Sensibilité à la translation

17 octobre 2023

Le but du présent document est d'illustrer la mise en œuvre algorithmique d'un calcul de sensibilité géométrique sur l'exemple simple d'un parallélépipède (figure 1). Nous nous intéressons ici à la déformation géométrique liée à la translation de sa paroi supérieure et étudions l'impact de cette translation sur le flux reçu par un récepteur situé sur sa paroi inférieure. Dans cet exercice, la sensibilité du flux est estimée par un algorithme de Monte Carlo analogue à la physique du transport de la sensibilité géométrique. Cette pratique des algorithmes de Monte Carlo est largement utilisée en transferts radiatifs et consiste à imiter numériquement le transport de photons, en échantillonnant les sources du problème, puis en les propageant dans le milieu selon ses propriétés et la statistique donnée par les lois d'extinctions (Beer-Lambert) et de diffusions (fonction de phase). L'extension de cette pratique à la sensibilité géométrique est possible en s'appuyant sur le modèle de sensibilité, qui décrit les sources de sensibilité géométrique et la phénoménologie de leur transport dans le milieu. La démarche suivie dans ce document est la suivante :

- le problème de sensibilité géométrique est posé ;
- les sources de sensibilité sont identifiées, elles dépendent de quantité décrites par d'autres physiques (*i.e.* la luminance) ;
- les couplages par les sources sont exposés ;
- un algorithme de Monte Carlo analogue est utilisé pour résoudre le problème couplé (via la *double randomization*) ;
- la mise en œuvre de l'algorithme est explicite tout au long du document.

L'exemple présenté dans ce document est illustratif et sans aucune ambition de généralité. Il montre une application du modèle de sensibilité et une utilisation de la méthode de Monte Carlo qui lui sont spécifiques. Cela se traduit dans les choix de notations des variables (indice h pour paroi du haut, d pour paroi de droite *etc.*), dans la simplification des équations du modèle, et enfin dans l'écriture de l'algorithme de Monte Carlo. En effet, contrairement à une pratique plus conventionnelle, l'ensemble des données qui décrivent le système (la configuration géométrique et ses propriétés physiques) ne sont pas séparées de la procédure d'échantillonnage des chemins. Autrement dit, l'ensemble des chemins seront suivis relativement aux parois de la scène et à leurs propriétés physiques.

1 Description du problème

L'observable radiative de notre problème est le flux φ perçu par le récepteur et s'exprime comme ci-dessous :

$$\varphi = \int_{A_r} dS \int_{\mathcal{H}^-} d\vec{\omega}\ (\vec{\omega} \cdot \vec{n}) L(\vec{x}, \vec{\omega}, \ddot{\pi}) \tag{1}$$

avec $\ddot{\pi}$ le paramètre géométrique, A_r la surface du récepteur et $\mathcal{H}^-$ l'hémisphère orientée par $\vec{n} = -\vec{e}_z$ (figure 1).

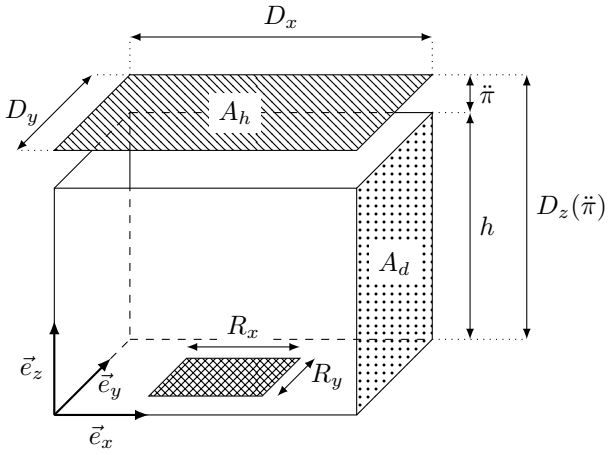

FIGURE 1 – **La configuration géométrique** est un parallélépipède de dimension $D_x \times D_y \times D_z(\ddot{\pi})$, avec $D_z(\ddot{\pi}) = h + \ddot{\pi}$, repéré dans la base $(\vec{e}_x, \vec{e}_y, \vec{e}_z)$. Le paramètre $\ddot{\pi}$ est le paramètre géométrique défini sur $\mathbb{R}^+$. En le modifiant, la position de la paroi supérieure est translatée vers le haut, ou dit autrement "la boîte s'ouvre". Les parois latérales ne dépendent pas de $\ddot{\pi}$. Enfin, le récepteur de surface A_r est positionné sur la paroi du bas et a pour dimension $R_x \times R_y$.

La configuration radiative Les parois du parallélépipède sont toutes noires à l'exception de la paroi du haut (de surface $A_h = D_x \times D_y$) qui est froide, spéculaire, et dont le coefficient de réflexion ρ est donné par :

$$\rho(\vec{x}, -\vec{\omega}) = 0.25\Big[1 - \cos\Big(2\pi\frac{x}{Dx}\Big)\Big]\Big[1 - \cos\Big(2\pi\frac{y}{Dy}\Big)\Big] \tag{2}$$

avec les constantes D_x et D_y décrites en figure 1. Nous ne discutons pas ici du profil spécifique de ρ mais soulignons néanmoins qu'il simplifie le problème et permet ainsi de resserrer le présent document sur le seul développement d'un algorithme analogue de calcul de sensibilité.

En ce qui concerne les sources, seule la paroi noire de droite est émettrice. Sa surface vaut $A_d = D_y \times h$ et la condition à la limite en luminance correspondante est décrite par :

$$L(\vec{x}, \vec{\omega}, \ddot{\pi}) = S_b(\vec{x}) \qquad \vec{x} \in A_d \ ; \ \vec{\omega} \cdot \vec{n}_d > 0 \tag{3}$$

avec $\vec{n}_d$ la normale de la paroi de droite et S_b la source radiative de surface qui correspond ici à son émission thermique :

$$S_b(\vec{x}) = L^{eq}(T)\Big[1 - \cos\Big(2\pi \frac{x}{Dx}\Big)\Big]\Big[1 - \cos\Big(2\pi \frac{y}{Dy}\Big)\Big] \tag{4}$$

Enfin, le milieu englobant notre "boîte" est considéré comme étant froid et transparent.

La sensibilité géométrique du flux On s'intéresse à la sensibilité du flux (équation 1) par rapport à la variation de la position de la paroi spéculaire :

$$\partial_{\ddot{\pi}} \varphi = \int_{A_r} dS \int_{\mathcal{H}^-} d\vec{\omega} \ (\vec{\omega} \cdot \vec{n}) \partial_{\ddot{\pi}} L(\vec{x}, \vec{\omega}, \ddot{\pi}) \tag{5}$$

Pour évaluer le flux φ nous avons besoin de connaître la luminance $L(\vec{x}, \vec{\omega}, \ddot{\pi})$, dans toutes les directions entrantes et en tout point du récepteur. De façon similaire, pour évaluer $\partial_{\ddot{\pi}} \varphi$ nous avons besoin de connaître la sensibilité géométrique $\partial_{\ddot{\pi}} L(\vec{x}, \vec{\omega}, \ddot{\pi})$, dans toutes les directions entrantes et en tout point du récepteur.

La suite du document se concentre sur l'évaluation de cette sensibilité en commençant par énnoncer le modèle physique qui décrit les sources et le transport de la sensibilité géométrique (section 2). Nous développons alors un algorithme Monte Carlo pour résoudre le problème que nous venons de poser en suivant la propagation des sources de sensibilité via l'échantillonnage de chemins qui partent directement de ces sources, en l'occurrence ici la seule paroi du haut (section 3).

2 Modèle de sensibilité géométrique

La sensibilité géométrique de la luminance est définie telle que :

$$s(\vec{x}, \vec{\omega}, \ddot{\pi}) = \partial_3 L(\vec{x}, \vec{\omega}, \ddot{\pi}) = \partial_{\ddot{\pi}} L(\vec{x}, \vec{\omega}, \ddot{\pi}) \tag{6}$$

Elle est considérée comme une quantité physique à part entière dont la phénoménologie est décrite par une équation de transfert radiatif dans le domaine et par des contraintes aux frontières (conditions aux limites) sur la sensibilité entrante dans le domaine.

