Imagerie par lancer de rayons

David Monniaux

David.Monniaux@ens.fr

IMPORTANT

On trouvera sur http://www.enseignement.polytechnique.fr/profs/informatique/David.Monniaux/ray-tracing une page avec des renseignements complémentaires, notamment pour spécifier des points qui auraient été trop laborieux dans l'énoncé.

1 Principe de base

Nous désirons produire une image de synthèse en simulant informatiquement le trajet des rayons lumineux entre les surfaces illuminées et l'œil (ou la caméra) de l'observateur, d'où le nom de la technique, le *lancer de rayons* (*ray-tracing* en anglais).[1]

Nous allons en fait simuler le trajet inverse du rayon depuis l'œil de l'observateur (Fig. 1) : après être parti de l'œil en direction d'un point sur l'écran virtuel, le rayon atteint un objet : c'est la couleur de la surface de cet objet que nous allons afficher. Nous verrons plus tard comment calculer cette couleur en fonction des conditions d'éclairement.

Plus mathématiquement : on se donne une position C de la caméra, une direction $\vec{u}$ dans laquelle elle regarde, et deux vecteurs $\vec{v}$ et $\vec{w}$. Soient L et H respectivement la largeur et la hauteur en pixels de l'écran. Il faut alors imaginer l'image produite tendue comme un écran dont le centre est en $C + \vec{u}$ et dont la largeur et la hauteur sont dirigées par $\vec{v}$ et $\vec{w}$. Le pixel de coordonnées (x, y) correspond donc au point de coordonnées $C + \vec{d}(x, y)$ où

$$\vec{d}(x,y) = \vec{u} + \left(\frac{x + 1/2}{L} - \frac{1}{2}\right)\vec{v} + \left(\frac{y + 1/2}{H} - \frac{1}{2}\right)\vec{w}.$$

Le rayon lumineux correspondant est $C + \vec{d}(x, y)\mathbb{R}_+$.

2 Spécification des objets

Il nous faut un moyen pour l'utilisateur de spécifier des objets géométriques. Nous avons choisi de considérer des objets formés par combinaisons logiques (union, intersection...) de primitive géométriques.

[1] Il ne s'agit pas de la seule technique permettant de produire des images de synthèse. Les scènes d'animation tridimensionnelles des jeux sont généralement produites à l'aide de techniques plus rapides, mais moins basées sur la simulation physique (tracés en Z-buffer...). Il existe par ailleurs des techniques de simulation physique beaucoup plus puissantes et réalistes... mais aussi très coûteuses en temps de calcul et en complexité d'implémentation, telles que les techniques de radiosité.

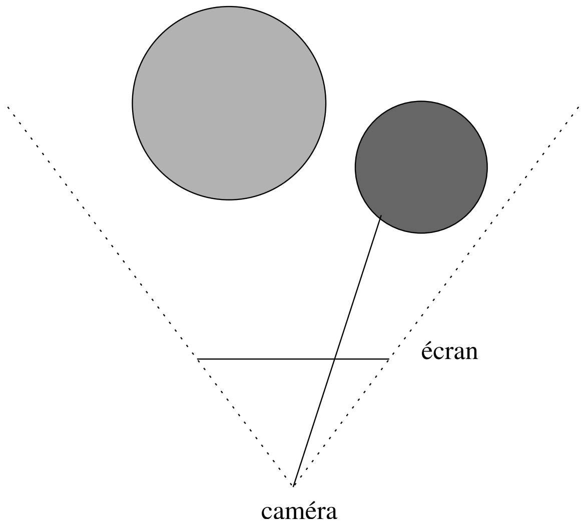

FIG. 1 – Trajet de la lumière depuis les objets (boule) jusqu'à la caméra, en passant par l'écran.

