— Informatique (ITC) —

Devoir libre n°1 (DL1)

À travailler et à rendre avant le 17 janvier 2025

Penser à lire le sujet intégralement avant de composer

Lancer de rayons

Et qu'au contraire les chats voient de nuit par le moyen des rayons qui tendent de leurs yeux vers les objets.

— René Descartes, *Discours de la méthode - La dioptrique(1637)*

Le sujet présente une méthode de génération d'images en deux dimensions représentant des objets dans l'espace, éclairés par des sources lumineuses. Cette technique appelée *lancer de rayons* s'appuie sur les lois de propagation de la lumière, ce qui lui confère une grande qualité de rendu, au prix d'un temps de calcul souvent élevé. L'augmentation de la puissance des processeurs devrait bientôt pouvoir rendre cette technique compatible avec les jeux vidéo.

Figure 1 Boite en bois et boules[1]

Ce sujet propose d'illustrer les principales caractéristiques de la méthode, en se limitant à des scènes ne comportant que des sphères et des sources lumineuses ponctuelles.

De nombreuses questions sont indépendantes, il est toutefois demandé de les traiter dans l'ordre.

Les seuls langages informatiques autorisés dans cette épreuve sont Python et SQL. Pour répondre à une question, il est possible, et souvent souhaitable, de faire appel aux fonctions définies dans les questions précédentes.

Les modules `math` et `numpy` ont été rendus accessibles grâce à l'instruction

```
import math, numpy as np
```

Dans tout le sujet, le terme « liste » désigne une valeur de type `list`. Le terme « tableau » désigne une valeur de type `np.ndarray`. Enfin le terme « séquence » désigne une suite finie *indiçable* et *itérable*.

— *Indiçable* signifie que les éléments sont accessibles par des indices, `seq[0]` désignant le premier élément de la séquence `seq`.

[1] Exemple fourni avec le logiciel libre de lancer de rayons POV-Ray. Fichier woodbox.pov, POV-Ray scene file by Dan Farmer, sous licence Creative Commons Attribution-ShareAlike 3.0

— *Itérable* signifie que la séquence peut être parcourue dans une boucle par `for element in seq: ...`

Un tuple d'entiers, une liste d'entiers et un tableau d'entiers sont trois exemples de « séquence d'entiers ».

Les entêtes des fonctions demandées sont annotés pour préciser les types des paramètres et du résultat. Ainsi,

```
def uneFonction(n:int, X:[float], c:str, u) -> (np.ndarray, int):
```

signifie que la fonction `uneFonction` prend quatre paramètres `n`, `X`, `c` et `u`, où `n` est un entier, `X` une liste de nombres à virgule flottante, `c` une chaine de caractères et le type de `u` n'est pas précisé. Cette fonction renvoie un couple constituée d'un tableau et d'un entier.

Il n'est pas demandé aux candidats d'annoter leurs fonctions, la rédaction pourra commencer par

```
def uneFonction(n, X, c, u):
    ...
```

De façon générale, une attention particulière sera portée à la lisibilité, la simplicité et la clarté du code proposé. L'utilisation d'identifiants significatifs, l'emploi judicieux de commentaires seront appréciés.

Une liste de fonctions potentiellement utiles est fournie à la fin du sujet.

I Géométrie À traiter absolument

Dans tout le sujet, l'espace est muni d'un repère $(O, \vec{u}_x, \vec{u}_y, \vec{u}_z)$ orthonormé direct. $\|\vec{v}\|$ désigne la norme euclidienne du vecteur $\vec{v}$. Le produit scalaire de deux vecteurs $\vec{v}_1$ et $\vec{v}_2$ est noté $\vec{v}_1 \cdot \vec{v}_2$, leur produit terme à terme (produit de Hadamard) est noté $\vec{v}_1 \odot \vec{v}_2$: $\vec{v}_1 \odot \vec{v}_2 = v_{1x}v_{2x}\vec{u}_x + v_{1y}v_{2y}\vec{u}_y + v_{1z}v_{2z}\vec{u}_z$.

Tout point ou vecteur de l'espace est représenté en Python par le tableau de ses trois coordonnées cartésiennes, de type `float`. Pour faciliter la compréhension, un tel tableau est considéré de type `point` quand il désigne un point et de type `vecteur` quand il désigne un vecteur : `np.array([0.,0.,0.])` est considéré de type `point` quand il représente le point O et de type `vecteur` quand il représente le vecteur nul.