Équation de transport Dans [Lapeyre et al., 2022] l'équation de la sensibilité dans le milieu est donnée en toute généralité. Dans notre exemple le milieu est transparent ($k_a = k_s = 0$) et peut donc se résumer à l'équation 7, avec $s = s(\vec{x}, \vec{\omega}, \ddot{\pi})$:

$$\vec{w} \cdot \vec{\nabla} s = 0 \tag{7}$$

Les conditions aux limites Toujours dans [Lapeyre et al., 2022] la condition à la limite de la sensibilité géométrique est donnée en toute généralité et dans les cas spécifiques des parois noires, parois spéculaires et parois diffuses. Dans notre configuration, seule la paroi supérieure du parallélépipède est paramétrée par $\ddot{\pi}$. Elle est donc la seule source de sensibilité géométrique :

$$\begin{aligned} s(\vec{x},\vec{\omega},\ddot{\pi}) &= 0 \qquad\qquad \vec{x} \notin A_h \\ s(\vec{x},\vec{\omega},\ddot{\pi}) &= S_{b,\ddot{\pi}} + \rho(\vec{x},-\vec{\omega}) s(\vec{x},\vec{\omega}_{spec},\ddot{\pi}) \qquad\qquad \vec{x} \in A_h \end{aligned} \tag{8}$$

avec $S_{b,\ddot{\pi}}$ la source de sensibilité et $\rho(\vec{x},-\vec{\omega})s(\vec{x},\vec{\omega}_{spec},\ddot{\pi})$ la réflection de la sensibilité incidente à la paroi dans la direction spéculaire. Dans notre exemple, le milieu est transparent et toutes les autres conditions aux limites de sensibilité sont nulles. Il n'y a donc pas de sensibilité géométrique incidente à la paroi du haut spéculaire. La source de sensibilité $S_{b,\ddot{\pi}}$ est donc définie comme ci-dessous en renvoyant le lecteur à l'annexe A pour les développements qui mènent à cette expression :

$$\begin{aligned} S_{b,\ddot{\pi}} = &- \beta_{\vec{\chi},h}[\partial_{1,\vec{u}_h}\ \rho(\vec{x},-\vec{\omega})] L(\vec{x},\vec{\omega}_{spec},\ddot{\pi}) \qquad\qquad \vec{x} \in A_h\ ;\ \vec{\omega}\cdot\vec{n}_h > 0 \\ &+ \rho(\vec{x},-\vec{\omega})\partial_{1,\vec{\chi}} L(\vec{x},\vec{\omega}_{spec},\ddot{\pi}) \\ &- \rho(\vec{x},-\vec{\omega})\beta_{\vec{\chi},h}\partial_{1,\vec{u}_h} L(\vec{x},\vec{\omega}_{spec},\ddot{\pi}) \end{aligned} \tag{9}$$

avec $\vec{\omega}_{spec} = \vec{\omega} - 2(\vec{\omega}\cdot\vec{n}_h)\vec{n}_h$ et $\beta_{\vec{\chi},h}$ le coefficient issu de la décomposition de $\vec{\chi}$ en deux vecteurs, l'un orienté par $\vec{\omega}$ et l'autre orienté par un vecteur $\vec{u}_h$ tangent à la paroi du haut (voir annexe B). Enfin $\beta_{\vec{\chi},h}$ est la norme du vecteur $\vec{\chi}$ projeté sur $\vec{u}_h$. La dérivée spatiale $\partial_{1,\vec{\gamma}} f(\vec{x},\vec{\omega}) = \vec{\gamma}\cdot\vec{\nabla}_{\vec{x}} f(\vec{x},\vec{\omega})$ est la dérivée directionnelle dans la direction $\vec{\gamma}$.

La source de sensibilité est dans notre cas une source de surface, émise par la paroi du haut et donnée par la condition à la limite décrite par l'expression 9. Dans cette équation on note que la condition à la limite de sensibilité dépend de :

- la luminance incidente à la paroi dans la direction de transport spéculaire ;
- la dérivée spatiale de la luminance dans la direction de dérivation $\vec{u}$, incidente à la paroi dans la direction de transport spéculaire ;
- et la dérivée spatiale de la luminance dans la direction de dérivation $\vec{\chi}$, incidente à la paroi dans la direction de transport spéculaire.

En résumé, la source de sensibilité émise par la paroi spéculaire est le résultat du couplage entre le modèle de sensibilité, le modèle de transfert radiatif et le modèle de dérivée spatiale. La dérivée spatiale de la luminance est simplement considérée comme une quantité physique, au même titre que la sensibilité géométrique (voir [Lapeyre et al., 2022] pour la description de son modèle). Résoudre notre problème de sensibilité géométrique revient donc à résoudre un problème de transport couplé qui dépend à la fois des source radiatives (à travers $L(\vec{x},\vec{\omega}_{spec},\ddot{\pi})$), des sources de dérivées spatiales dans la direction $\vec{u}$ (à travers $\partial_{1,\vec{u}} L(\vec{x},\vec{\omega}_{spec},\ddot{\pi})$) et des sources de dérivées spatiales dans la direction $\vec{\chi}$ (à travers $\partial_{1,\vec{\chi}} L(\vec{x},\vec{\omega}_{spec},\ddot{\pi})$).

Les sources du problème couplé La seule source radiative de notre configuration est donnée en section 1 par l'équation 3. Elle correspond à l'émission thermique S_b de la paroi de droite de surface A_d. Pour les sources de dérivée spatiale, le modèle de dérivée spatiale élaboré dans [Lapeyre et al., 2022] autorise des sources volumiques, des sources de surfaces et des sources locales situées sur les arrêtes d'une géométrie triangulées. Dans notre exemple ce modèle se simplifie de sorte que ces sources se résument aux seules sources émises par la surface du haut A_h et la surface de droite A_d (figure 1).

Pour la dérivée spatiale dans la direction $\vec{u}_h$, la source de la paroi du haut est donnée par la condition à la limite :

$$\partial_{1,\vec{u}_h} L(\vec{x},\vec{\omega},\ddot{\pi}) = \beta_{\vec{u}_h,h}[\partial_{1,\vec{u}_s}\rho(\vec{x},-\vec{\omega})]L(\vec{x},\vec{\omega}_{spec},\ddot{\pi}) \qquad \vec{x}\in A_h \ ; \ \vec{\omega}\cdot\vec{n}_h > 0 \tag{10}$$

et la source de la paroi de droite est donnée par la condition à la limite :

$$\partial_{1,\vec{u}_h} L(\vec{x},\vec{\omega},\ddot{\pi}) = \beta_{\vec{u}_h,d}\partial_{1,\vec{u}_{hd}} S_b(\vec{x}) \qquad \vec{x}\in A_d \ ; \ \vec{\omega}\cdot\vec{n}_d > 0 \tag{11}$$

Pour la dérivée spatiale dans la direction $\vec{\chi}$, la source de la paroi du haut est donnée par la condition à la limite :

$$\partial_{1,\vec{\chi}} L(\vec{x},\vec{\omega},\ddot{\pi}) = \beta_{\vec{\chi},h}[\partial_{1,\vec{u}_h}\rho(\vec{x},-\vec{\omega})]L(\vec{x},\vec{\omega}_{spec},\ddot{\pi}) \qquad \vec{x}\in A_h \ ; \ \vec{\omega}\cdot\vec{n}_h > 0 \tag{12}$$

et la source de la paroi de droite est donnée par la condition à la limite :

$$\partial_{1,\vec{\chi}} L(\vec{x},\vec{\omega},\ddot{\pi}) = \beta_{\vec{\chi},d}\partial_{1,\vec{u}_{\chi,d}} S_b(\vec{x}) \qquad \vec{x}\in A_d \ ; \ \vec{\omega}\cdot\vec{n}_d > 0 \tag{13}$$

3 Résolution par Monte Carlo

Dans cette section nous écrivons un algorithme de Monte Carlo pour résoudre le problème décrit en section 1 à partir du modèle de sensibilité géométrique décrit en section 2. Notre algorithme est construit de manière analogue avec pour support la mise en œuvre pratique de sa ⟨*Fonction de réalisation* 5⟩ en langage C [Kernighan, 1988]. Pour cela, nous échantillonnerons des chemins qui démarrent directement de la source de sensibilité, à savoir la paroi du haut, et qui la propage jusqu'au récepteur positionné sur la paroi du bas. La mise en œuvre ici proposée est néanmoins particulière dans le sens où chaque chemin du problème couplé est d'abord échantillonné et conservé en totalité (section 3.1). Son poids n'est calculé qu'a posteriori à partir du chemin ainsi construit (section 3.2). Cette proposition diffère d'un algorithme Monte Carlo plus conventionnel où la position et la direction courante du chemin ne sont que des données locales à chaque étape de sa construction ; son poids étant mis à jour si nécessaire à chacune de ces étapes. Ce choix singulier est dicté par la volonté de garder à tout instant une vue d'ensemble du problème pour s'éviter un travail d'abstraction et ainsi faciliter l'écriture analogue de notre algorithme. Une démarche adaptée en raison de la configuration simplifiée à laquelle nous nous intéressons.

5 ⟨*Fonction de réalisation* 5⟩≡

```
static res_T
realisation
  (struct ssp_rng* rng,
   const struct sgs_scene* scene,
   double* w)
{
  ⟨Données locales à la fonction de réalisation 11a⟩
  res_T res = RES_OK;

  ⟨Initialiser le poids 7c⟩

  ⟨Échantillonner un chemin du problème couplé 6a⟩
  ⟨Calcul du poids 8d⟩

exit:
  ⟨Nettoyer les données locales de la fonction 11b⟩
  ⟨Renvoyer le poids 10c⟩
  return res;
}
```

Notre fonction de réalisation prend en entrée un générateur de nombres aléatoires (`rng`) et un pointeur vers les données du système (`scene`). Dans la variable `w` sera renvoyé le poids de la sensibilité a $\ddot{\pi}$.

3.1 Le chemin

Pour construire un chemin du problème couplé on échantillonne d'abord un chemin de sensibilité partant d'une position quelconque sur la surface de la source de sensibilité A_h. Ce chemin est alors complété par l'échantillonnage d'un chemin de dérivée spatiale dont la couplage avec le chemin de sensibilité forme un chemin du problème couplé.

