RAY OF GOD
Rapport Méthodes et algorithmes

Yuko Baudet - Valentin Bonnesoeur - Bastien Schatt

7 novembre 2014

Clients : Mathias Paulin - David Vanderhaeghe

Table des matières

1 Introduction **3**

2 Problématique du chef d'oeuvre **4**

3 L'existant **5**

4 Algorithme du chef d'oeuvre **7**
4.1 Formulation de la scattering integrale 7
4.2 Explication des étapes de l'algorithme 8

5 Lancement **16**

6 Planning **18**

7 Glossaire **19**

Références **20**

A ANNEXE - La géométrie épipolaire **21**

1 Introduction

Ce rapport entre dans le cadre du Chef d'Oeuvre du Master Image et Multimédia. Notre sujet concerne le développement d'une approche physique réaliste d'un modèle d'ombrage volumique en temps réel destiné au laboratoire de l'IRIT. Dans ce premier rapport, nous présentons les algorithmes et méthodes existants et nécessaires à la réalisation de ce projet ainsi que les outils que nous utiliserons lors de notre travail.

Le rapport des « Méthodes et Algorithmes »permet à l'équipe de comprendre les algorithmes ainsi que les formules mathématiques que chacun va devoir développer tout au long du chef d'oeuvre.

Il s'agit également de détailler les travaux ultérieurs qui ont été fait sur ce sujet, de les comprendre, et, si possible, de les intégrer dans nos travaux.

2 Problématique du chef d'oeuvre

La simulation de l'éclairage dans les milieux participants, comme la fumée ou le brouillard, augmente considérablement le réalisme du rendu de scènes complexes. Différentes méthodes existent mais ne sont pas réalisables en temps réel, comme par exemple la méthode de Monte Carlo qui peut nécessiter de plusieurs minutes à plusieurs heures (suivant la complexité de la scène) pour effectuer le rendu d'une seule image. Les méthodes de rendus temps réel pour simuler un milieu participant sont tirées ou dérivées du « fog model »qui existait dans les API graphiques avant l'introduction des shaders (OpenGL 2, DirectX 9). Néanmoins, ces approches excluent un grand nombre d'effets physiques réalistes, comme le halo autour des sources de lumière, l'atténuation des ombres et des tâches spéculaires des surfaces ou l'ombrage volumique résultant de l'occlusion par les objets présents sur la scène.

Le domaine de la simulation de l'éclairage dans les milieux participants à tout de même grandement évolué ces dernières années. Aujourd'hui, il existe plusieurs méthodes permettant d'avoir une approche en temps réel. Comme par exemple en pré-calculant et en exploitant certaines données sous formes de textures, tout en intégrant les effets précédemment mis de côté.

L'article « Real-Time Volumetric Shadows using 1D Min-Max Mipmaps »[2] fait partie de l'état de l'art d'ombre volumique dans les milieux participants en rendu temps réel. Dans ce chef d'oeuvre, la problématique sera d'implémenter cet article.

Notre premier objectif est de comprendre les méthodes et algorithmes nécessaire à la réalisation du projet. Nous avons à notre disposition plusieurs articles décrivant l'évolution temporelle du problème ainsi que des techniques de résolution. Dans un premier temps, nous analyserons ces articles en présentant les méthodes et formules de calcul. Nous détaillerons également l'algorithme que nous développerons par la suite. Comme il s'agit aussi de mettre en application nos acquis de gestion de projet, nous présenterons dans un second temps les outils que nous utiliserons durant la période de travail (pour le développement, le versionning, la gestion des bugs etc).

Dans ce chef d'oeuvre, notre but est donc d'analyser la bibliographie récente puis d'implémenter un démonstrateur technologique intégrant les effets

de l'éclairage dans les milieux participants. Ce démonstrateur doit bien évidemment fonctionner en temps réel, dans le sens où il doit être interactif.

3 L'existant

Tous les articles cités par la suite se placent dans un environnement isotrope et homogène. Le milieu participant englobe toute la scène, y compris la caméra et les sources de lumières. De plus on ne traite que le cas du « single scattering ». Les articles étant relativement dépendant de leur prédécesseurs, nous préférons les traiter dans un ordre chronologique.