Beaucoup d'opérations classiques sur les vecteurs se transposent simplement dans la syntaxe de `numpy`. Si $\vec{v}_1$, $\vec{v}_2$ sont 2 vecteurs et t un réel, représentés respectivement par `v1`, `v2` et `t`, alors $\vec{v}_1 + \vec{v}_2$, $\vec{v}_1 - \vec{v}_2$, $\vec{v}_1 \odot \vec{v}_2$ et $t\vec{v}_1$ peuvent être calculés simplement par `v1 + v2`, `v1 - v2`, `v1 * v2` et `t * v1`.

Quand il n'y a pas de confusion possible, un objet mathématique est assimilé à sa représentation en Python. Ainsi, par exemple, « la fonction `f` prend en paramètre le vecteur $\vec{v}_1$ » signifie que la fonction `f` accepte des valeurs de type `vecteur` représentant le vecteur $\vec{v}_1$.

On rappelle que la valeur du cosinus de l'angle formé par deux vecteurs unitaires correspond à la valeur de leur produit scalaire. Ainsi, si $\vec{u}_1$ et $\vec{u}_2$ sont deux vecteurs unitaires, $\cos(\widehat{\vec{u}_1, \vec{u}_2}) = \vec{u}_1 \cdot \vec{u}_2$. Par ailleurs, si $\vec{u}$ est un vecteur unitaire et $\vec{v}$ un vecteur quelconque, le projeté du vecteur $\vec{v}$ dans la direction $\vec{u}$ est donné par $(\vec{u} \cdot \vec{v})\,\vec{u}$.

Q 1. Écrire une fonction d'entête

```
def vec(A:point, B:point) -> vecteur:
```

qui prend deux points en paramètre et renvoie le vecteur $\overrightarrow{AB}$.

Q 2. Écrire une fonction d'entête

```
def ps(v1:vecteur, v2:vecteur) -> float:
```

qui prend en paramètres deux vecteurs $\vec{v}_1$ et $\vec{v}_2$ et qui renvoie la valeur de leur produit scalaire $\vec{v}_1 \cdot \vec{v}_2$.

Q 3. Écrire une fonction d'entête

```
def norme(v:vecteur) -> float:
```

qui prend en paramètre un vecteur $\vec{v}$ et qui renvoie $\|\vec{v}\|$, la valeur de sa norme euclidienne.

Q 4. Écrire une fonction d'entête

```
def unitaire(v:vecteur) -> vecteur:
```

qui prend en paramètre un vecteur $\vec{v}$ non nul et qui renvoie $\dfrac{1}{\|\vec{v}\|}\vec{v}$, le vecteur unitaire correspondant.

On utilise dans ce sujet le modèle du rayon lumineux de l'optique géométrique. Un rayon lumineux issu du point S est une demi-droite d'origine S. Ce rayon est représenté en Python par le couple $(S, \vec{u})$, où S est le point de départ du rayon et $\vec{u}$ le vecteur unitaire donnant la direction de propagation du rayon lumineux. Tout point M du rayon est tel que $\overrightarrow{SM} = t\vec{u}$, avec $t \in \mathbb{R}^+$. Un tel couple est désignée dans la suite par le type `rayon`.

Ainsi, en définissant `O = np.array([0., 0., 0.]` et `u = np.array([0, 0.6, 0.8]`, le couple `(O, u)` représente le rayon issu de O dans la direction $3\vec{u}_y + 4\vec{u}_z$.

Q 5. Que font les fonctions `pt`, `dir` et `ra` ci-dessous ?

```
1  def pt(r:rayon, t:float) -> point:
2      assert t >= 0
3      (S, u) = r
4      return S + t * u
```

```
5   def dir(A:point, B:point) -> vecteur:
6       return unitaire(vec(A, B))

7   def ra(A:point, B:point) -> rayon:
8       return A, dir(A, B)
```

Comme toutes les fonctions définies dans ce sujet, les fonctions `pt`, `dir` et `ra` peuvent être utilisées dans la suite.

Une sphère de centre C et de rayon $r > 0$ est l'ensemble des points de l'espace situés à la distance r du point C. Ici, elle est représentée en Python par le couple (C, r). Un tel couple est désigné par le type `sphère`.

Q 6. Écrire une fonction d'entête

```
def sp(A:point, B:point) -> sphère:
```

qui renvoie la sphère de centre A passant par B.