6a ⟨*Échantillonner un chemin du problème couplé* 6a⟩≡

```
⟨Échantillonner une position sur la source de sensibilité 6b⟩
⟨Échantillonner un chemin de sensibilité 7a⟩
⟨Échantillonner un chemin de dérivée spatiale 7b⟩
```

Comme point de départ du chemin du problème couplé, on commence donc par échantillonner uniformément un point sur la surface émettrice de sensibilité à l'aide de la fonction `sgs_geometry_sample_sensitivity_source`, et on stocke dans les variables `pos_h` et `normal_h` sa position et la normale correspondante. On récupère également dans `surf_A_h` l'identifiant de la surface que l'on vient d'échantillonner, dans notre cas la surface supérieure A_h identifié dans le code par la constante `SGS_SURFACE_Z_MAX` (voir figure 1).

6b ⟨*Échantillonner une position sur la source de sensibilité* 6b⟩≡

```
/* Échantillonner uniformément une position sur la source de sensibilité */
sgs_geometry_sample_sensitivity_source(scene->geom, rng, &frag);
d3_set(pos_h, frag.position);
d3_set(normal_h, frag.normal);
surf_A_h = frag.surface; /* surf_A_h == SGS_SURFACE_Z_MAX */
```

On rappelle que les sources de sensibilité proviennent des parois perturbées par une modification du paramètre $\ddot{\pi}$. En toute hypothèse, toute source de sensibilité incidente à A_h serait réfléchie de façon spéculaire. Or, dans notre cas, nous n'avons qu'une seule source de sensibilité, la surface A_h elle-même, et par conséquent nous n'avons pas à tenir compte de ces sensibilités réfléchies. Nous n'échantillons donc que le seul chemin qui propage l'émission de sensibilité par A_h. Ce chemin sera notre chemin de sensibilité. Pour cela, il suffit d'échantillonner une direction d'émission lambertienne `dir_emit_h` autour de la normale `normal_h` de la surface A_h, et de lancer un rayon dans cette direction. Nous stockons alors dans `hit0` l'intersection de ce rayon avec la géométrie de la scène.

7a ⟨*Échantillonner un chemin de sensibilité* 7a⟩≡
```
/* Échantillonner une direction d'émission de sensibilité  */
ssp_ran_hemisphere_cos(rng, normal_h, dir_emit_h, NULL);
/* Lancer le rayon qui propage l'émission de sensibilité */
TRACE_RAY(pos_h, dir_emit_h, surf_A_h, &hit0);
```

La source de sensibilité est donnée dans la condition à la limite décrite par l'équation 9. Elle dépend de la dérivée spatiale selon $\vec{\chi}$ incidente dans la direction spéculaire $\vec{\omega}_{spec}$ et de la dérivée spatiale selon $\vec{u}$ incidente à la même direction spéculaire. Ces contributions à l'émission de sensibilité ne sont pas connues et sont ici échantillonnées par *double randomization*. Comme ces deux dérivées spatiales sont incidentes à la même direction $\vec{\omega}_{spec}$ nous pouvons nous contenter de ne suivre qu'un seul chemin dans cette direction. Ce chemin sera notre chemin de dérivée spatiale. Pour échantillonner ce chemin nous calculons d'abord $\vec{\omega}_{spec}$ (`dir_spec_h`) par réflexion spéculaire de la direction d'émission $\vec{\omega}$ (`dir_emit_h`) avant de lancer un rayon dans cette direction jusqu'à l'intersection `hit1` avec une surface .

7b ⟨*Échantillonner un chemin de dérivée spatiale* 7b⟩≡
```
/* Calculer la direction spéculaire à dir_emit_h */
reflect(dir_spec_h, dir_emit_h, normal_h);
/* Tracer le rayon spéculaire partant de surf_A_h */
TRACE_RAY(pos_h, dir_spec_h, surf_A_h, &hit1);
```

3.2 Le poids

Dans notre problème, un chemin du problème couplé est composé d'un chemin de sensibilité et d'un chemin de dérivée spatiale, chacun d'entre eux se résumant à un segment dont l'origine commune se situe sur la paroi du haut, source de sensibilité. Or, on peut dès à présent déterminer qu'un chemin couplé aura une contribution nulle si le chemin de sensibilité n'atteint pas le récepteur ou si le chemin de dérivé spatiale n'atteint pas la source radiative, à savoir la paroi de droite.

7c ⟨*Initialiser le poids* 7c⟩≡
```
sensib = 0;
```

8a ⟨*Échantillonner un chemin de sensibilité* 7a⟩+≡
```
if(!hit_receiver(scene, pos_h, dir_emit_h, &hit0)) {
  goto exit;
}
```

8b ⟨*Échantillonner un chemin de dérivée spatiale* 7b⟩+≡
```
if(!hit_source(scene, pos_h, dir_spec_h, &hit1)) {
  goto exit;
}
```

En conséquence nous pouvons assumer dans la suite de la fonction que nous n'aurons à calculer le poids que des seuls chemin couplés dont la contribution est non nulle. Dès lors, `hit1` représente une intersection sur la source radiative. On stocke dans `normal_d` la normale de la paroi correspondante dont on aura besoin pour le calcul du poids. De même, on initialise la variable `dir_emit_d` à $-\vec{\omega}_s$, cette direction nous sera également utile pour évaluer la source de dérivée spatiale.

8c ⟨*Échantillonner un chemin de dérivée spatiale* 7b⟩+≡
```
d3_normalize(normal_d, hit1.normal);
d3_minus(dir_emit_d, dir_spec_h);
```

Dans notre problème couplé, la contribution du chemin, *i.e.* le poids Monte Carlo, va s'exprimer à travers la condition à la limite de sensibilité (équation 9) et des sources de chacun de ses couplages.

8d ⟨*Calcul du poids* 8d⟩≡
⟨*Décomposition du vecteur de déformation* $\vec{\chi}$ 8e⟩

⟨*Calcul de la dérivée surfacique de* ρ 8f⟩
⟨*Calcul des sources de dérivées spatiales* 9b⟩

⟨*Calculer le poids de sensibilité* 10b⟩

La décomposition du vecteur de déformation $\vec{\chi}$ permet d'obtenir le vecteur tangent $\vec{u}$ nécessaire dans l'expression de la source de sensibilité et de ses dérivées surfaciques (équation 9).

8e ⟨*Décomposition du vecteur de déformation* $\vec{\chi}$ 8e⟩≡
```
decomposition(chi, normal_h, dir_emit_h, &proj_chi_h);
```

Étant donné que le coefficient de réflection n'est défini qu'en frontière, à savoir sur un plan en deux dimensions, calculer la dérivée surfacique de ρ revient à travailler dans le plan. Et cette dérivée surfacique est le produit scalaire entre le gradient surfacique de ρ et la direction de dérivation $\vec{u}$ transformée dans ce plan (`u_2d`).

8f ⟨*Calcul de la dérivée surfacique de* ρ 8f⟩≡
⟨*Récupérer le gradient surfacique de* ρ 9a⟩

```
/* Transformer u dans le plan XY */
u_2d[0] = proj_chi_h.u[X];
u_2d[1] = proj_chi_h.u[Y];
```

```
/* Calculer la dérivée surfacique de rho */
d_rho = d2_dot(grad_rho_2d, u_2d);
```

Pour récupérer ρ et son gradient, il nous suffit de transformer dans le plan la position d'émission `pos_h`, et d'interroger les données associées à la position ainsi transformée (`pos_h_2d`).

9a ⟨*Récupérer le gradient surfacique de* ρ 9a⟩≡

```
/* Transformer pos_h dans le plan XY */
pos_h_2d[0] = pos_h[X];
pos_h_2d[1] = pos_h[Y];

rho = get_rho(scene, pos_h_2d);
get_grad_rho(scene, pos_h_2d, grad_rho_2d);
```

On rappelle que la sensibilité est couplée à deux dérivées spatiales (selon $\vec{\chi}$ et $\vec{u}$) dont les sources sont données par les équations 10, 11, 12 et 13.

9b ⟨*Calcul des sources de dérivées spatiales* 9b⟩≡

⟨*Décomposition du vecteur* $\vec{\chi}$ 9c⟩
⟨*Décomposition du vecteur* $\vec{u}$ 9d⟩
⟨*Calcul de la dérivée surfacique de Sb dans la direction* $\vec{u}_e$ 9e⟩
⟨*Calcul de la dérivée surfacique de Sb dans la direction* $\vec{u}_{cs}$ 10a⟩

Sur la paroi de droite, la décomposition du vecteur de déformation $\vec{\chi}$ permet d'obtenir le vecteur tangent $\vec{u}$ nécessaire dans l'expression de la source de de la dérivée spatiale dans la direction $\vec{\chi}$ et de sa dérivée surfacique (équation 13).

9c ⟨*Décomposition du vecteur* $\vec{\chi}$ 9c⟩≡

```
decomposition(chi, normal_d, dir_emit_d, &proj_chi_d);
```

De la même façon, la décomposition du vecteur $\vec{u}$ permet d'obtenir le vecteur tangent $\vec{u}_e$ nécessaire dans l'expression de la source de la dérivée spatiale dans la direction $\vec{u}$ et de sa dérivée surfacique (equation 11).