La méthode Sun et coll. [4] dérive le modèle de brouillard qu'on retrouve par exemple dans OpenGL pour en faire une version physiquement réaliste sans augmenter le temps de calcul. En effet, leur modèle est purement analytique : pour chaque pixel on va calculer la « quantité de brouillard »à ajouter de façon mathématique, donc hautement parallelisable dans un pixel shader !

Dans ce premier article, la luminance perçu en un point de l'objectif caméra ou de la surface d'une scène est la somme des contributions de la lumière directe émise par la source lumineuse et de la lumière diffusée par le milieu, appelé « single scattering ».

L'inconvénient de cette formule est que les termes sont complexes, donc coûteux à calculer. En effet, le calcul de la lumière diffuse est une intégrale simple pour le calcul des lueurs et une intégrale double pour le calcul d'éclairement des surfaces. L'apport de cet article est de décomposer ces intégrales afin de pouvoir en pré-calculer une partie. Ainsi, l'intégration du single scattering peut être remplacé par de simples accès à des textures 2D (très rapide sur GPU).

Cette méthode rend la lumière de la scène en présence de milieu participant plus réaliste que ce qui existait précédemment, notamment en faisant apparaître des lueurs autour des sources de lumières. Cependant, il manque à cette méthode un aspect important puisqu'elle ne prend pas en compte les

occlusions de la lumière. La radiance est donc surestimée là où il devrait y avoir des volumes d'ombre.

Ce problème est traité par Wyman et Ramsey [5] au moyen d'une méthode basée sur le ray marching. Ici les volumes d'ombres sont traitées à l'aide de « shadow volumes ». Le calcul de la luminance devient une intégration par parties ne prenant pas en compte les zones incluses dans ces « shadow volume »(c'est à dire les parties dans l'ombre). L'inconvénient de cette démarche est que la génération des shadow volumes est coûteuse. Pour avoir un résultat précis, et donc éviter de « rater »des zones d'ombre (dû à un échantillonnage pas assez précis), l'échantillonnage du ray marching doit être important et donc très coûteux.

L'article précédent, bien qu'étant imparfait, montre qu'il est possible de calculer des volumes d'ombres dans un milieu participant sans devoir faire du ray marching complet (brute-force) pour chaque pixel de l'image. Engelhardt et coll. [3] propose une autre méthode pour éviter de faire du ray marching sur l'image entière. Le principe est d'échantillonner uniquement des droites épipolaires. Une fois ces droites générées, il est rapide de trouver les intersections avec les rayons de la caméra : au lieu de chercher dans toute l'image (brute-force) on cherche uniquement sur une ligne de l'image. On rajoute ensuite des echantillons à côté des ruptures de profondeurs. Une fois tous les échantillons trouvés, on utilise le ray marching pour calculer l'intégral, puis pour le restant des pixels, leur valeurs est trouvées par interpolation.

Finalement, les articles de Baren et coll. [1] ainsi que Chen et coll. [2], partent de ces droites épipolaires mais ajoute un arbre pour limiter le ray marching et donc accélérer le rendu ! Dans l'article de Baran et coll. [1], les auteurs introduisent un arbre partiel permettant une intégration incrémental beaucoup plus rapide que le ray marching. Cependant, cet arbre ne peut pas être traité de manière massivement parallèle et doit donc être exécuté sur CPU. L'article suivant, écrit par Chen et coll. [2], reprend la conclusion de l'article précédent. En effet, afin d'aller plus vite, il faut une structure de donnée pouvant être traitée en parallèle sur GPU. Cette fois, un arbre min-max est utilisé. Celui-ci est stocké dans une mip-map 1D permettant une utilisation simple et efficace sur GPU. Un avantage supplémentaire de cette méthode par rapport à la précédente est la non nécessité d'effectuer une rectification épipolaire des lignes de vues.