Q 7. Montrer qu'une droite passant par le point A de vecteur directeur $\vec{u}$ et une sphère (C, r) sont sécantes si et seulement si l'équation d'inconnue t

$$t^2 + 2t(\vec{u} \cdot \overrightarrow{CA}) + \|\overrightarrow{CA}\|^2 - r^2 = 0 \tag{1}$$

possède deux solutions réelles, éventuellement confondues.

Q 8. Écrire une fonction d'entête

```
def intersection(r:rayon, s:sphère) -> (point, float) or None:
```

qui renvoie le premier point de la sphère `s` frappé par le rayon lumineux `r` et la distance entre ce point et l'origine du rayon. La fonction renvoie `None` si le rayon ne coupe pas la sphère. On suppose que l'origine du rayon n'est pas située à l'intérieur de la sphère.

II Optique À traiter absolument

Les couleurs seront représentées par des tableaux de 3 nombres flottants, associés dans la suite au type `couleur`. Les trois composantes codent respectivement le niveau de rouge, vert et bleu de la couleur considérée (composantes RVB). Chaque composante est comprise entre 0 et 1. Plus la valeur est élevée, plus la composante contribue fortement à la couleur. Ainsi, (1, 1, 1) correspond au blanc, (0, 0, 0) au noir, (0.5, 0.5, 0.5) désigne un gris moyen et (0.6, 1, 0.6) un vert pastel.

Pour toute la suite, on a défini les variables globales

```
noir  = np.array([0., 0., 0.])
blanc = np.array([1., 1., 1.])
```

II.A – Visibilité

Q 9. Une source lumineuse ponctuelle S ne peut être vue d'un point P d'une sphère (C, r) que si la source est au-dessus de l'horizon de P, défini ici comme le plan tangent à la sphère en P. Donner une condition géométrique pour que la source soit au-dessus de l'horizon.

Q 10. Écrire une fonction booléenne, d'entête

```
def au_dessus(s:sphère, P:point, src:point) -> bool:
```

qui détermine si la source située en `src` est au-dessus de l'horizon du point `P` de la sphère `s`.

Q 11. On considère une scène contenant plusieurs sphères et une source lumineuse. Pour que la source soit visible d'un point P d'une sphère particulière, il faut que cette source soit au-dessus de l'horizon et qu'aucune autre sphère ne la cache. Écrire une fonction booléenne d'entête

```
def visible(obj:[sphère], j:int, P:point, src:point) -> bool:
```

où le paramètre `obj` est une liste contenant les sphères de la scène et `src` l'emplacement de la source lumineuse. Cette fonction détermine si la source est visible du point `P`, appartenant à la sphère `obj[j]`.

II.B – Diffusion

On considère un point P, à la surface d'une sphère, éclairé sous l'incidence θ par une source lumineuse S ponctuelle de couleur $C_s = (R_s, V_s, B_s)$. On note $\vec{N}$ le vecteur unitaire normal à la sphère en P, dirigé vers l'extérieur de la sphère, et $\vec{u}$ le vecteur unitaire du rayon lumineux qui éclaire P en provenance de la source. On considère que le point P diffuse la lumière de la source de manière isotrope dans tout le demi-espace délimité par le plan tangent à la sphère en P qui contient la source (figure 2). Autrement dit, le point P diffuse la lumière de la source dans toutes les directions $\vec{w}$ telles que $\vec{w} \cdot \vec{N} \geqslant 0$ (les vecteurs $\vec{u}$, $\vec{w}$ et $\vec{N}$ ne sont pas forcément coplanaires) et la lumière diffusée ne dépend pas de la direction d'observation $\vec{w}$, en particulier elle ne dépend pas de θ'.