9d ⟨*Décomposition du vecteur* $\vec{u}$ 9d⟩≡

```
decomposition(proj_chi_h.u, normal_d, dir_emit_d, &proj_uh_d);
```

La dérivée surfacique de S_b dans la direction $\vec{u}_e$ (`dSb_uhd`) est le produit scalaire entre le gradient surfacique de S_b et la direction de dérivation $\vec{u}_e$ transformée dans le plan de la paroi de droite (`u_hd_2d`). Mais pour effectuer ce calcul nous devons au préalable récupérer le gradient de S_b (`grad_Sb_2d`). Pour cela il nous suffit là encore de transformer dans le plan de la paroi de droite la position d'émission `pos_d`, et d'interroger les données associées à la position ainsi transformée (`pos_d_2d`). Et nous en profitons au passage pour récupérer S_b qui nous sera utile pour le calcul du poids. Pour rappel, dans notre configuration les deux dérivées spatiales $\partial_{\vec{\chi}} I$ et $\partial_{\vec{u}} I$ partagent une même position d'émission (`pos_d`).

9e ⟨*Calcul de la dérivée surfacique de Sb dans la direction* $\vec{u}_e$ 9e⟩≡

```
/* Calculer la position sur l'émetteur */
pos_d[X] = pos_h[X] + dir_spec_h[X]*hit1.distance;
pos_d[Y] = pos_h[Y] + dir_spec_h[Y]*hit1.distance;
```

```
pos_d[Z] = pos_h[Z] + dir_spec_h[Z]*hit1.distance;

/* Transformer pos_d dans le plan YZ */
pos_d_2d[0] = pos_d[Y];
pos_d_2d[1] = pos_d[Z];

/* Récupérer le gradient surfacique de Sb */
Sb = get_Sb(scene, pos_d_2d);
get_grad_Sb(scene, pos_d_2d, grad_Sb_2d);

/* Transformer u_e dans le plan YZ */
u_hd_2d[0] = proj_uh_d.u[Y];
u_hd_2d[1] = proj_uh_d.u[Z];

/* Dérivée surfacique de Sb dans la direction u_e */
dSb_uhd = d2_dot(grad_Sb_2d, u_hd_2d);
```

Ne reste plus qu'à calculer la dérivée surfacique de S_b dans la direction $\vec{u}_{cs}$ (`dSb_uchid`) comme étant le produit scalaire entre le gradient surfacique de S_b et la direction $\vec{u}_{cs}$ transformée dans le plan de la paroi de droite (`u_chid_2d`).

10a ⟨*Calcul de la dérivée surfacique de Sb dans la direction* $\vec{u}_{cs}$ 10a⟩≡

```
/* Transformer u_cs dans le plan YZ */
u_chid_2d[0] = proj_chi_d.u[Y];
u_chid_2d[1] = proj_chi_d.u[Z];

/* Dérivée surfacique de Sb dans la direction u_cs */
dSb_uchid = d2_dot(grad_Sb_2d, u_chid_2d);
```

On est alors en mesure d'évaluer le poids de sensibilité comme la contribution portée par le chemin du problème couplé, soit les sources de dérivées spatiales propagées jusqu'à la paroi du haut puis intégrées dans les sources de sensibilité propagées ensuite vers le récepteur. Cette contribution est ensuite multipliée par la surface A_h et l'angle π (`PI`) étant donné l'échantillonnage des densités de probabilités de la position sur la surface et de la direction dans l'hémisphère sortante de la paroi.

10b ⟨*Calculer le poids de sensibilité* 10b⟩≡

```
/* Calcul de la contribution du chemin couplé */
Sb_sensib =
  - proj_chi_h.beta * d_rho * Sb
  - rho * proj_chi_h.beta * proj_uh_d.beta * dSb_uhd
  + rho * proj_chi_d.beta * dSb_uchid;

/* Poids de sensibilité */
sensib = Sb_sensib * PI * get_Sr(scene);
```

10c ⟨*Renvoyer le poids* 10c⟩≡

```
w[0] = sensib;
```

3.3 Variables locales et macro

Lors de l'échantillonnage des chemins (section 3.1) nous nous sommes appuyé sur la fonction `TRACE_RAY` nous permettant de lancer un rayon dans la scène. Cette fonction est en réalité un macro, locale à la fonction, qui encapsule l'appel à la fonction qui lance un rayon. En plus de l'origine et de la direction du rayon, cette fonction nécessite en entrée une plage des distances d'intersection possible (`range`), dans notre cas toujours définie à $[0, \infty]$. À noter également le paramètre d'entrée `StartFrom` qui stocke le triangle sur lequel se trouve l'origine du rayon, une donnée d'entrée utilisée pour éviter une auto-intersection, c'est à dire l'intersection du rayon avec le triangle dont il est issu.

11a ⟨*Données locales à la fonction de réalisation* 11a⟩≡
```
/* Macro utilisée comme sucre syntaxique */
#define TRACE_RAY(Org, Dir, StartFrom, Hit) {                          \
  double range[2];                                                     \
  range[0] = 0;                                                        \
  range[1] = INF;                                                      \
  sgs_geometry_trace_ray                                               \
    (scene->geom, (Org), (Dir), range, (StartFrom), (Hit));\
} (void)0
```

11b ⟨*Nettoyer les données locales de la fonction* 11b⟩≡
```
#undef TRACE_RAY
```

Enfin, nous déclarons ci-après l'ensemble des variables locales nécessaires à notre ⟨*Fonction de réalisation* 5⟩ :

11c ⟨*Données locales à la fonction de réalisation* 11a⟩+≡
```
/* Variables de la source de sensibilité */
double dir_emit_h[3];
double pos_h[3];
double normal_h[3];
double dir_spec_h[3];
double pos_h_2d[2];
enum sgs_surface_type surf_A_h;
struct sgs_fragment frag; /* Position échantillonnée */

/* Variables de la source radiative */
double pos_d[3];
double normal_d[3];
double dir_emit_d[3];
double pos_d_2d[2];

/* Propriétés radiatives des surfaces et leur dérivée */
double rho;
double d_rho;
double grad_rho_2d[2];
double Sb;
double dSb_uhd;
```

```
double dSb_uchid;
double grad_Sb_2d[2];

/* Vecteurs des déformations et variables de projection */
const double chi[3] = {0, 0, 1};
double u_2d[2];
double u_hd_2d[2];
double u_chid_2d[2];
struct projection proj_chi_h;
struct projection proj_chi_d;
struct projection proj_uh_d;

/* Pour le calcul du poids */
double Sb_sensib;
double sensib;

/* Intersections avec la géométrie */
struct sgs_hit hit0;
struct sgs_hit hit1;
```

4 Résultats

Au delà du simple calcul de sensibilité nous proposons ici une vérification des résultats par le calcul de différences finies. La différentiation est effectuée à partir des estimations du flux $\varphi(\ddot{\pi})$ reçu par le capteur pour différentes valeurs du paramètre géométrique $\ddot{\pi}$, soit pour différentes hauteurs de la paroi spéculaire. Le calcul du poids associé au calcul du flux est décrit en annexe C. La figure 2 présente les estimations de la sensibilité du flux et des différences finies correspondantes. Les paramètres des simulations Monte Carlo et des calculs en différences finies, et plus généralement le code source des scripts à l'origine de ces résultats sont données en annexe F.

A Détails de la condition à la limite de sensibilité

Nous récupérons ici la condition à la limite pour une paroi spéculaire donnée dans [Lapeyre et al., 2022] $s = s(\vec{x}, \vec{\omega}, \ddot{\pi})$:

$$s = \mathcal{C}_b[s] + S_{b,\ddot{\pi}}[L, \partial_{1,\vec{u}} L, \partial_{1,\vec{\chi}} L, \partial_{2,\vec{\gamma}_t} L] \qquad \vec{x} \in \partial G(\ddot{\pi}); \vec{\omega} \cdot \vec{n} > 0 \tag{14}$$

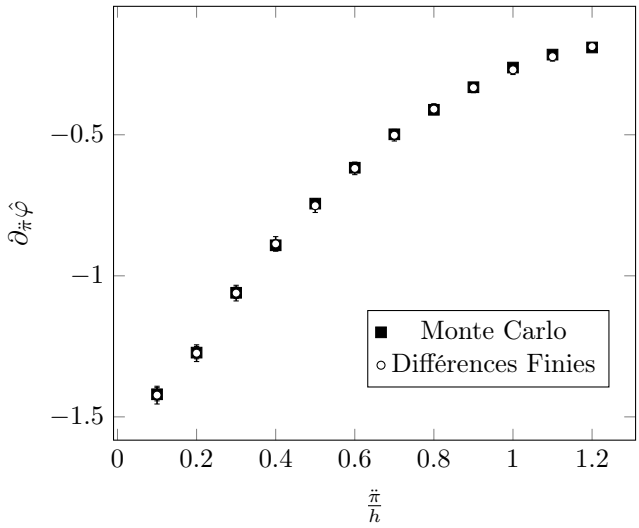

FIGURE 2 – Estimations des sensibilités du flux par Monte Carlo et comparaison avec les différences finies du flux. Les résultats sont adimensionnalisés et affichés en fonction de l'ouverture du parallélépipède, qui est paramétrée par $\ddot{\pi}$. La sensibilité du flux $\partial_{\ddot{\pi}}\hat{\varphi} = \frac{\partial_{\ddot{\pi}}\varphi}{\varphi_{max}}h$ avec $\varphi_{max} = \varphi_{spec}(\ddot{\pi} = 0)$. Dans cette expression, φ_{spec} correspond uniquement à la partie du flux qui arrive au récepteur après avoir été réfléchie sur la paroi spéculaire (voir annexe C). Le nombre d'échantillonnage du poids de Monte Carlo nécessaire à la reproduction de ce résultat est 10^8.