4 Algorithme du chef d'oeuvre

4.1 Formulation de la scattering integrale

La luminance diffusée vers la caméra est intégrée le long de chaque rayon de la caméra jusqu'à rencontrer une surface. Pour chaque échantillon du rayon, si la source de lumière est visible depuis celui-ci, une certaine quantité de cette lumière est diffusée vers l'oeil. De plus, la lumière est également atténuée lors de son transfert de la source de lumière jusqu'à l'oeil. Cela donne l'équation suivante, où L est la luminance diffusée vers l'oeil :

$$L(v) = \int_0^d \mathrm{e}^{-\sigma_t s} V(sv)\sigma_s \rho(\theta) L_{in}(sv) \mathrm{d}s$$

v	direction du rayon de la camera
d	distance entre la camera et la première occlusion le long du rayon
σ_s	coefficient de diffusion
σ_t	coefficient d'atténuation
$V(sv)$	1 si le point (sv) est éclairé, 0 sinon
$L_{in}(sv)$	luminance incidente au point (sv)
$\rho(\theta)$	fonction de phase
θ	angle entre v et la direction de la lumière
$L_{in}(sv) = \frac{I\,\mathrm{e}^{-\sigma_t d(sv)}}{d(sv)^2}$, $d(sv) = \|x - sv\|$	

4.2 Explication des étapes de l'algorithme

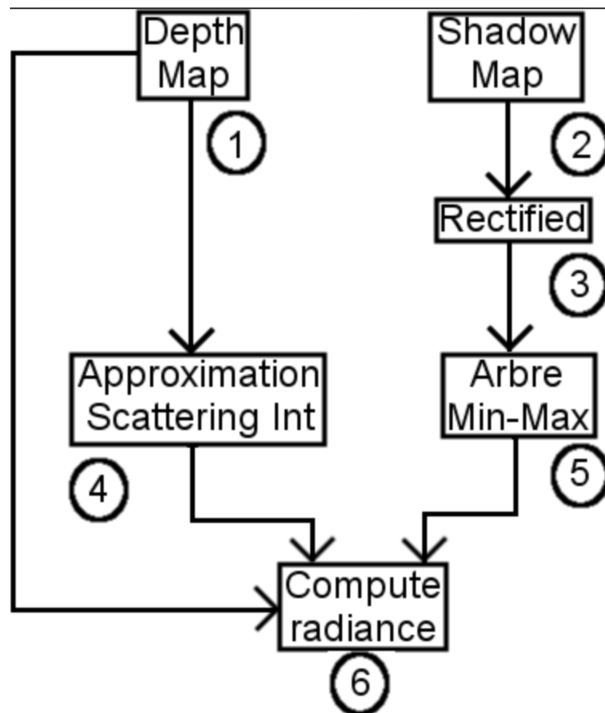

FIGURE 1 – Etape de l'algorithme

La figure 1 présente l'algorithme de notre chef d'oeuvre.

1. **Générer une carte de profondeur du point de vue de la caméra (depth map)**
   Pour chaque point 3D de la scène, il faut calculer sa distance avec la caméra. Donc pour un point X, d(X) est sa profondeur dans le repère caméra. Toutes les profondeurs sont stockées dans une texture, appelée carte de profondeur. Plus exactement, pour chaque pixel, sa profondeur (coordonnées Z) est stockée dans une texture. Une texture est un tableau à deux dimensions, chaque élément étant un pixel de l'écran. De plus, un test de profondeur garantie que la carte de profondeur contient uniquement la position des éléments les plus proches : si deux objets de la scène se projette sur le même pixel, alors le programme compare les deux profondeurs et ne garde que la profondeur

du plus proche de la caméra.

2. **Générer une carte des ombres du point de vue de la lumière (shadow map)**
   Les shadow map sont des cartes de profondeur utilisant le repère de la lumière plutôt que le repère caméra. La génération d'une telle carte est donc similaire à la génération d'une carte de profondeur : la profondeur de chaque pixel de l'image est stockée dans une texture.

   En règle général, cette shadow map est ensuite utilisé lors du rendu de la scène (depuis le point de vue de la caméra) pour tester la visibilité d'un point par la lumière. Ainsi, pour chaque pixel de la scène on compare la profondeur du fragment avec la profondeur contenue dans la shadow map. C'est donc un simple test de profondeur. Si ce test passe (i.e. la profondeur du fragment est inférieur à la valeur contenu dans la texture d'ombre) on sait que le fragment n'est pas visible depuis la lumière et donc que ce fragment est dans l'ombre.