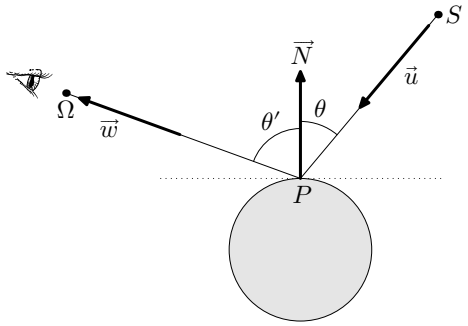

Figure 2 Parcours de la lumière de la source à l'œil

La faculté d'un objet à diffuser la lumière est modélisée par ses coefficients de diffusion k_{dr}, k_{dv} et k_{db} qui mesurent son aptitude à réémettre les composantes rouge, verte et bleue. Chaque coefficient est un nombre à virgule flottante compris entre 0 et 1. On représente les coefficients de diffusion d'un objet par un triplet $k_d = (k_{dr}, k_{dv}, k_{db})$ de type `couleur`. La couleur $C_d = (R_d, V_d, B_d)$ de la lumière diffusée par le point P éclairé par la source S suit alors la loi de Lambert :

$$C_d = (k_d \odot C_s)\cos\theta \quad \text{soit} \quad (R_d, V_d, B_d) = (k_{dr}R_s\cos\theta,\ k_{dv}V_s\cos\theta,\ k_{db}B_s\cos\theta). \tag{2}$$

Q 12. Écrire une fonction d'entête

```
def couleur_diffusée(r:rayon, Cs:couleur, N:vecteur, kd:couleur) -> couleur:
```

qui renvoie la couleur de la lumière diffusée par le point P éclairé par un rayon lumineux `r` en provenance d'une source ponctuelle de couleur `Cs`. Les paramètres `N` et `kd` représentent respectivement le vecteur unitaire normal à l'objet en P et les coefficients de diffusion de l'objet (figure 2). La source S est supposée visible de P.

II.C – Réflexion

Si la surface de l'objet est réfléchissante, au phénomène de diffusion s'ajoute le phénomène de réflexion. Lorsqu'un rayon lumineux $(S, \vec{u})$ issu d'un point source S atteint un point P de la surface, il donne naissance au rayon réfléchi $(P, \overrightarrow{w})$. Les lois de la réflexion de Descartes stipulent que, avec les notations de la figure 2,

— $\vec{u}$, $\overrightarrow{w}$ et $\overrightarrow{N}$ sont coplanaires ;

— $(\vec{u} + \overrightarrow{w}) \cdot \overrightarrow{N} = 0$, ce qui correspond à $\theta' = \theta$.

Q 13. Écrire une fonction d'entête

```
def rayon_réfléchi(s:sphère, P:point, src:point ) -> rayon:
```

qui renvoie le rayon réfléchi par le point `P` de la sphère `s` en provenance de la source S placée en `src`. Le résultat est le couple $(P, \overrightarrow{w})$ représentant le rayon émergent. La source S est supposée visible de P.

III Enregistrement des scènes

Les différentes scènes à représenter sont enregistrées dans une base de données relationnelle. La figure 3 donne sa structure physique.

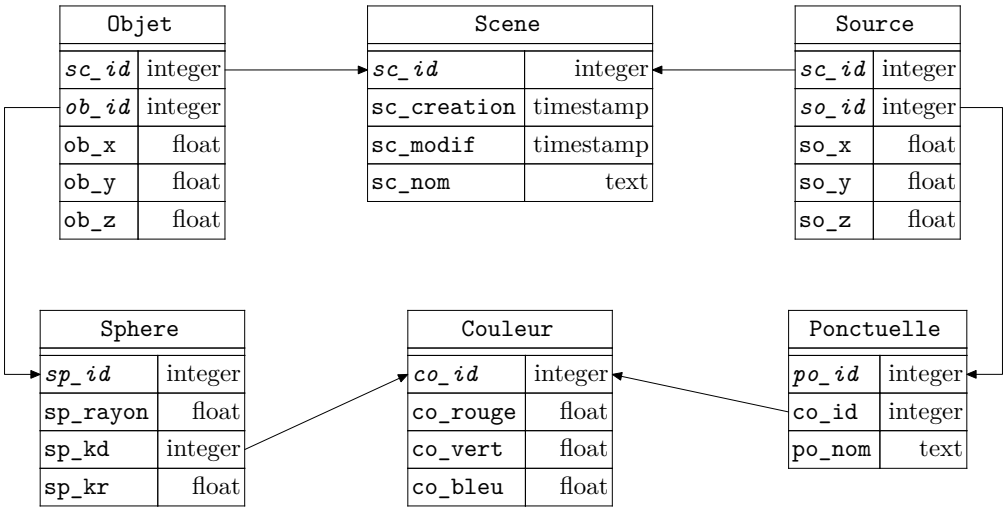

Figure 3 Structure physique de la base de données des scènes

Cette base comporte les six tables listées ci-dessous avec la description de leurs colonnes :