avec $S_{b,\ddot{\pi}}$ la source surfacique de sensibilité :

$$\begin{aligned} S_{b,\ddot{\pi}} = &- \alpha(\mathcal{C}[L] + S) \\ &- \beta\partial_{1,\vec{u}}S_b - \partial_{2,\vec{\gamma}}S_b + \partial_{\ddot{\pi}}S_b \\ &- \beta\partial_{1,\vec{u}}\mathcal{C}_b[L] + \partial_{\ddot{\pi}}\mathcal{C}_b[L] \\ &- \partial_{2,\vec{\gamma}}\rho(\vec{x}, -\vec{\omega}) \int_{H'} p_{\Omega'}(-\vec{\omega}'|\vec{x}, -\vec{\omega}) d\vec{\omega}' L \\ &- \beta\mathcal{C}_b[\partial_{1,\vec{u}}L] + \mathcal{C}_b[\partial_{1,\vec{\chi}}L] \\ &+ 2\mu\partial_{2,\vec{\gamma}_t}L(\vec{x}, \vec{\omega}_{spec}, \ddot{\pi}) \end{aligned} \tag{15}$$

Dans cette équation $\mathcal{C}$ est l'opérateur collisionnel du milieu :

$$\mathcal{C}[L] = -k_a L(\vec{x}, \vec{\omega}, \ddot{\pi}) - k_s L(\vec{x}, \vec{\omega}, \ddot{\pi}) + k_s \int_{\mathcal{S}} p_{\Omega'}(\vec{\omega}'|\vec{x}, \vec{\omega}) d\vec{\omega}' L(\vec{x}, \vec{\omega}', \ddot{\pi}) \tag{16}$$

La source S est la source radiative du milieu (émission thermique $k_a L^{eq}(T)$). On trouve également $\mathcal{C}_b$ l'opérateur collisionnel de la surface. Appliqué à la

luminance et dans le cas d'une paroi spéculaire il s'écrit :

$$\mathcal{C}_b[L] = \rho(\vec{x}, -\vec{\omega}) \int_{H'} \delta(\vec{\omega}' - \vec{\omega}_{spec}) L(\vec{x}, \vec{\omega}', \ddot{\pi}) d\vec{\omega}' \tag{17}$$

avec $\vec{\omega}_{spec} = \vec{\omega} - 2(\vec{\omega} \cdot \vec{n})\vec{n}$

Condition à la limite de notre exemple Pour commencer, seule la paroi spéculaire est source de sensibilité géométrique. Nous voyons dans l'équation 14 que le terme collisionnel $\mathcal{C}_b[s]$ traduit la réflexion de la sensibilité incidente à la paroi spéculaire. Ce terme est obligatoirement nul puisque il n'y a aucune autre source qui pourrait émettre une sensibilité géométrique et aucune autre paroi réfléchissante qui pourrait réfléchir la sensibilité émise par la paroi spéculaire.

Dans notre exemple le milieu est transparent, les termes $\mathcal{C}[L]$ et S sont donc nuls. La paroi spéculaire est froide, la source surfacique S_b qui dans cet exemple correspondrait à l'émission thermique de la paroi est donc aussi nulle. L'opérateur collisionnel de la surface $\mathcal{C}_b$ est indépendant de $\ddot{\pi}$, la dérivée $\partial_{\ddot{\pi}} \mathcal{C}_b$ est donc nulle. Pour finir la déformation géométrique de la paroi spéculaire est une translation, l'axe de rotation $\vec{\gamma}$ est donc nul et toutes les dérivées angulaires n'ont plus lieux d'être dans la condition à la limite.

La condition à la limite de sensibilité de la paroi spéculaire de la boite devient donc :

$$\begin{aligned} s = &- \beta \partial_{1,\vec{u}} \mathcal{C}_b[L] \\ &+ \mathcal{C}_b[\partial_{1,\vec{\chi}} L - \beta \partial_{1,\vec{u}} L] \end{aligned} \tag{18}$$

avec

$$\beta \partial_{1,\vec{u}} \mathcal{C}_b[L] = \beta \left(\partial_{1,\vec{u}} \rho(\vec{x}, -\vec{\omega}) \right) \int_{H'} \delta(\vec{\omega}' - \vec{\omega}_{spec}) L(\vec{x}, \vec{\omega}', \ddot{\pi}) d\vec{\omega}' \tag{19}$$

En prenant en compte le fait que :

$$\int_{H'} \delta(\vec{\omega}' - \vec{\omega}_{spec}) f(\vec{\omega}') d\vec{\omega}' = f(\vec{\omega}_{spec}) \tag{20}$$

on trouve finalement :

$$\begin{aligned} s(\vec{x}, \vec{\omega}, \ddot{\pi}) = &- \beta \left(\partial_{1,\vec{u}} \rho(\vec{x}, -\vec{\omega}) \right) L(\vec{x}, \vec{\omega}_{spec}, \ddot{\pi}) \\ &+ \partial_{1,\vec{\chi}} L(\vec{x}, \vec{\omega}_{spec}, \ddot{\pi}) - \beta \partial_{1,\vec{u}} L(\vec{x}, \vec{\omega}_{spec}, \ddot{\pi}) \end{aligned} \tag{21}$$

B Décomposition du vecteur de déformation

Dans le modèle de sensibilité la déformation est caractérisée par le vecteur de déformation $\vec{\chi}$. La condition à la limite de sensibilité dépend alors de la dérivée spatiale $\partial_{1,\vec{\chi}} L$ (équation 9). Le champs de luminance n'étant pas connu il n'existe pas de solution analytique à cette dérivée. À la frontière nous avons donc choisi de décomposer la direction $\vec{\chi}$ (`chi`) en deux directions distinctes, une direction tangente à la frontière $\vec{u}$ (`u`) et la direction de transport $\vec{\omega}$ (`omega`). Ainsi la dérivée de L selon $\vec{\chi}$ devient une composition de dérivées de L le long

de la paroi (sur laquelle la luminance est connue) et le long de la direction de transport (retrouvant ainsi le terme de transport de l'ETR).

La décomposition de $\vec{\chi}$ s'écrit :

$$\vec{\chi} = \alpha\vec{\omega} + \beta\vec{u} \tag{22}$$

avec α (`alpha`) et β (`beta`) les coefficients issus de la projection de $\vec{\chi}$ sur $\vec{n}$ (`normal`) et $\vec{u}$. Pour plus de précisions sur la base non-orthogonale utilisée pour cette décomposition, nous renvoyons le lecteur vers [Lapeyre et al., 2022]. Nous nous contentons de donner ici les résultats :

$$\alpha = \frac{\vec{\chi}.\vec{n}}{\vec{\omega}.\vec{n}} \qquad \beta = \|\vec{\chi} - \alpha\vec{\omega}\| \qquad \vec{u} = \frac{\vec{\chi} - \alpha\vec{\omega}}{\beta} \tag{23}$$

Ce qui permet d'obtenir :

$$\partial_{1,\vec{\chi}} L = \alpha\partial_{1,\vec{\omega}} L + \beta\partial_{1,\vec{u}} L \tag{24}$$

et de résoudre la dérivée en estimant $\partial_{1,\vec{u}} L$ le long de la surface et en utilisant l'ETR pour résoudre $\partial_{1,\vec{\omega}} L$:

$$\partial_{1,\vec{\omega}} L = \mathcal{C}[L] \tag{25}$$

La fonction `decomposition` réalise cette décomposition et les résultats (`alpha`, `beta` et `u`) sont retournés via les variables membres de la structure `projection` :

15 ⟨*Fonctions utilitaires* 15⟩≡

```
struct projection {
  double alpha;
  double beta;
  double u[3];
};

static void
decomposition
  (const double chi[3],
   const double normal[3],
   const double omega[3],
   struct projection* projection)
{
  ASSERT(chi && normal && omega && projection);
  ASSERT(d3_is_normalized(normal));
  ASSERT(d3_is_normalized(omega));

  projection->alpha = d3_dot(chi, normal) / d3_dot(omega, normal);
  projection->u[X] = chi[X] - projection->alpha*omega[X];
  projection->u[Y] = chi[Y] - projection->alpha*omega[Y];
  projection->u[Z] = chi[Z] - projection->alpha*omega[Z];
  projection->beta = d3_normalize(projection->u, projection->u);
}
```

C Calcul de la contribution de la luminance qui dépend de $\ddot{\pi}$

Le flux reçu par le capteur est décrit par l'équation 1. Dans notre problème, la luminance $L(\vec{x}, \vec{\omega}, \ddot{\pi})$ incidente au récepteur dépend de deux sortes de chemins radiatifs qui traduisent le transport de la source émise par la paroi de droite :

— soit la source est transportée directement depuis la paroi de droite jusqu'au récepteur ;
— soit la source est transportée en direction de la paroi spéculaire puis réfléchie jusqu'au récepteur.