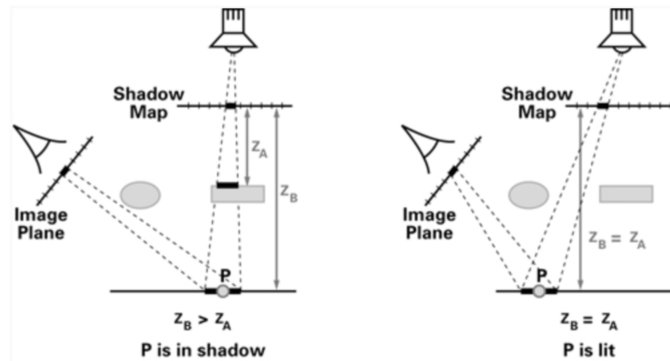

Figure 2 – Shadow map
http ://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter09.html

3. **Faire une rectification épipolaire de la shadow map**
   Une shadow map est donc une carte de visibilité de la lumière : chaque point (dans le repère de la lumière) ayant une profondeur supérieur à la valeur contenue dans la shadow map n'est pas visible par la lumière ! Ces points non visibles forment des ombres volumiques, qu'on

appelle Ray of God.
Pour un rayon de la caméra donné on veut donc savoir quelles parties du rayon sont dans l'ombre et quelles parties sont dans la lumière pour pouvoir faire une intégration par partie du scattering dans le milieu participant.
Il est donc très important de pouvoir accéder rapidement à cette information si on veut que notre application tourne en temps réél. On utilise donc une rectification épipolaire pour faciliter les étapes suivantes.
La rectification épipolaire permet de se placer dans une configuration parallèle (cf. figure 3), c'est-à-dire qu'un point dans la carte de profondeur vue depuis la caméra ait son correspondant sur la même ligne dans la shadow map rectifiée ! Le but est bien sur de pouvoir parcourir cette ligne simplement pour faire l'intégration par parties.

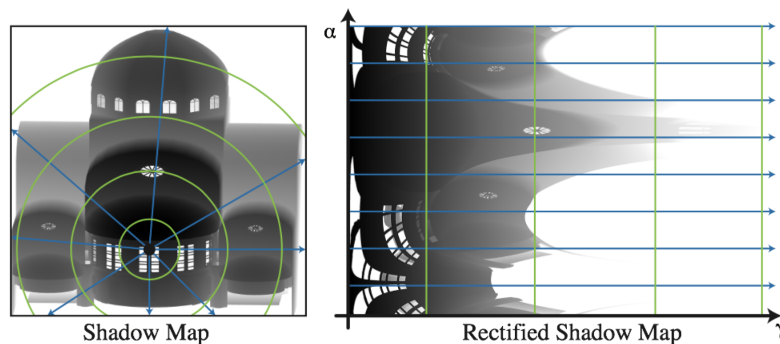

FIGURE 3 – Rectification épipolaire

L'image ci-dessus montre le fonctionnement de la rectification épipolaire. Un rayon émanant de l'épipole de manière radiale (ligne bleue dans la shadow map) se transforme en ligne (dans la shadow map rectifiée).

La rectification épipolaire est une affectation des coordonnées (α, β, γ) à chaque point de l'espace p de telle sorte que les rayon de la caméra soient indexés par (α, β) et que les rayons de la lumière soient indexés par (α, γ).

Tout d'abord le monde est partitionné en « tranche », des plans qui contiennent l'oeil et qui sont parallèles à la direction de la lumière. α donne la tranche du point p.
La coordonnée β donne le rayon de vue à l'intérieur de la tranche. Pour une « point light »c'est l'angle entre le rayon de vue et la direction allant de l'oeil à la lumière.
Finalement, γ donne le rayon de lumière à l'intérieur de la tranche. Dans notre cas (« point light ») on mesure γ comme l'angle du le rayon de lumière vers l'oeil.
Mettons « e »la position de l'oeil, « l »la position de la lumière et « p »le point. Commençons par calculer β, qui est définit comme l'angle entre le rayon de vue ($\vec{ep}$) et la direction entre l'oeil et la lumière ($\vec{el}$). On sait que le produit scalaire entre ces 2 vecteurs (normalisés) est égal au cosinus de l'angle que l'on cherche :

$$\left\langle \vec{ep}, \vec{el} \right\rangle = \cos(\beta) \rightarrow \beta = \arccos(\left\langle \vec{ep}, \vec{el} \right\rangle)$$

On utilise le même principe pour calculer γ , l'angle du rayon de lumière ($\vec{lp}$) vers l'oeil ($\vec{le}$).

