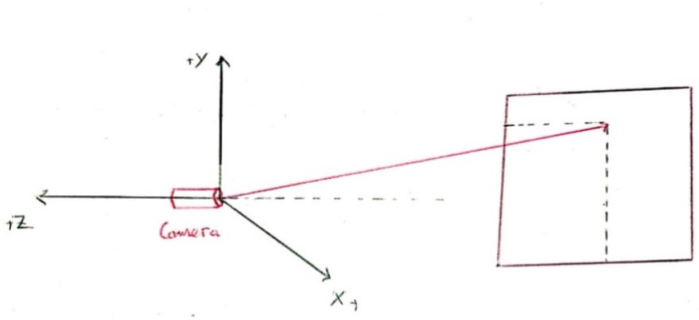
Projet synthèse d'image

PAR SABIR ILYASS

Dans toute la suite, on positionne la caméra en position $\mathbf{(0,0,0)}$ dans le repère de l'espace et la scène de vue dans le plan d'équation $\boldsymbol{z=-1}$.



On reprend le même code de **Vec3.*h***, **Color.*h*** et **Ray.h** ces codes sont modifiés eu cours de l'avancement du projet.

Pour automatiser la création des images .ppm, j'ai créé une classe **ImagePPM**, le but de cette classe est de créer une **image.ppm** dans un répertoire donnée en argument et d'ajouter les valeurs aux pixels de cette image.

Ce que j'ai développé personnellement est: la classe **ImagePPM**, la classe **ListSphere**, la classe **Triangle**, la classe **Texture**, la méthode **applyTexture** de la classe **Sphere** avec d'autres méthodes pour appliquer le rendu 3D aux sphères (voir la partie 3) et la méthode inline **reflection** dans la classe **Vec3**.

Après l'exécution du code, pour créer la figure(i), il faut entrée le nombre i pour $i \in \{1,2,3,4,5,6\}$.

1 Représentation d'une sphère

Le premier objectif du projet est de tracer une image en blanc et noir, où la partie blanche doit correspondre à l'emplacement de la sphère vue dans la scène par rapport à la caméra.

Donc il suffit ici de tester s'il y a une intersection rayon/sphère ou pas. Pour cela on reprend la définition du rayon, un rayon r est tout simplement un droite, d'origine O_r, et de direction D_r.

Un point $P \in r$ si et seulement s'il existe $t \in \mathbb{R}$ tel que $P = O_r + tD_r$.

Une sphère S est définie par son centre C_s et son rayon R_s.

Un point $P \in S$ si et seulement si $\|P - C_s\| = R_s$ où $\|.\|$ est la norme euclidienne de $\mathbb{R}^3$.

Donc il y a une intersection entre le rayon r et la sphère S si et seulement s'il existe $t \in \mathbb{R}$ tel que:

$$\|P - C_s\| = R_s$$

Et donc le problème pourrait se traduit par:

Est-ce que l'équation d'inconnue t: $\|O_r + t\,D_r - C_s\| = R_s$ (1) admet-elle des solutions?

Notons $\langle .|. \rangle$ le produit scalaire associé à $\|.\|$.

Donc l'équation (1) devient:

$$\begin{aligned} \|O_r + t\,D_r - C_s\| = R_s &\Leftrightarrow \langle O_r + t\,D_r - C_s | O_r + t\,D_r - C_s \rangle = R_s^2 \\ &\Leftrightarrow \|O_r - C_s\|^2 + 2t\langle D_r | O_r - C_s \rangle + \|D_r\|^2 t^2 = R_s^2 \quad (2) \end{aligned}$$

Les équations (1) et (2) sont équivalentes, et donc l'intersection rayon-sphére revient à résoudre une équation simple de trinôme.

Pour que (2) admet des solutions réelles si et seulement si
$\langle D_r | O_r - C_s \rangle^2 - (\|O_r - C_r\|^2 - R_s^2)\|D_r\|^2 \geqslant 0$.

Voici la figure qu'on a obtenue après l'exécution du code **figure1()**.

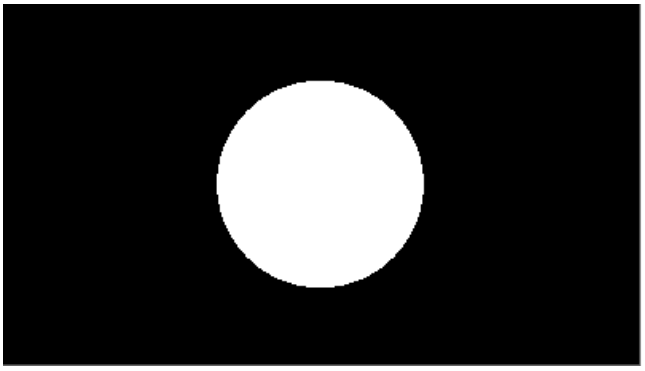


Figure 1. Représentation d'une sphère.