Dans notre configuration le paramètre géométrique $\ddot{\pi}$ n'a d'influence que sur la hauteur de la paroi spéculaire. Ainsi, la contribution radiative de la source émise par la paroi de droite, transportée directement vers le récepteur, reste identique pour toutes valeurs de $\ddot{\pi}$ (c'est à dire quelle que soit la hauteur de la paroi spéculaire). Du point de vue de la sensibilité cette contribution n'aura donc aucune influence.

En pratique cela signifie que, en vue d'une validation via un calcul des différences finies, le calcul par Monte Carlo du flux au récepteur n'est pas entièrement nécessaire. Nous pouvons nous contenter d'estimer l'unique partie du flux qui sera perturbée par une variation de $\ddot{\pi}$, soit les contributions portées par les chemins réfléchis par la paroi spéculaire. En terme d'algorithme nous pouvons donc réutiliser les chemins déjà échantillonnés pour le problème couplé puisque leur statistique correspond exactement à celle d'un chemin émis par la paroi de droite et réfléchi par la paroi du haut.

Le poids (`weight_flux_part_spec`) correspondant à la partie du flux venant de la réflection sur la paroi spéculaire est donc calculé au même moment que celui de la sensibilité.

16a ⟨*Données locales à la fonction de réalisation* 11a⟩+≡
```
double weight_flux_part_spec;
```

Le poids de cette contribution correspond à la source radiative S_b (equation 3) représentée par la variable `Sb` multipliées par l'angle π (`PI`), la surface A_h et le coefficient de réflection ρ (`rho`).

16b ⟨*Calcul du poids* 8d⟩+≡
```
weight_flux_part_spec = Sb * rho * PI * get_Sr(scene);
```

16c ⟨*Renvoyer le poids* 10c⟩+≡
```
w[1] = weight_flux_part_spec;
```

Ce poids est également initialisé en même temps que celui de la sensibilité de sorte que les chemins réfléchis n'atteignant pas le récepteur aient une contribution nulle.

16d ⟨*Initialiser le poids* 7c⟩+≡
```
weight_flux_part_spec = 0;
```

D Fonction de calcul

Le programme présenté jusqu'alors s'est concentré sur la mise en œuvre de la seule fonction de réalisation de notre algorithme de Monte Carlo ; la boucle d'intégration, l'accumulation des poids ou encore l'affichage des résultats y sont absents. Dans cette section nous détaillons ces étapes manquantes afin de compléter le programme écrit jusqu'ici et ainsi proposer une mise en œuvre complète de l'algorithme de Monte Carlo objet du présent document. Ces étapes sont regroupées dans la fonction qui suit :

17a ⟨*Calculer la sensibilité à la translation* 17a⟩≡

```
res_T
compute_sensitivity_translation(struct sgs* sgs)
{
  ⟨Variables locales au calcul de sensibilité 18b⟩
  res_T res = RES_OK;

  ⟨Exécuter l'intégration Monte Carlo 17b⟩
  ⟨Afficher les résultats de l'estimation 18a⟩

exit:
  ⟨Libérer les variables locales au calcul de sensibilité 19a⟩
  return res;
error:
  goto exit;
}
```

Dans cette fonction on commence par exécuter l'intégration Monte Carlo. Cette étape consiste à invoquer notre ⟨*Fonction de réalisation* 5⟩ autant de fois que de nombre de réalisations demandées, et à accumuler les poids qu'elle retourne. D'apparence triviale, cette simple boucle s'avère plus compliquée pour qui souhaite paralléliser son calcul. Au delà des questions propres à une exécution parallèle, d'aucun doit s'assurer que chaque processus dispose d'une séquence de nombre aléatoires qui lui est propre. C'est pourquoi nous utilisons ici la bibliothèque `Star-MonteCarlo` en charge de cette intégration parallèle. Pour cela nous créons d'abord un système `Star-MonteCarlo`, c'est à dire une variable qui matérialise la bibliothèque à l'échelle de notre programme (`smc`). Et nous le configurons pour qu'il utilise le journal d'évènement (`logger`), l'allocateur mémoire (`allocator`) et le nombre de processus légers (`nthreads`) soumis en entrée de la fonction via la variable `sgs`. Nous configurons enfin le type de générateur aléatoire à utiliser, en l'occurrence le générateur pseudo alétoire Mersenne-Twister [Matsumoto and Nishimura, 1998] (`SSP_RNG_MT19937_64`). Nous pouvons alors lancer l'intégration Monte Carlo à proprement parler. Pour cela nous définissons un intégrateur qui détermine la fonction à appeler à chaque réalisation (`run_realisation`), le type de poids calculés (`smc_doubleN`, *i.e.* un vecteur de réels) et le nombre total de réalisations (`nrealisations`) défini comme paramètre d'entrée via la variable `sgs`.

17b ⟨*Exécuter l'intégration Monte Carlo* 17b⟩≡

```
/* Configurer et créer le système Star-MonteCarlo */
smc_args.logger = &sgs->logger;
smc_args.allocator = sgs->allocator;
smc_args.nthreads_hint = sgs->nthreads;
smc_args.rng_type = SSP_RNG_MT19937_64;
res = smc_device_create(&smc_args, &smc);
if(res != RES_OK) goto error;

/* Configurer l'intégrateur */
integrator.integrand = run_realisation;
integrator.type = &smc_doubleN;
integrator.max_realisations = sgs->nrealisations;
ctx.count = 2; /* Nombre de poids calculés (Sensibilité & "luminance") */
ctx.integrand_data = sgs; /* Données d'entrée de la fonction integrand */

/* Intégration Monte Carlo */
res = smc_solve(smc, &integrator, &ctx, &estimator);
if(res != RES_OK) goto error;
```

En sortie de l'intégration Monte Carlo (fonction `smc_solve`) nous disposons d'un estimateur de nos variables aléatoires, en l'occurrence la sensibilité à la translation et la fraction de la luminance qui dépend du paramètre $\ddot{\pi}$. Nous pouvons dès lors récupérer l'état de notre estimateur pour afficher l'espérance et l'écart type de ces variables.

18a ⟨*Afficher les résultats de l'estimation* 18a⟩≡

```
res = smc_estimator_get_status(estimator, &status);
if(res != RES_OK) goto error;

sgs_log(sgs, "Sensibilité ~ %g +/- %g\n",
  SMC_DOUBLEN(status.E)[0], /* Espérance */
  SMC_DOUBLEN(status.SE)[0]); /* Écart type */

sgs_log(sgs, "Luminance ~ %g +/- %g\n",
  SMC_DOUBLEN(status.E)[1], /* Espérance */
  SMC_DOUBLEN(status.SE)[1]); /* Écart type */
```

Ne reste plus qu'à déclarer les variables locales utilisées par notre fonction de calcul et de libérer en sortie l'espace mémoire allouée dynamiquement pour ces variables.

18b ⟨*Variables locales au calcul de sensibilité* 18b⟩≡

```
/* Système */
struct smc_device_create_args smc_args = SMC_DEVICE_CREATE_ARGS_DEFAULT;
struct smc_device* smc = NULL;

/* Intégrateur */
struct smc_integrator integrator = SMC_INTEGRATOR_NULL;
struct smc_doubleN_context ctx = SMC_DOUBLEN_CONTEXT_NULL;

/* Résultat de l'estimatation */
```

```
struct smc_estimator* estimator = NULL;
struct smc_estimator_status status = SMC_ESTIMATOR_STATUS_NULL;
```

19a ⟨*Libérer les variables locales au calcul de sensibilité* 19a⟩≡

```
if(estimator) smc_estimator_ref_put(estimator);
if(smc) smc_device_ref_put(smc);
```

Le lecteur attentif aura remarqué que l'intégrateur utilise la fonction `run_realisation` et non directement la ⟨*Fonction de réalisation* 5⟩ développée dans ce document (voir ⟨*Exécuter l'intégration Monte Carlo* 17b⟩). `run_realisation` est une fonction intermédiaire qui ne fait qu'appeler la fonction de réalisation. `run_realisation` est donc parfaitement dispensable sauf à la bibliothèque `Star-MonteCarlo` qui nous impose la signature de la fonction à utiliser, c'est à dire le type des paramètres d'entrées et de sorties. En d'autres termes, l'utilisation de cette fonction intermédiaire nous permet de faciliter l'écriture et la lecture de la fonction de réalisation en la libérant de contraintes fonctionnelles imposées par `Star-MonteCarlo`.