$$\left\langle \vec{lp}, \vec{le} \right\rangle = \cos(\gamma) \rightarrow \gamma = \arccos(\left\langle \vec{lp}, \vec{le} \right\rangle)$$

Revenons maintenant à γ. On a vu que le plan déterminant la tranche du point p. Ce plan passe par l'oeil et est parallèle à la direction de la lumière. On peut donc trouver sa normale en utilisant un produit vectoriel entre la direction de la caméra ($\vec{ep}$) et la direction de la lumière ($\vec{lp}$).

$$n = \vec{ep} \wedge \vec{lp}$$

On veut caractériser ce plan par un angle. Pour cela on va utiliser l'angle entre ce plan et le plan ayant pour normal l'axe y ($\vec{up}$).

$$\langle n, \vec{up} \rangle = \cos(\gamma) \rightarrow \gamma = \arccos(\langle n, \vec{up} \rangle)$$

Il est nécessaire ensuite de faire un changement de variable de la scattering integral afin de l'écrire en coordonées épipolaires :

$$L(\alpha, \beta) = \int_0^{D(\alpha,\beta)} \mathrm{e}^{-\sigma_t s(\beta,\gamma)} \, V(\alpha, \beta, \gamma)\sigma_s \rho(\beta, \gamma) L_{in}(\beta, \gamma) \frac{ds}{d\gamma} \mathrm{d}\gamma$$

où D(α , β) est la coordonnées γ du rayon lumineux où le rayon de la caméra (α , β) est bloqué. Excepté le terme de la visibilité, l'intégrale dépend uniquement de β et γ et pas de α . On peut donc écrire :

$$L(\alpha, \beta) = \int_0^{D(\alpha,\beta)} V(\alpha, \beta, \gamma) I(\beta, \gamma) \mathrm{d}\gamma$$

4. **Calculer une approximation de la scattering intégrale sur tous les termes visibles**

   Une fois la rectification épipolaire faite, il serait intéressant d'un point de vue algorithmique de pré-calculer I(β, γ). La fonction I(β, γ) contient tout les termes d'intégration autre que la visibilité. Pour cela, on l'approxime comme une somme des fonctions constantes. Il faut échantilloner I avec des valeurs discrètes et calculer les valeurs propres de la matrice résultante de la discrétisation de I. On utilisera ici une matrice 64x64 pour discrétiser I. On effectue ensuite une décomposition en valeur singulières :

   $$I = PDP^{-1}$$

   avec P la matrice contenant les vecteurs propres de I et D la matrice contenant les valeurs propres de I :

   $$P = \begin{pmatrix} x_{00} & x_{01} & \dots & x_{0m} \\ x_{10} & x_{11} & \dots & x_{1m} \\ \vdots & \vdots & \dots & \vdots \\ x_{n0} & x_{n1} & \dots & x_{nm} \end{pmatrix}$$

$$D = \begin{pmatrix} \lambda_0 & 0 & \dots & 0 \\ 0 & \lambda_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

On ne garde que les N premières valeurs propres ainsi que les N vecteurs propres qui leurs sont associés. Ici, N=4.
Par exemple pour λ_0 :

$$\lambda_0 \begin{pmatrix} x_0 \\ \vdots \end{pmatrix} \begin{pmatrix} x_0^{-1} & \dots & \end{pmatrix}$$
$$= \lambda_0 x_0(\beta) x_0^{-1}(\gamma) + ... + \lambda_0 x_n(\beta) x_n^{-1}(\gamma)$$

avec β le β_{ieme} élément de x et γ le γ_{ieme} élément de x^{-1}

Pour simplifier on utilise les notations suivantes :

$B_i = \lambda_0 x_0$ et $\Gamma_i = x_0^{-1}$

On obtient donc :

$I(\beta, \gamma) \approx \sum_i^N \Gamma_i(\gamma) B_i(\beta)$

Plus N est grand, meilleure sera l'approximation de I, mais les temps de calculs seront plus long. Les auteurs utilise N=4 car cela est suffisant pour un résultat de bonne qualité dans un milieu uniforme.
Ainsi nous obtenons :

