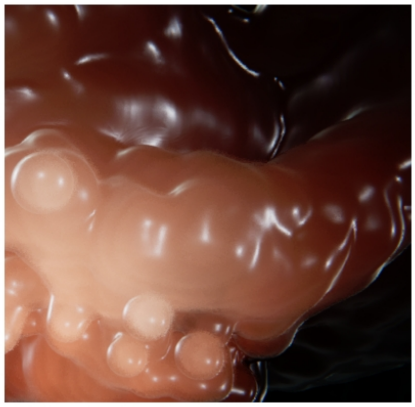


Figure 3.15 – Système de particules voxélisées : *SSS*

Les résultats sont visuellement satisfaisants. Si les performances sont bonnes, cette technique reste malgré tout moins extensible à une scène complexe que celles présentées en 2.3 dans lesquelles l'éclairage est calculé par voxel.

Conclusion et perspectives

En 2020, le problème du rendu volumétrique en temps réel n'a pas de solution unique. Comme nous l'avons montré, les moteurs de jeux sont fondamentalement inadaptés au rendu d'objets semi-transparents.

Nous avons distingué deux grand types de solutions. Celles basées sur des images en 2D, avec une forte variabilité artistique, et celles, plus consistantes, construites autour d'une représentation tridimensionnelle discrète de la scène. C'est ce deuxième type, unifié et physiquement correct, qui se développe aujourd'hui dans l'industrie. Comme pour le rendu surfacique, la production artistique réaliste se trouve facilitée par l'élimination des paramètres n'ayant pas de justification physique.

Ces solutions unifiées sont encore limitées par le matériel. Celui-ci évolue, mais avec lui les ambitions concernant les autres limites visuelles du temps réel, et elles sont encore nombreuses : quantité de polygones, ombrage par *ray tracing*, illumination globale, profondeur de champ et flou de mouvement…

L'idée d'un design unique traitant la totalité de ces problèmes, comme c'est le cas avec l'image pré-calculée, constitue encore une perspective lointaine. Le jeu vidéo relève toujours de l'art de tromper en contournant les limitations techniques.

Table des figures

Table des figures (suite)

Bibliographie

[0] Léonard de Vinci : Artiste et scientifique, Domenico Laurenza

[1] Kajiya, James T., « The rendering equation », 1986

[2] R. Cook and K. Torrance. « A reflectance model for computer graphics »

[3] B. T. Phong, « Illumination for computer generated pictures » , 1975

[4] James F. Blinn, « Models of light reflection for computer synthesized pictures » , 1977

[5] S.K. Nayar and M. Oren, « Generalization of the Lambertian Model and Implications for Machine Vision », 1995

[7] L. G. Henyey, J. L. Greenstein, « Diffuse radiation in the Galaxy », The Astrophysical Journal, 1941

[8] Real-Time Rendering, Fourth Edition

[9] Physically Based Rendering : from theory to implementation

[10] S. Hillaire, « Physically-Based and Unified Volumetric Rendering in Frostbite » 2015

[11] Carsten Wenzel, Real-time Atmospheric Effects in Games Revisited, 2007

[12] Sun, Bo, Ravi Ramamoorthi, Srinivasa Narasimhan, and Shree Nayar, « A Practical AnalyticSingle Scattering Model for Real Time Rendering »

[13] Pegoraro, Vincent, Mathias Schott, and Steven G. Parker, « An Analytical Approach to Single Scattering for Anisotropic Media and Light Distributions »

[14] Miles Macklin, In-Scattering Demo, 2010

[15] Tom Lokovic, Eric Veach, "Deep Shadow Maps," 2000

[16] Salvi, Marco, Kiril Vidimce, Andrew Lauritzen, and Aaron Lefohn, « Adaptive VolumetricShadow Maps », 2010

[17] Fürst, Rene, Oliver Mattausch, and Daniel Scherzer, « Real-Time Deep Shadow Maps », Wolfgang Engel's GPU Pro 360

[18] Jon Jansen, « Fast rendering of opacity-mapped particles using DirectX 11 tessellation and mixed resolutions », 2011

[19]Wronski, Bartlomiej, « Volumetric Fog: Unified Compute Shader-Based Solution to Atmo-spheric Scattering », 2014