2.1 Géométrie solide constructive

Nous spécifions les objets de la scène en géométrie solide constructive (*constructive solid geometry* ou *CSG*), c'est-à-dire comme des volumes que l'on peut combinet à l'aide d'opérateurs logiques $\wedge$ (et), $\vee$ (ou) et $\neg$ (négation). Chaque objet de la scène est donc représenté sous la forme d'un arbre dont les nœuds sont les opérateurs logiques et les feuilles des primitives solides. Supposons par exemple que nous disposons de primitives pour un cylindre infini défini par $x^2 + y^2 \leq 1$ et pour les demi-espaces $z \leq 1$ et $z \geq -1$; l'intersection des trois nous fournira un cylindre de longueur 2.

Les opérations suivantes doivent être implémentées :

- en fonction d'un point (x, y, z), dire s'il appartient au solide (en fait, on peut se passer de cette opération) ;
- en fonction d'un rayon $O + \mathbb{R}\vec{v}$ donné par les coordonnées de O et $\vec{v}$, obtenir la liste des coordonnées t décrivant les intersections $O + t\vec{v}$ avec le solide.

On négligera les problèmes de précision numérique s'ils deviennent complexes à traiter.

Les deux opérations à implémenter le seront par induction sur l'arbre représentant la formule en géométrie solide constructive (ainsi, la liste des intersections d'un rayon R avec l'union de deux objets A et B peut facilement s'obtenir à partir des listes d'intersection de R avec ces deux objets).

2.2 Primitives solides

Nous considérerons des objets définis par des inéquations polynômiales de la forme $P(x, y, z) \leq 0$. Ainsi, une boule de rayon 1 centrée en $(0, 0, 0)$ sera définie par $x^2 + y^2 + z^2 \leq 0$; un demi-espace dont la frontière est située à une distance d du point $(0, 0, 0)$ et dont le vecteur normal unitaire tourné vers l'extérieur est (v_x, v_y, v_z) sera défini par $v_x x + v_y y + v_z z - d \leq 0$.

On pourra considérer plus généralement les quadriques définies par $Q(x, y, z) + L(x, y, z) + d \leq 0$ où Q est une forme quadratique, L une forme linéaire et d un scalaire ; suivant la signature de la forme quadratique, on obtiendra des ellipsoïdes,

des hyperboloïdes à une nappe, des hyperboloïdes à deux nappes ou des quadriques dégénérées.

Il pourra être plus facile de considérer une quadrique projective (forme quadratique Q en dimension 4 et résolution de $Q(x, y, z, 1) = 0$). Un tel formalisme facilite beaucoup les opérations de translations, rotations etc..., qui seront vues comme des matrices à multiplier à la forme quadratique.

Remarquons que calculer les intersections d'un rayon $O + \mathbb{R}\vec{d}$ avec la surface définie par P est équivalent à chercher les zéros en t du polynôme $P(O + t\vec{d})$, lequel est au plus du même degré que le degré total de P.

2.3 Transformations géométriques

On permettra d'appliquer des transformations géométriques telles que translation, rotations, facteur d'échelle sur les trois axes. Ces opérateurs sont des opérateurs affines ; par souci d'homogénéité, nous définirons les opérations comme des transformations projectives, associant à un point $(x, y, z, 1)$ son image $(x', y', z', 1)$:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M. \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1}$$

On permettra en particulier les opérations suivantes

Translation Pour une translation de vecteur (τ_x, τ_y, τ_z) :

$$M = \begin{bmatrix} 1 & 0 & 0 & \tau_x \\ 0 & 1 & 0 & \tau_y \\ 0 & 0 & 1 & \tau_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Rotation Pour une rotation de θ radians autour de l'axe Z (on prévoiera une spécification en degrés pour l'utilisateur) :

$$M = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

On permettra aussi les rotations autour des axes X et Y.

Mise à l'échelle Pour une mise à l'échelle de facteurs respectifs α_x, α_y, et α_z sur les axes X, Y et Z,

$$M = \begin{bmatrix} \alpha_x & 0 & 0 & 0 \\ 0 & \alpha_y & 0 & 0 \\ 0 & 0 & \alpha_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

La composition de plusieurs transformations se fait par produit de matrices. Il pourra être utile d'implémenter une petite bibliothèque de calcul vectoriel et matriciel.