2 Représentation de plusieurs sphères

L'objectif de cette seconde partie du projet est d'afficher plusieurs sphères dans la scène.

On doit normalement afficher que les sphères les plus proches à la caméra, et donc savoir s'il y a intersection ou par entre rayon / sphère ne suffit pas, il faut connaître aussi les points d'intersections.

Et donc à partir de l'équation (2), si l'équation admet des solutions réelles

Les solutions de l'équation (2) sont $t_- = \dfrac{\langle D_r | O_r - C_s \rangle - \sqrt{\langle D_r | O_r - C_r \rangle^2 - (\|O_r - C_r\|^2 - R_s^2)\|D_r\|^2}}{\|D_r\|^2}$ et $t_+ = \dfrac{\langle D_r | O_r - C_s \rangle + \sqrt{\langle D_r | O_r - C_r \rangle^2 - (\|O_r - C_r\|^2 - R_s^2)\|D_r\|^2}}{\|D_r\|^2}$

La solution qui nous intéresse est la solution qui donne le point d'intersection le plus proche de la caméra.

Pour $N \in \mathbb{N}^\star$, pour une famille de sphères $(S_i(C_{s_i}, R_{s_i}))_{1 \leqslant i \leqslant N}$ où C_{s_i} et R_{s_i} désigne respectivement le centre et le rayon de la sphère S_i, $\forall i = 1, \ldots, n$

Après calcul des $(t_{i-}, t_{i+})_{1 \leqslant i \leqslant N}$ pour savoir les points d'intersections rayon/sphère ($t_{i-} = t_{i+} = -1$ dans le cas ou l'intersection entre le rayon et la sphère S_i est vide).

Puis on cherche la sphère la plus proche de la camèra

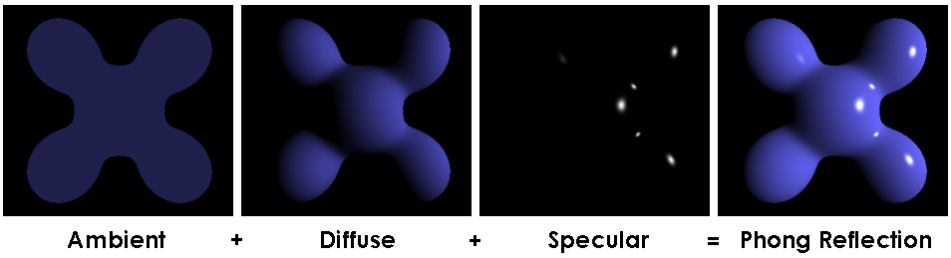
$$i_0 = \underset{1 \leqslant i \leqslant N}{\operatorname{argmax}}(t_{i+})$$

On attribue à ce point d'intersection la couleur de la sphère S_{i_0}.
Voici la figure qu'on a obtenue après l'exécution du code **figure2()**.

Figure 2. Représentation de plusieurs sphères.

3 Ajout du Rendu 3D aux sphères (modèle Phong)

Pour ajouter le rendu 3D à une sphère, il faut calculer les couleurs: **Ambient**, **Diffuse** et **Specular**.



Ambient est tout simplement la couleur de la sphère multiplie par un coefficient d'ambiance.
La couleur diffuse et la couleur specular dépendent de la couleur de la position de la lumière et du point d'intersection rayon/sphère.
Notons $\boldsymbol{L}$: le vecteur unitaire de la droite qui vient de la lumière vers le point d'intersection (rayon / sphère).
N: le vecteur unitaire normal au point d'intersection (rayon / sphère).
Pour calculer la couleur diffuse, on applique la formule simplifiée (la formule dans le cas de plusieurs sources de lumière https://en.wikipedia.org/wiki/Phong_reflection_model)

$$\text{diffuse} = \text{Coeff}_{\text{diffuse}} \times (\langle L|N\rangle) \times (\text{couleur de la sphère})$$

Où $\textbf{Coeff}_{\textbf{diffuse}}$ est le coefficient de diffusion.
Il reste à calculer la couleur specular, pour çela on applique la formule dans le cas d'un seul source de lumière (la formule dans le cas de plusieurs sources de lumières https://en.wikipedia.org/wiki/Phong_reflection_model)

$$\text{specular} = \text{Coeff}_{\text{specular}} \times (\langle D_r|\text{reflection}(L,N)\rangle)^{\text{shiness}} \times (\text{couleur de la lumière})$$

Où **shiness** et $\textbf{Coeff}_{\textbf{specular}}$ sont des coefficients constants.
reflection$(\boldsymbol{L},\boldsymbol{N})$ représente le vecteur de réflexion la lumière à la surface de la sphère.
Voici la figure qu'on a obtenue après l'exécution du code **figure3()**.