— la table `Scene` répertorie les scènes
  - `sc_id` identifiant (entier arbitraire) de la scène (clé primaire)
  - `sc_creation` date de création de la scène
  - `sc_modif` date de dernière modification de la scène
  - `sc_nom` nom de la scène

— la table `Couleur` définit les couleurs utilisées
  - `co_id` identifiant (entier arbitraire) de la couleur (clé primaire)
  - `co_rouge` valeur de la composante rouge (dans l'intervalle $[0, 1]$)
  - `co_vert` valeur de la composante verte (dans l'intervalle $[0, 1]$)
  - `co_bleu` valeur de la composante bleue (dans l'intervalle $[0, 1]$)

— la table `Sphere` liste les sphères utilisées pour construire les scènes
  - `sp_id` identifiant (entier arbitraire) de la sphère (clé primaire)
  - `sp_rayon` rayon de la sphère
  - `sp_kd` coefficients de diffusion de la sphère (référence dans la table `Couleur`)
  - `sp_kr` coefficient de réflexion de la sphère (cf. partie V)

— la table `Ponctuelle` répertorie les sources ponctuelles disponibles
  - `po_id` identifiant (entier arbitraire) de la source ponctuelle (clé primaire)
  - `co_id` couleur de la source (référence dans la table `Couleur`)
  - `po_nom` nom de la source

— la table `Objet` fait le lien entre les scènes et les objets qu'elles contiennent, ses deux premières colonnes constituent sa clé primaire
  - `sc_id` identifiant de la scène
  - `ob_id` identifiant de l'objet
  - `ob_x`, `ob_y`, `ob_z` coordonnées du centre de l'objet dans la scène considérée

— la table `Source` indique quelles sont les sources qui éclairent chaque scène, ses deux premières colonnes constituent sa clé primaire
  - `sc_id` identifiant de la scène
  - `so_id` identifiant de la source
  - `so_x`, `so_y`, `so_z` coordonnées de la source dans la scène considérée

Q 14. Écrire une requête SQL qui donne le nom des scènes créées au cours de l'année 2021.

Q 15. Écrire une requête SQL qui donne, pour chaque scène, son identifiant et le nombre de sources qui l'éclairent.

Q 16. Écrire une requête SQL qui liste l'identifiant, les coordonnées du centre et le rayon de toutes les sphères contenues dans la scène dont le nom est `woodbox`. On suppose qu'une seule scène possède ce nom.

Il est possible en SQL de définir des fonctions utilisateur qui s'utilisent comme les fonctions SQL pré-définies.

On dispose d'une fonction utilisateur booléenne de signature `OCCULTE(sc_id, objr_id, so_id, objo_id)` où `sc_id`, `objr_id`, `so_id` et `objo_id` sont les identifiants respectifs d'une scène, d'un objet de cette scène dit « récepteur », d'une source de cette scène et d'un autre objet de la scène dit « occultant ». La fonction renvoie `TRUE` si l'objet occultant projette son ombre sur l'objet récepteur lorsqu'il est éclairée par la source considérée.

Q 17. Écrire une requête SQL qui, pour la scène `woodbox`, renvoie tous les triplets `objr_id, so_id, objo_id` pour lesquels l'objet d'identifiant `objo_id` occulte la source d'identifiant `so_id` pour l'objet d'identifiant `objr_id`.

IV Lancer de rayons À traiter absolument

Cette partie implante l'algorithme qui génère l'image 2D de la scène 3D à visualiser. Pour représenter une scène en Python, on utilise les quatre variables globales suivantes :

— `Objet` est une liste des n_o objets (sphères de type `sphère`) contenues dans la scène à représenter ;

— `KdObj` est une liste de n_o tableaux de type `couleur` représentant les coefficients de diffusion des sphères, `KdObj[i]` est associé à la sphère `Objet[i]` ;

— `Source` est une liste de n_s points (de type `point`) donnant l'emplacement des n_s sources lumineuses (ponctuelles) qui éclairent la scène à représenter ;

— `ColSrc` est une liste de n_s couleurs, `ColSrc[i]` représente la couleur de la source placée en `Source[i]`.