Soit un objet A tel que la liste des t tels que $O + t\vec{v}$ soit une intersection du rayon $O + \mathbb{R}\vec{v}$ avec la surface de O soit donné par $\mathrm{intersect}(A, O, \vec{v})$. Nous nous intéressons à l'image $M.A$ de l'objet A par la transformation affine de matrice projective A. Alors,

$$\mathrm{intersect}(M.A, O, \vec{v}) = \mathrm{intersect}(A, \pi_3(M^{-1}.\iota_4^1(O)), \pi_3(M^{-1}.\iota_4^0(\vec{v}))) \tag{5}$$

où $\pi_3(x,y,z,1)=(x,y,z)$, $\iota_4^t(x,y,z)=(x,y,z,t)$.

Illustrons cela sur une quadrique A définie par $Q(x,y,z,1)=0$.Notant Q^* la forme bilinéaire associée à la forme quadratique Q, l'équation définissant les intersections intersect$(A,O,\vec{v})$ est :

$$Q(\iota_4^1(O))+Q^*(\iota_4^1(O),\iota_4^0(\vec{v}))t+Q(\iota_4^1(\vec{v}))t^2=0 \tag{6}$$

On en déduit que l'objet $M.A$ est défini par la forme quadratique ${}^tM^{-1}.Q.M^{-1}$. En fait, nous n'avons besoin que de M^{-1} et non de M.

3 Modèle d'illumination

Afin d'obtenir un résultat réaliste, nous associerons à chaque point vu à la surface des objets une couleur. Par souci de simplicités, les couleurs seront calculées comme un triplet (R,V,B) représentant la décomposition de la couleur en composantes rouge, verte et bleue ; la superposition additive de deux couleurs sera simplement l'ajout des vecteurs.

Le calcul de la couleur d'un point pourra se faire à l'aide du modèle suivant, version simplifiée de celle du logiciel POV-Ray. Notons
- $\vec{N}$ le vecteur unitaire normal de la surface pointant vers l'extérieur (en supposant que la caméra soit à l'extérieur).
- $\vec{V}$ le vecteur unitaire menant de la caméra au point considéré de la surface.
- $\vec{L}$ le vecteur unitaire menant du point considéré vers la source lumineuse primaire.

Diffusion Il s'agit de la composante principale de l'éclairement. Son intensité dépend de $I_d=(\vec{N}.\vec{L})^b$ où b est un exposant de « brillance » (1 par défaut) ;

Phong Il s'agit d'une modélisation du phénomène de reflet direct d'une source lumineuse sur une surface luisante : $I_p=(\vec{R}.\vec{L})^s$ où $R=-2(\vec{V}.\vec{N})\vec{N}+\vec{V}$ est le symétrique du vecteur $-\vec{V}$ par rapport au vecteur $\vec{N}$;

Spéculaire Une autre modélisation du même phénomène : $I_s=(\vec{H}.\vec{N})^{1/r}$ où $\vec{H}=\frac{\vec{L}-\vec{V}}{\|\vec{L}-\vec{V}\|}$ est le vecteur bisecteur normalisé de $-\vec{V}$ et de $\vec{L}$; r indique la « rugosité » du matériau ($1/r=30$ donne des résultats intéressants).

On pourra ajouter la modélisation de phénomènes de réflexion, de réfraction et d'ombres portées.

Nous rappelons qu'en ce qui concerne une surface de la forme $Q(P)=0$ où Q est une forme quadratique, un vecteur normal dirigé vers les $Q(P)>0$ est donné par $2\vec{u}$ où $Q^*(P,\vec{u})=P.\vec{u}$, c'est à dire que $\vec{u}$ est l'image de P par la matrice symétrique définissant Q.