$L(\alpha, \beta) \approx \sum_i^N (B_i(\beta) \int_0^{D(\alpha,\beta)} V(\alpha, \beta, \gamma) \Gamma_i(\gamma) \mathrm{d}\gamma)$

Il faut ensuite approximer l'intégrale restante. Pour cela nous l'approximons comme une somme de Riemann :

$L(\alpha,\beta) \approx \sum_i^N (B_i(\beta) \sum_{\gamma<D(\alpha,\beta),S[\alpha,\gamma]>\beta} \Gamma_i(\gamma)\Delta\gamma)$

5. **Calculer un arbre binaire complet pour chaque ligne de la shadow map rectifiée**

   Parcourir une ligne de la shadow map rectifiée pour faire l'intégration par partie peut se révéler encore trop coûteux. On va donc construire un arbre min-max qui se révélera plus efficace à parcourir qu'un simple tableau (ligne de la shadow map rectifiée).

   L'image ci-dessous (figure 4) présente la construction d'un arbre. La partie en bas représente une ligne de la texture rectifiée (elle contient les valeurs β). Les feuilles de l'arbre contiennent ces valeurs, qui indiquent les limites entre ombre et lumière. Par exemple, pour γ=1, la frontière se situe à β=9, c'est à dire qu'il n'y a que de la lumière pour cette valeur de γ. Par contre pour γ=2, la frontière se situe à 2 : pour $\beta < 7$ c'est de l'ombre (le reste est bien entendu dans la lumière). Les noeuds de l'arbre donnent les valeurs min-max des fils.

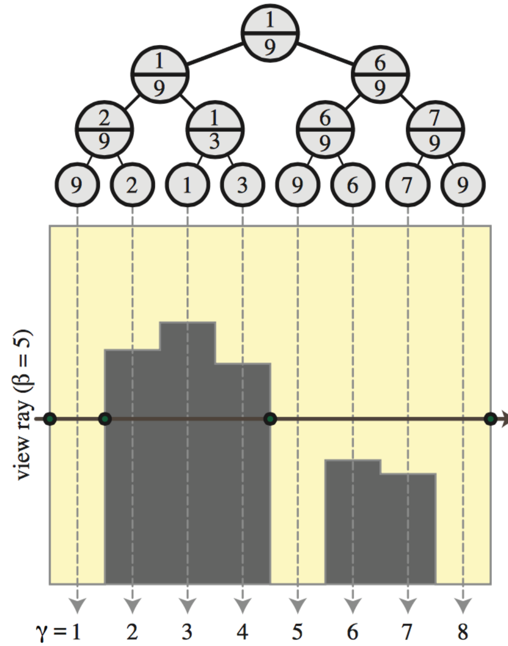

FIGURE 4 – Construction de l'arbre

6. **Pour chaque rayon de la caméra trouver les segments éclairés à l'aide de la min-max map et les accumuler dans le calcul de l'éclairage**

   Il est important de considérer chaque rayons indépendamment, ce qui a pour conséquence d'être massivement parallélisable.

   Pour chaque rayons, il faut chercher les segments éclairés. On cherche donc les segments qui vérifie : S[α, γ] $> \beta$

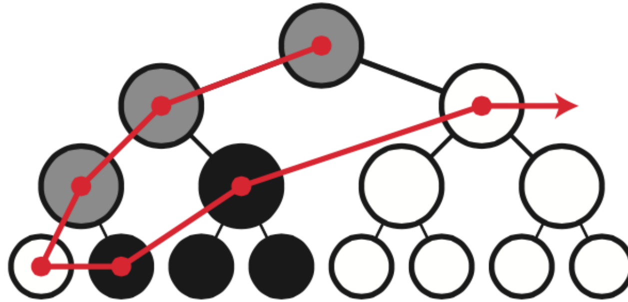

FIGURE 5 – Parcours de l'arbre

Pour calculer la scattering intégrale, chaque arbre est parcouru en seuillant la min-max mipmap avec la valeur du beta du rayon de la caméra associé. Si beta est compris entre le min et le max du noeud, alors le noeud est « gris ». Donc si on continue à parcourir l'arbre dans ce sens, on tombera surement sur un segment éclairé. Sinon, si beta est inférieur à la valeur min du noeud c'est que le segment est complètement éclairé, on rajoute alors à l'intégrale le calcule de sa luminance. Si jamais le noeud est noir, c'est qu'il est complètement dans l'ombre. Il n'est pas nécessaire de poursuivre le parcours dans ce sens, car ses fils seront également en noirs et n'apporteront donc aucune contribution au calcul de la luminance (figure 5).