19b ⟨*Fonctions utilitaires* 15⟩+≡

```
static res_T
realisation
  (struct ssp_rng* rng,
   const struct sgs_scene* scene,
   double* weight);

static res_T
run_realisation
  (void* output,
   struct ssp_rng* rng,
   const unsigned ithread,
   void* ctx)
{
  struct smc_doubleN_context* context = NULL;
  struct sgs* sgs = NULL;
  ASSERT(ctx && output);
  (void)ithread; /* Éviter l'avertissement "variable inutilisée" */
  context = ctx;
  sgs = context->integrand_data;
  return realisation(rng, &sgs->scene, SMC_DOUBLEN(output));
}
```

E Structure de mise œuvre

Cette partie décrit la structure du fichier C dans lequel l'algorithme de calcul de sensibilité est mis en oeuvre :

19c ⟨*sgs_compute_sensitivity_translation.c* 19c⟩≡

⟨*Liste des inclusions* 22c⟩

⟨*Constantes* 22b⟩

⟨*Fonctions utilitaires* 15⟩
⟨*Fonction de réalisation* 5⟩
⟨*Calculer la sensibilité à la translation* 17a⟩

En plus des fonctions de calcul écrites dans les sections précédents, notre fichier contient des fonctions utilitaires notamment utilisées pour interroger les propriétés physiques du système (figure 1) telles que ρ, la réflectivité de la paroi du haut (equation 2), ou S_b, l'émission thermique de la paroi de droite (équation 4). Le calcul de leur gradient surfacique, utilisé pour évaluer leur dérivée surfacique (section 3.2), est également ajouté comme fonction utilitaire :

20 ⟨*Fonctions utilitaires* 15⟩+≡

```
static double
get_rho
  (const struct sgs_scene* scene,
   const double pos[2])
{
  ASSERT(scene && pos);

  return 0.25
    * (1 - cos(2*PI*pos[X]/scene->D[X]))
    * (1 - cos(2*PI*pos[Y]/scene->D[Y]));
}

static void
get_grad_rho
  (const struct sgs_scene* scene,
   const double pos[2],
   double grad[2])
{
  ASSERT(scene && pos && grad);

  grad[X] = 0.25
    * (((2*PI)/scene->D[X])*sin(2*PI*pos[X]/scene->D[X]))
    * (1 - cos(2*PI*pos[Y]/scene->D[Y]));
  grad[Y] = 0.25
    * (((2*PI)/scene->D[Y])*sin(2*PI*pos[Y]/scene->D[Y]))
    * (1 - cos(2*PI*pos[X]/scene->D[X]));
}

static double
get_Sb
  (const struct sgs_scene* scene,
   const double pos[2])
{
  ASSERT(scene && pos);
  return
    (1 - cos(2*PI*pos[X]/scene->D[Y]))
```

```
    * (1 - cos(2*PI*pos[Y]/scene->D[Z]));
}

static void
get_grad_Sb
  (const struct sgs_scene* scene,
   const double pos[2],
   double grad[2])
{
  ASSERT(scene && pos && grad);

  grad[X] =
      (((2*PI)/scene->D[Y])*sin(2*PI*pos[X]/scene->D[Y]))
    * (1 - cos(2*PI*pos[Y]/scene->D[Z]));
  grad[Y] =
      (((2*PI)/scene->D[Z])*sin(2*PI*pos[Y]/scene->D[Z]))
    * (1 - cos(2*PI*pos[X]/scene->D[Y]));
}
```

Autres fonctions utilitaires, les fonctions utilisées lors du suivi des chemins pour déterminer si un rayon a intersecté le récepteur (fonction `hit_receiver`) ou la source (fonction `hit_source`). Pour la source il suffit de s'assurer que le rayon intersecte le côté droit de la boîte, ici identifié par la constante `SGS_SURFACE_X_MAX`.

21a ⟨*Fonctions utilitaires* 15⟩+≡

```
static int
hit_source
  (const struct sgs_scene* scene,
   const double ray_org[3],
   const double ray_dir[3],
   const struct sgs_hit* hit)
{
  ASSERT(scene && ray_org && ray_dir && hit);
  /* Éviter l'avertissement de compilation "variable inutilisée" */
  (void)scene, (void)ray_org, (void)ray_dir;

  if(SGS_HIT_NONE(hit) || hit->surface != SGS_SURFACE_X_MAX) {
    return 0;
  } else {
    return 1; /* L'intersection a lieu sur la source */
  }
}
```

Pour le récepteur on teste d'abord si l'intersection a lieu sur la paroi du parallélépipède sur laquelle celui-ci est positionné, une paroi identifiée dans le code par la constante `SGS_SURFACE_Z_MIN`. Reste alors à déterminer si l'intersection se situe sur le récepteur lui même. Pour cela il nous suffit de tester si la position d'intersection appartient au sous domaine de la surface sur lequel le récepteur est défini.

21b ⟨*Fonctions utilitaires* 15⟩+≡

```
static int
hit_receiver
  (const struct sgs_scene* scene,
   const double ray_org[3],
   const double ray_dir[3],
   const struct sgs_hit* hit)
{
  double hit_pos[3];
  ASSERT(scene && ray_org && ray_dir && hit);

  /* Le rayon n'intersecte pas la surface où le récepteur se trouve */
  if(SGS_HIT_NONE(hit) || hit->surface != SGS_SURFACE_Z_MIN) {
    return 0;
  }

  /* Calculer la position de l'intersection */
  hit_pos[X] = ray_org[X] + hit->distance*ray_dir[X];
  hit_pos[Y] = ray_org[Y] + hit->distance*ray_dir[Y];
  hit_pos[Z] = ray_org[Z] + hit->distance*ray_dir[Z];

  /* Le rayon n'intersecte pas le récepteur*/
  if(hit_pos[X] < scene->recv_min[X] || hit_pos[X] > scene->recv_max[X]
  || hit_pos[Y] < scene->recv_min[Y] || hit_pos[Y] > scene->recv_max[Y]) {
    return 0;

  /* Le rayon intersecte le récepteur*/
  } else {
    return 1;
  }
}
```

Dernière fonction utilitaire à écrire, la fonction qui calcule la surface A_h de la paroi spéculaire :

22a ⟨*Fonctions utilitaires* 15⟩+≡

```
static double
get_Sr(const struct sgs_scene* scene)
{
  ASSERT(scene);
  return scene->D[X] * scene->D[Y];
}
```

On complète notre fichier en définissant les constantes `X`, `Y` et `Z` utilisées tout du long pour simplifier la lecture du code lors des accès aux éléments d'un vecteur.

22b ⟨*Constantes* 22b⟩≡

```
enum {X, Y, Z};
```

Enfin, on liste en début des sources les fichiers d'en-tête utilisés par notre code.

22c ⟨*Liste des inclusions* 22c⟩≡

```
#include "sgs_c.h"
#include "sgs_geometry.h"
#include "sgs_log.h"

#include <star/smc.h> /* Calcul Monte Carlo */
#include <star/ssp.h> /* Générateur de nombre aléatoire et distributions */

/* Manipuler des vecteurs de double à 2 et 3 dimensions */
#include <rsys/double2.h>
#include <rsys/double3.h>
```

F Script d'exécution du calcul et résultats

Nous écrivons ici les scripts *shell* qui calculent la sensibilité du flux à $\ddot{\pi}$ d'abord par Monte Carlo puis par différences finies en vue d'une validation croisée des résulats. Le premier script ⟨*mc.sh* 23a⟩ lance plusieurs fois le programme dont le présent document décrit la fonction de réalisation ; l'objet étant de calculer à différentes valeurs de $\ddot{\pi}$ la sensibilité du flux φ reçu par le capteur (équation 1) ainsi que la fraction de ce même flux qui dépend de $\ddot{\pi}$ (annexe C).

23a ⟨*mc.sh* 23a⟩≡

```
#!/bin/sh -e

# Estimation par Monte Carlo de la sensibilité et de la fraction du
# flux qui dépend de pi

⟨Configuration géométrique 24d⟩
⟨Paramètres des calculs Monte Carlo 24e⟩

⟨Pour différentes valeurs de π̈ 24b⟩
do
  ⟨Lancer un calcul Monte Carlo 23b⟩
  ⟨Post traiter le résultat 24a⟩
  ⟨Passer à la valeur de π̈ suivante 24c⟩
done
```

Chaque calcul Monte Carlo se résume à exécuter ledit programme nommé `sgs` (acronyme de *Star Geometric Sensitivity*) pour une valeur de $\ddot{\pi}$ considérée. On notera que la configuration géométrique décrite en figure 1 est indépendante de ses dimensions réelles. Dans notre script ces dimensions sont fixées via deux variables qui définissent le coin inférieur (`lower`) et le coin supérieur (`upper`) d'un parallélépipède aligné aux axes, deux variables passées en arguments du programme via l'option `-b`.

23b ⟨*Lancer un calcul Monte Carlo* 23b⟩≡

```
out=$(./sgs \
  -n "${nrealisations}" \
  -b low="${lower}":upp="${upper}":pi="${pi}")
```

avec `nrealisations` le nombre de réalisations utilisées par le calcul Monte Carlo. Si ce document décrit en détail les sources C de sa ⟨*Fonction de réalisation* 5⟩, nous renvoyons le lecteur vers les autres fichiers C et l'aide du programme `sgs` (affichée via l'option `-h`) pour plus d'informations quant aux fonctionnement et options du programme.

Une fois le calcul terminé, en post-traiter le résultat se résume à afficher sur la sortie standard la valeur de $\ddot{\pi}$ courante suivie de l'estimation et de l'écart type de la sensibilité et du flux que nous venons d'estimer. Ci-après nous utilisons la commande `sed` pour extraire et afficher ces valeurs stockées dans la variable `out` à l'issu de notre calcul Monte Carlo.