4 Travail demandé

Le logiciel réalisé devra prendre un entrée une description de la scène écrit dans un langage simple et produire en sortie une image. Le langage d'entrée devra permettre de spécifier facilement des objets construits en CSG.

Une description d'une grammaire d'entrée fortement recommandée sera donnée sur la page WWW du projet.

Voici un exemple de fichier source, pour un petit logiciel de rendu réalisé en Java par l'auteur de l'énoncé :

```
union {
  applyTexture(
    solid {
      diffusion <0.32, 0.32, 0.9>
```

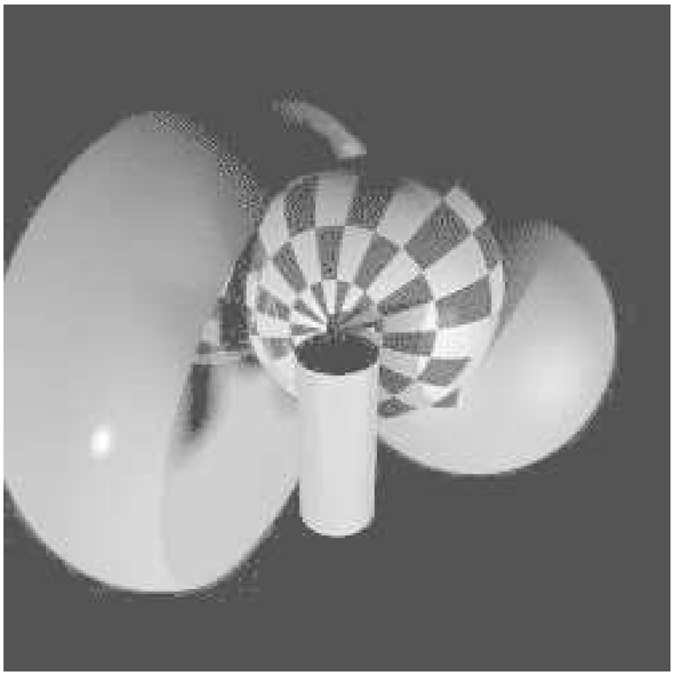

FIG. 2 – Exemple de scène

```
      specular <0.3, 0.3, 0.3>
    },
    translate (<1, 0, 0>, sphere1 ))
  applyTexture(
    solid {
      diffusion <0.8, 0.3, 0.3>
      specular <0.8, 0.8, 1.0>
      specularExponent 1000
      reflexion <0.4, 0.5, 0.4>
    },
    translate (<-1, 0, 0.5>,
      rotateZ(20, scale (<1.5, 2.0, 2.0>,
rotateY(20,
intersection {
           translate(<0.5, 0, 0>, halfSpace(<1,0,0>)) sphere1 } )))))
    translate (<0.2, -0.3, -0.9>,
      rotateZ(40,
        rotateX(25,
          applyTexture(
            checker(cylindrical(scale(<0.1, 0.5, 1>, id)), {
              solid {
                diffusion <0.32, 0.9, 0.4>
                specular <0.3, 0.3, 0.3>
              }
              solid {
                diffusion <0.1, 0.1, 0.1>
                specular <0.9, 0.9, 0.9>
                specularExponent 1000
        reflexion <0.9, 0.9, 0.9>
              }}),
          scale (0.8, sphere1 )))))
  applyTexture(
    solid {
      diffusion <0.5, 0.5, 0.5>
      specular <0.3, 0.3, 0.3>
```

```
      specularExponent 50
    },
    translate(<0, 0.5, -1.6>,
      rotateX(30,
        intersection { scale(0.2, cylinderY1)
               translate(<0,-0.5,0>, halfSpace(<0,-1,0>))
               translate(<0,0.5,0>, halfSpace(<0,1,0>))} )))
}
```

décrivant un objet qui, éclairé par deux lumières, donne l'image en Fig. 2.

On trouvera d'autres exemples sur la page du projet.