5 Lancement

Nous voulons un rendu temps réel, il est donc nécessaire d'utiliser un langage offrant des performances maximales. C'est pour cette raison que nous utiliserons le C++.

Ce langage a aussi l'avantage de pouvoir s'interfacer avec les principales API 3D qui existe. Nous utiliserons OpenGL qui a l'avantage d'être mutliplateforme. Nous nous limiterons à la version 3 qui offre un pipeline programmable sans toutefois être trop récent (et donc supporté sur toutes les machines de développement de l'équipe).

CMake sera utilisé pour générer des configurations de compilation diverses, permettant l'utilisation de n'importe quel environnement de programmation.

Concernant la gestion de projet, un bug tracker a également été mis en place. Il nous permettra de définir une roadmap avec des délais précis et de définir les taches à réaliser durant chaque étape. En plus de permettra un suivi de l'avancement du projet, ce bug tracker servira aussi, bien entendu, à répertorier les bugs existants avant de les corriger. Nous avons mis en place un dépôt GIT qui nous permettra tout d'abord d'avoir un versionning de notre travail mais permettra également une mise en commun simplifié de notre travail.

Finalement, nous utiliserons les outils Google pour faciliter la communication et le partage de documents. Google Drive (associé à Google Docs) sera utilisé pour partager des documents et de rédiger de manière collaborative les différents rapports que nous aurons à fournir tout au long de ce chef d'oeuvre. Google Hangout nous permettra de communiquer en groupe et en temps réél, de manière textuelle ou en visio-conférence).

6 Planning

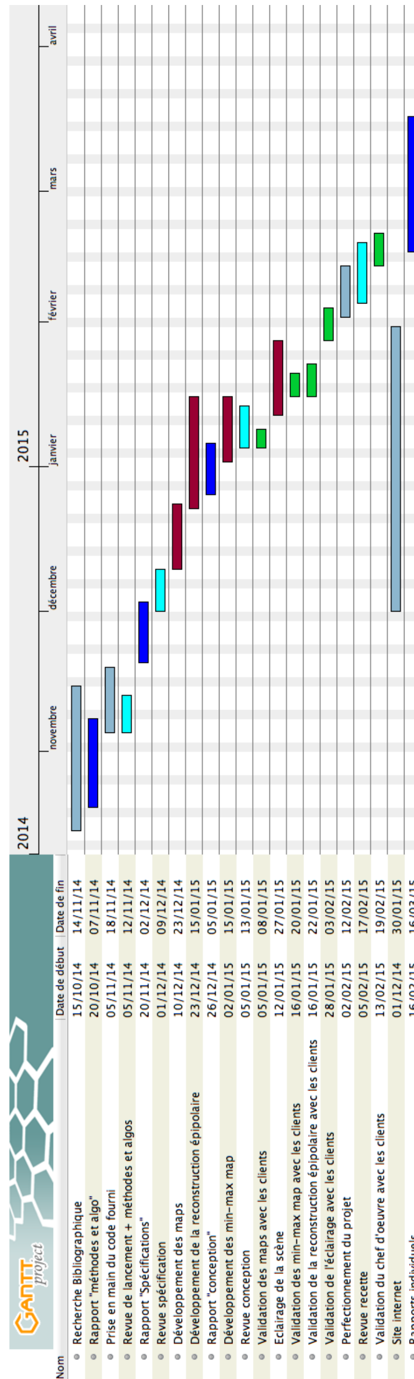

FIGURE 6 – Diagramme de Gantt

7 Glossaire

Isotrope : Caractérise l'invariance des propriétés physiques d'un milieu en fonction de la direction de propagation du rayon lumineux (les propriétés du milieu ne changent pas selon la direction). Le rayon ne sera pas dévié, ni ralenti, ni divisé en spectre.

Single Scattering : Un rayon de lumière peut rentrer dans l'oeil de manière directe (trajectoire droite) ou indirecte (diffusée par un milieu). Dans le cas ou un rayon est diffusé une fois par le milieu on parle de single-scattering.