Pour des raisons d'efficacité, il est d'usage de construire le parcours de la lumière de l'œil vers les sources : on « lance » des rayons. Cela est possible grâce au principe du retour inverse de la lumière : un rayon parcourt le

même chemin pour joindre 2 points A et B, qu'il aille de B vers A ou de A vers B. Dans la suite, les vecteurs unitaires caractérisant les rayons sont donc orientés dans le sens opposé au sens réel de propagation de la lumière.

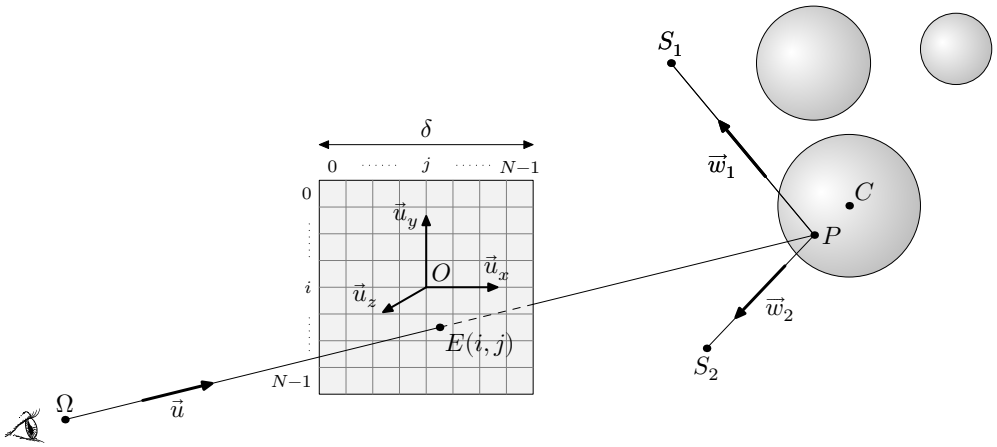

Figure 4 Une scène avec 3 sphères et 2 sources ponctuelles

Un écran translucide est placé dans le plan $(O, \vec{u}_x, \vec{u}_y)$. Le point O coïncide avec le centre de l'écran.
Il sépare les objets de la scène, placés dans le demi espace $z < 0$, de l'œil Ω placé de l'autre côté : $z_\Omega > 0$.

IV.A – Écran

L'écran est un carré de côté δ, divisé en $N \times N$ cases (N pair) qui représentent les pixels de l'image finale. Les variables globales `Delta` et `N` contiennent respectivement les valeurs de δ et N.

Q 18. Écrire une fonction d'entête

```
def grille(i:int, j:int) -> point:
```

qui renvoie les coordonnées cartésiennes du point E, centre de la case repérée par les indices (i, j) de la grille (figure 4).

Q 19. Écrire une fonction d'entête

```
def rayon_écran(omega:point, i:int, j:int) -> rayon:
```

qui renvoie le rayon issu du point `omega` et passant par $E(i, j)$.

IV.B – Couleur d'un pixel

Dans un premier temps, les objets de la scène à représenter sont supposés parfaitement mats, ils se contentent de diffuser la lumière des sources sans la réfléchir.

On considère un point P d'un objet de la scène atteint par un rayon issu de l'œil. On note E le point de l'écran coupé par ce rayon allant de l'œil au point P. La couleur $C_d = (R_d, V_d, B_d)$ diffusée par P est la somme des couleurs diffusées par ce point, dues à l'éclairement des sources visibles du point P.

$$C_d = \sum_{S_i \text{ visibles}} (k_d \odot C_{s_i}) \cos\theta_i. \tag{3}$$

On suppose que les couleurs ne saturent pas : toutes les composantes de C_d restent inférieures à 1. La couleur C_d est affectée au point E de l'écran.

Q 20. Écrire une fonction d'entête

```
def interception(r:rayon) -> (point, int) or None:
```

qui prend en paramètre un rayon `r` et qui renvoie le premier point matériel de la scène atteint par ce rayon ainsi que l'indice de la sphère concernée dans la liste `Objet`. Si le rayon n'intercepte aucune sphère, la fonction renvoie `None`.

Q 21. Écrire une fonction d'entête

```
def couleur_diffusion(P:point, j:int) -> couleur:
```

qui renvoie la couleur diffusée par le point `P` appartenant à la sphère `Objet[j]`. Si aucune source n'éclaire P, la fonction renvoie `noir`.

IV.C – Constitution de l'image

L'image de la scène que l'on souhaite obtenir est construite dans un tableau à 3 dimensions, de taille $N \times N \times 3$, associé au type `image`. L'affectation de la couleur `c` de type `couleur` au pixel (i, j) de l'image `im` s'écrira simplement `im[i, j] = c`.