24a ⟨*Post traiter le résultat* 24a⟩≡
```
prompt="[^~]\{1,\}~ "
estimation="[^[:blank:]]\{1,\}"
error="[^\n$]\{1,\}"
line0="${prompt}\(${estimation}\) +/- \(${error}\)\n" # Sensibilité
line1="${prompt}\(${estimation}\) +/- \(${error}\)$" # Flux
printf "%s " "${pi}"
echo "${out}" | sed -n "1{N;s#^${line0}${line1}#\1 \2 \3 \4#p}"
```

Ces deux étapes, à savoir le calcul Monte Carlo et son post-traitement, sont lancées `nsteps` fois pour différentes valeurs de $\ddot{\pi}$, la valeur de $\ddot{\pi}$ à chaque itération étant simplement sa valeur précédente incrémentée d'un pas constant égal à `pi_step`.

24b ⟨*Pour différentes valeurs de* $\ddot{\pi}$ 24b⟩≡
```
i=0
pi=0
while [ "${i}" -lt "${nsteps}" ]
```

24c ⟨*Passer à la valeur de* $\ddot{\pi}$ *suivante* 24c⟩≡
```
i=$((i + 1))
pi=$(printf "%s + %s\n" "${pi}" "${pi_step}" | bc)
```

Pour compléter le script, ne reste plus qu'à définir les variables qui caractérisent notre configuration géométrique ainsi que les paramètres qui pilotent nos différents calculs, tels que le nombre de calculs à lancer ou encore le nombre de réalisations par calcul.

24d ⟨*Configuration géométrique* 24d⟩≡
```
h=1 # Hauteur du parallélépipède
lower="0,0,0"
upper="1,1,${h}"
```

24e ⟨*Paramètres des calculs Monte Carlo* 24e⟩≡
```
nrealisations=100000000
nsteps=14
pi_step=0.1
```

Dans un nouveau script `fd.sh` nous pouvons alors calculer par différences finies la sensibilité du flux à $\ddot{\pi}$ à partir des estimations Monte Carlo en sortie de ⟨*mc.sh* 23a⟩, des résultats lus via l'entrée standard. Pour pouvoir comparer les résultats, ce second script recopie en sortie les sensibilités estimées par Monte Carlo (`sen_mc`) en plus d'écrire ces mêmes sensibilités calculées cette fois par différences finies (`sen_fd`). Leur écarts types respectif (`err_mc` et `err_fd`) sont également des données de sortie. On notera enfin que l'ensemble des résultats sont adimentionnalisés et sont par conséquent donnés pour une valeur de $\ddot{\pi}$ indépendante de la hauteur du parallélépipède (`pi_over_h`).

25a ⟨*fd.sh* 25a⟩≡

```
#!/bin/sh -e

# Calcule la sensibilité par différences finies à partir des
# estimations par Monte Carlo de la fraction du flux qui dépend de pi

# Liste des données pour chaque ligne écrite en sortie
printf "pi_over_h sen_mc err_mc sen_fd err_fd\n"

⟨Fonctions utilitaires à fd.sh 27⟩

⟨Lire les paramètres d'entrée 25c⟩

⟨Pour chaque valeur de π̈ considérée 26b⟩
do
  ⟨Lire la valeur du flux autour de π̈ 26d⟩
  ⟨Calculer par différences finies la sensibilité du flux à π̈ 26e⟩
  ⟨Lire l'estimation Monte Carlo de la sensibilité du flux à π̈ 26f⟩
  ⟨Écrire les résultats adimensionnaliser 26g⟩
  ⟨Passer au calcul suivant 26c⟩
done
```

Pour les calculs en différences finies nous avons besoin de connaître les paramètres utilisés par les estimations Monte Carlo, comme le δ entre chaque valeur de $\ddot{\pi}$ (`pi_step`) ou encore la hauteur du parallélépipède nécessaire pour adimensionnaliser les résultats (`h`). Pour passer ces données d'un script à l'autre, on ajoute aux sorties de `mc.sh` un en-tête contenant ces paramètres, en-tête qui peut alors être lu par le script `fd.sh`.

25b ⟨*Paramètres des calculs Monte Carlo* 24e⟩+≡

```
printf "%s %s\n" "${h}" "${pi_step}"
```

25c ⟨*Lire les paramètres d'entrée* 25c⟩≡

```
read -r header
h=$(echo "${header}" | cut -d' ' -f1)
pi_step=$(echo "${header}" | cut -d' ' -f2)
```

Autre donnée nécessaire par l'adimensionnement, la valeur maximale du flux. Celle-ci correspond à la valeur de φ lorsque la boîte est fermée, c'est à dire lorsque $\ddot{\pi}$ vaut 0, soit la valeur de $\ddot{\pi}$ du premier calcul Monte Carlo lancé par

`mc.sh`. Après avoir lu l'en-tête de ses sorties nous lisons donc ce premier résultat que l'on stocke dans la variable `p` avant d'en extraire la valeur φ (`phi_max`).

26a ⟨*Lire les paramètres d'entrée* 25c⟩+≡
```
read -r p
phi_max=$(echo "${p}" | cut -d' ' -f4 | float_to_bc)
```

La boucle principale du script consiste à lire l'entrée standard jusqu'à ce qu'il n'y ait plus de résultat Monte Carlo à traiter. À chaque itération, le résultat Monte Carlo pour la valeur de $\ddot{\pi}$ courante est stocké dans la variable `c` tandis que le résultat précédent et suivant sont respectivement stockés dans les variables `p` et `n`.

26b ⟨*Pour chaque valeur de* $\ddot{\pi}$ *considérée* 26b⟩≡
```
read -r c
while read -r n
```

26c ⟨*Passer au calcul suivant* 26c⟩≡
```
p="${c}"
c="${n}"
```

Il suffit alors d'extraire le flux (`phi`) et son erreur (`err`) aux valeurs de $\ddot{\pi}$ précédente (`p`) et suivante (`n`) pour calculer par différences finies la sensibilité et l'erreur associée pour la valeur de $\ddot{\pi}$ courante :

26d ⟨*Lire la valeur du flux autour de* $\ddot{\pi}$ 26d⟩≡
```
phi_p=$(echo "${p}" | cut -d' ' -f4 | float_to_bc)
err_p=$(echo "${p}" | cut -d' ' -f5 | float_to_bc)
phi_n=$(echo "${n}" | cut -d' ' -f4 | float_to_bc)
err_n=$(echo "${n}" | cut -d' ' -f5 | float_to_bc)
```

26e ⟨*Calculer par différences finies la sensibilité du flux à* $\ddot{\pi}$ 26e⟩≡
```
sen_fd=$(echo "(${phi_n}-${phi_p})/(2*${pi_step})" | bc_cmd)
err_fd=$(echo "(${err_n}+${err_p})/(2*${pi_step})" | bc_cmd)
```

On utilise alors le résultat Monte Carlo au $\ddot{\pi}$ considéré pour retrouver non seulement la valeur de $\ddot{\pi}$ mais aussi pour extraire l'estimation Monte Carlo de la sensibilité de φ à $\ddot{\pi}$:

26f ⟨*Lire l'estimation Monte Carlo de la sensibilité du flux à* $\ddot{\pi}$ 26f⟩≡
```
pi=$(echo "${c}" | cut -d' ' -f1 | float_to_bc)
sen_mc=$(echo "${c}" | cut -d' ' -f2 | float_to_bc)
err_mc=$(echo "${c}" | cut -d' ' -f3 | float_to_bc)
```

Ne reste plus qu'à écrire l'ensemble des résultats attendus :

26g ⟨*Écrire les résultats adimensionnaliser* 26g⟩≡
```
pi_over_h=$(echo "${pi}/${h}" | bc_cmd)
sen_mc=$(echo "${sen_mc}/${phi_max}*${h}" | bc_cmd)
err_mc=$(echo "${err_mc}/${phi_max}*${h}" | bc_cmd)
sen_fd=$(echo "${sen_fd}/${phi_max}*${h}" | bc_cmd)
err_fd=$(echo "${err_fd}/${phi_max}*${h}" | bc_cmd)
printf "%s %s %s %s %s\n" \
  "${pi_over_h}" "${sen_mc}" "${err_mc}" "${sen_fd}" "${err_fd}"
```

On complète finalement notre script par les fonctions utilitaires utilisées tout du long, à savoir la fonction `float_to_bc` qui convertie un nombre à virgule flottante dans le format attendu par le programme `bc`, et la fonction `bc_cmd` qui exécute le programme `bc` et en post-traite le résultat pour supprimer les zéros qui suivent le dernier chiffre significatif.

27 ⟨*Fonctions utilitaires à* `fd.sh` 27⟩≡

```
float_to_bc()
{
  sed 's/\([+-]\{0,1\}[0-9]\{0,\}\.\{0,1\}[0-9]\{1,\}\)'\
'[eE]+\{0,1\}\(-\{0,1\}\)\([0-9]\{1,\}\)/(\1*10^\2\3)/g'
}

bc_cmd()
{
  bc -l | sed '/\./s/\.\{0,\}0\{1,\}$//'
}
```

Références

[Kernighan, 1988] Kernighan, B. W. (1988). *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition.

[Lapeyre et al., 2022] Lapeyre, P., Blanco, S., Caliot, C., Coustet, C., d'Eon, E., Fournier, R., He, Z., and Mourtaday, N. C. (2022). A physical model and a monte carlo estimate for the spatial derivative of the specific intensity. *arXiv preprint arXiv :2206.05167*.

[Matsumoto and Nishimura, 1998] Matsumoto, M. and Nishimura, T. (1998). Mersenne twister : a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1) :3–30.