Ray Marching : Un rayon partant de la caméra est distribuer à travers tous les pixels de l'écran. La scattering intégrale est approximée en se déplacant le long du rayon et en vérifiant que chaque échantillon est éclairé ou dans l'ombre à l'aide d'une shadow map.

Shadow volume :

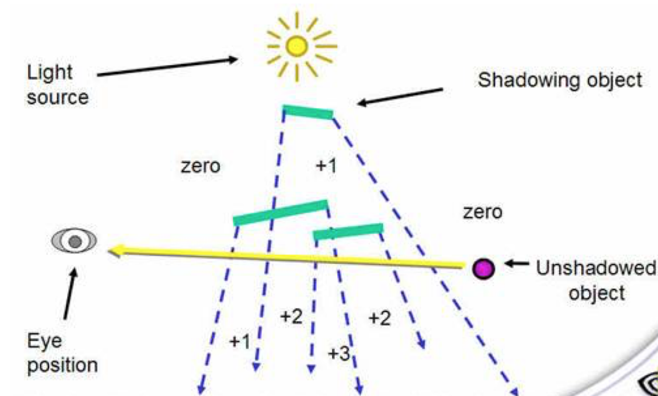

Figure 7 – Shadow Volume - http ://blog.csdn.net/timian/article/details/8966624

Les pixels qui ont une valeur positives sont dans l'ombre.

Références

[1] Baran. A hierarchical volumetric shadow algorithm for single scattering. 2010.

[2] Chen. Real-time volumetric shadows using 1d min-max mipmaps. 2011.

[3] Engelhardt and Dachsbacher. Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. 2010.

[4] Sun. A practical analytic single scattering model for real time rendering. *ACM Transactions on Graphics*, 2005.

[5] Wyman and Ramsey. Interactive volumetric shadows in participating media with single-scattering. 2008.

A ANNEXE - La géométrie épipolaire

Étant donné deux images d'une même scène prise à deux endroits différents, il est nécessaire de trouver les correspondances entre les pixels des deux images. Sachant que le programme doit tourner en temps réel, il n'est pas possible de chercher le correspondant d'un pixel dans toute l'image. Pour simplifier la recherche, nous nous plaçons dans une configuration épipolaire afin que la recherche du correspondant d'un pixel (et donc l'intégration par partie de l'equation du scattering) se limite à une droite. L'équation la droite épipolaire s'écrit ainsi :

$$(t_z.R_2.p^g - t_y.R_3.p^g)x^d + (t_x.R_3.p^g - t_g.R_1.p^g)y^d - t_x.R_2.p^g + t_y.R_y.p^g = 0$$

avec :

$x^d = \frac{z^g R_1 p^g + t_x}{z^g R_3 p^g + tz}$

$y^d = \frac{z^g R_2 p^g + t_y}{z^g R_3 p^g + tz}$

$R = \begin{pmatrix} R_1 \\ R_2 \\ R_3 \end{pmatrix}$, la matrice de rotation

$t = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$, le vecteur translation

Cette expression est donc l'équation de la contrainte épipolaire. Elle donne pour un point pg du plan normalisé gauche le lieu géométrique du plan image normalisé droit contenant les point pouvant correspondre au même point. La droite épipolaire associée au plan image gauche se trouve de la même façon.

La rectification épipolaire permet donc de transformer la shadow map (2D) en une collection de champ 1D indépendants.

Cependant, le parcours d'une droite quelconque dans une image lors de la recherche d'un correspondant n'est pas très aisé. Pour simplifier encore la recherche du correspondant, il faut faire une rectification épipolaire de

nos images. Pour cela on se place dans une configuration parallèle. Cela a pour avantage que la droite épipolaire associée à un point dans une image est située sur la même ligne dans l'autre image . Cette configuration est bien plus intéressante d'un point de vue algorithmique lors de la phase de mise en correspondance. Pour appliquer la rectification épipolaire à une image, il faut lui appliquer deux transformations : des homographies.

Par ailleurs, quand on applique une transformations géométriques à une image, le calcul des coordonnées des pixels peuvent ne pas être valides, c'est à dire qu'elles peuvent être réelles. Pour corriger ce problème, nous devons trouver une façon de faire afin que les coordonnées des pixels soient entières. Nous utiliserons ici la méthode des plus proches voisins