Q 22. Écrire une fonction d'entête

```
def lancer(omega:point, fond:couleur) -> image:
```

qui génère l'image associée à la scène. Si un rayon n'intercepte aucun objet, le pixel correspondant est de couleur `fond`.

IV.D – Complexités

Si vous le sentez…

Les complexités asymptotiques seront exprimées en fonction du nombre N de lignes de l'image, du nombre n_o d'objets et du nombre n_s de sources.

Q 23. Calculer la complexité temporelle de la fonction `lancer` dans le meilleur des cas en caractérisant la situation correspondante.

Q 24. Calculer la complexité temporelle de la fonction `lancer` dans le pire des cas en caractérisant la situation correspondante.

V Améliorations

Moins impératif, car ça monte en difficulté

V.A – Prise en compte de la réflexion

Désormais les sphères sont supposées au moins partiellement réfléchissantes. Un rayon issu de l'œil peut alors se réfléchir successivement sur plusieurs sphères.

Q 25. Écrire une fonction d'entête

```
def réflexions(r:rayon, rmax:int) -> [(point, int)]:
```

qui renvoie une liste de couples (P_k, i_k) correspondant aux points successivement rencontrés par le rayon `r` au fur et à mesure de ses réflexions. Dans chaque couple, P_k est le point où a lieu la $(k+1)^{\text{ème}}$ réflexion et i_k est l'indice de l'objet contenant P_k. Le résultat comporte au plus `rmax` éléments, les éventuelles réflexions ultérieures ne sont pas prises en compte.

Le pouvoir réfléchissant d'un objet est caractérisé par un coefficient de réflexion k_r propre à chaque objet, $0 \leqslant k_r \leqslant 1$. Les coefficients de réflexion des objets de la scène (de type `float`) sont stockés dans une liste globale `KrObj` à n_o éléments, telle que `KrObj[i]` est le coefficient de réflexion de l'objet `Objet[i]`.

En tenant compte des phénomènes de diffusion et de réflexion, la couleur $C_k = (R_k, V_k, B_k)$ du point P_k vaut

$$C_k = C_{dk} + k_r C_{k+1} \tag{4}$$

où C_{dk} désigne la couleur diffusée par le point P_k (cf. IV.B) et C_k vaut `noir` si k est supérieur au nombre de réflexions considérées.

Q 26. Écrire une fonction d'entête

```
def couleur_perçue(r:rayon, rmax:int, fond:couleur) -> couleur:
```

qui renvoie la couleur du premier point de la scène rencontré par le rayon `r` en tenant compte d'au maximum `rmax` réflexions de ce rayon. Si le rayon ne rencontre aucun objet, la fonction renvoie la couleur `fond`.

Q 27. Écrire une fonction d'entête

```
def lancer_complet(omega:point, fond:couleur, rmax:int) -> image:
```

qui construit l'image de la scène en tenant compte des diffusions et des réflexions.

Q 28. Exprimer la nouvelle complexité dans le pire cas.

V.B – Une optimisation

On construit, à partir de la base de données, les deux listes globales :

— `IdObj` telle que `IdObj[i]` soit l'identifiant dans la base de données (`ob_id`) de la sphère `Objet[i]` ;

— `IdSrc` telle que `IdSrc[i]` soit l'identifiant dans la base de données (`so_id`) de la source `Source[i]`.

Q 29. Écrire la fonction d'entête

```
def table_risque(risque:[[int, int, int]]) -> [[[int]]]:
```

qui prend en paramètre une liste de triplets correspondant au résultat de la requête SQL de la question 17 et construit une liste de listes de listes d'entiers telle que, si `res` est le résultat de la fonction, `res[i][j]` donne la liste (éventuellement vide) des indices des objets susceptibles de masquer la source `Source[j]` pour un point de l'objet `Objet[i]`.

Q 30. Le résultat de la fonction `table_risque` est conservé dans la variable globale `TableRisque`. Écrire la fonction `visible_opt` d'entête

```
def visible_opt(j:int, k:int, P:point) -> bool:
```

qui prend en paramètres l'indice `j` d'une source, l'indice `k` d'une sphère et un point `P` de cette sphère et qui, comme la fonction `visible` de la question 11, détermine si la source `Source[j]` est visible à partir du point `P`.

Opérations et fonctions disponibles en Python et en SQL

Constantes

— `math.inf`, `np.inf` correspondent à $+\infty$, n'importe quel nombre est strictement inférieur à cette valeur.

Fonctions Python diverses

— `range(n)` itérateur sur les `n` premiers entiers ($[\![0, n-1]\!]$).
`list(range(5))` → `[0, 1, 2, 3, 4]`.

— `range(d, f, p)` où `d`, `f` et `p` sont des entiers, itérateur sur les entiers $(r_i = d + ip \mid r_i < f)_{i\in\mathbb{N}}$ si $p > 0$ et $(r_i = d + ip \mid r_i > f)_{i\in\mathbb{N}}$ si $p < 0$. Le paramètre `p` est optionnel avec une valeur par défaut de 1.
`list(range(1, 5))` → `[1, 2, 3, 4]` ; `list(range(20, 10, -2))` → `[20, 18, 16, 14, 12]`.

— `math.sqrt(x)` calcule la racine carrée du nombre `x`.

Opérations sur les listes

— `len(L)` donne le nombre d'éléments de la liste `L`.

— `L1 + L2` construit une liste constituée de la concaténation des listes `L1` et `L2`.

— `e in L` et `e not in L` déterminent si l'objet `e` figure dans la liste `L`. Ces opérations ont une complexité temporelle en $O(\texttt{len}(\texttt{L}))$.

— `L.append(e)` ajoute l'élément `e` à la fin de la liste `L`.

— `L.index(e)` renvoie le plus petit entier `i` tel que `L[i] == e`. Lève l'exception `ValueError` si l'élément `e` n'apparait pas dans la liste. Cette opération a une complexité temporelle en $O(\texttt{len}(\texttt{L}))$.

— `L.sort()` trie en place la liste `L` (qui est donc modifiée) en réordonnant ses éléments dans l'ordre croissant.

Opérations sur les tableaux (`np.ndarray`)

— `np.array(s, dtype)` crée un nouveau tableau contenant les éléments de la séquence `s`. La taille de ce tableau est déduite du contenu de `s`. Le paramètre optionnel `dtype` précise le type des éléments du tableau créé.

— `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un tableau à une dimension de `n` éléments et un tableau à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype`. Si le paramètre `dtype` n'est pas précisé, il prend la valeur `float`.

— `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens.

— `np.sum(a)` ou `a.sum()` renvoie la somme des éléments du tableau `a`.

— `np.inner(a, b)` calcule la somme des produits terme à terme dans le cas où `a` et `b` sont deux tableaux à une dimension de même taille.

— `np.all(a)` vaut `True` si tous les éléments du tableau `a` ont une valeur logique « vrai ».

— `np.any(a)` vaut `True` si au moins un des éléments du tableau `a` a une valeur logique « vrai ».

SQL

— `SELECT … FROM t1 JOIN t2 ON …` effectue une requête sur le résultat du produit cartésien entre les tables `t1` et `t2` restreint par la condition indiquée. Par exemple `t1.a = t2.b` permet de limiter le résultat aux lignes pour lesquelles la colonne `a` de la table `t1` est égale à la colonne `b` de la table `t2`.

— `SELECT … FROM t AS t1 JOIN t AS t2 ON …` effectue une requête sur le résultat du produit cartésien de la table `t` avec elle-même restreint par la condition indiquée. Le premier exemplaire de la table `t` est désigné par `t1` et le second par `t2`.

— La fonction `EXTRACT(`*part* `FROM t)` extrait un élément de `t`, expression de type `date`, `time`, `timestamp` (jour et heure) ou `interval` (durée). *part* peut prendre les valeurs `year`, `month`, `day` (jour dans le mois), `doy` (jour dans l'année), `dow` (jour de la semaine), `hour`, etc.

— Les fonctions d'agrégation `SUM(e)`, `AVG(e)`, `MAX(e)`, `MIN(e)`, `COUNT(e)`, `COUNT(*)` calculent respectivement la somme, la moyenne arithmétique, le maximum, le minium, le nombre de valeurs non nulles de l'expression `e` et le nombre de lignes pour chaque groupe de lignes défini par la cause `GROUP BY`. Si la requête ne comporte pas de clause `GROUP BY` le calcul est effectué pour l'ensemble des lignes sélectionnées par la requête.

• • • FIN • • •