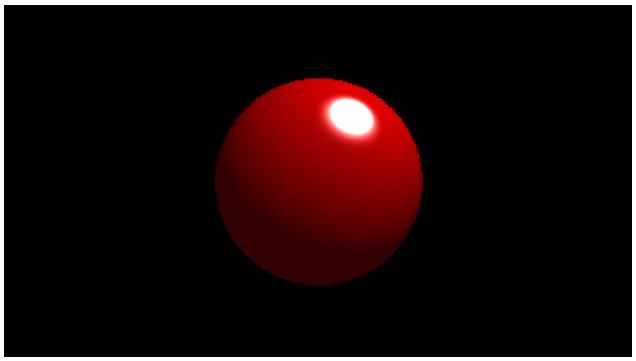


Figure 3. Ajout du Rendu 3D aux sphères (modèle Phong).

4 Arbre de sphères englobantes.

L'objectif de cette partie est de construire un arbre de sphères englobantes afin de réduire le nombre de tests d'intersections sphère rayon.

Pour $N \in \mathbb{N}^\star$ sphères $(S_i(C_{s_i}, R_{s_i}))_{1\leqslant i\leqslant N}$ où C_{s_i} et R_{s_i} désigne respectivement le centre et le rayon de la sphère S_i, $\forall i = 1, \ldots, n$

Notons **SphEng$((S_i)_{1\leqslant i\leqslant r})$**: la sphère englobante de $(S_i)_{1\leqslant i\leqslant r}$. $(r \in \mathbb{N}^\star)$.

On construit l'arbre de sphère englobant suivante

$$\mathrm{SphEng}((S_i)_{1\leqslant i\leqslant N})\begin{cases}\mathrm{SphEng}\Big((S_i)_{1\leqslant i\leqslant \frac{N}{2}}\Big)\begin{cases}\mathrm{SphEng}\Big((S_i)_{1\leqslant i\leqslant \frac{N}{4}}\Big)\begin{cases}\mathrm{SphEng}\Big((S_i)_{1\leqslant i\leqslant \frac{N}{8}}\Big)\cdots\\ \mathrm{SphEng}\Big((S_i)_{\frac{N}{8}+1\leqslant i\leqslant \frac{N}{4}}\Big)\cdots\end{cases}\\ \mathrm{SphEng}\Big((S_i)_{\frac{N}{4}+1\leqslant i\leqslant \frac{N}{2}}\Big)\begin{cases}\mathrm{SphEng}\Big((S_i)_{\frac{N}{4}+1\leqslant i\leqslant 3\frac{N}{8}}\Big)\cdots\\ \mathrm{SphEng}\Big((S_i)_{3\frac{N}{8}+1\leqslant i\leqslant \frac{N}{2}}\Big)\cdots\end{cases}\end{cases}\\ \mathrm{SphEng}\Big((S_i)_{\frac{N}{2}+1\leqslant i\leqslant N}\Big)\begin{cases}\mathrm{SphEng}\Big((S_i)_{\frac{N}{2}+1\leqslant i\leqslant 3\frac{N}{2}}\Big)\begin{cases}\mathrm{SphEng}\Big((S_i)_{\frac{N}{2}+1\leqslant i\leqslant 5\frac{N}{8}}\Big)\cdots\\ \mathrm{SphEng}\Big((S_i)_{5\frac{N}{8}+1\leqslant i\leqslant 3\frac{N}{2}}\Big)\cdots\end{cases}\\ \mathrm{SphEng}\Big((S_i)_{3\frac{N}{2}+1\leqslant i\leqslant N}\Big)\begin{cases}\mathrm{SphEng}\Big((S_i)_{3\frac{N}{2}+1\leqslant i\leqslant 7\frac{N}{8}}\Big)\cdots\\ \mathrm{SphEng}\Big((S_i)_{7\frac{N}{8}+1\leqslant i\leqslant N}\Big)\cdots\end{cases}\end{cases}\end{cases}$$

Le code de SphEng est dans la classe **ListSphere**, il est basé sur le code de sphère englobante qu'on a vue dans le cours adapté au cas 3D, j'ai évité d'utiliser une bibliothèque d'algèbre linéaire pour les calculs matriciels (pour que le code fonction sans installer une bibliothèque).

5 Triangles

Un triangle est définie par le donné de trois points P_1, P_2 et P_3.

On cherche dans un premier temps l'équation du plan du triangle de la forme $ax+by+cz=1$

Les 3 points P_1, P_2 et P_3 sont dans le plan, et donc

$$\begin{cases} ap_{1x}+bp_{1y}+cp_{1z}=1 \\ ap_{2x}+bp_{2y}+cp_{2z}=1 \\ ap_{3x}+bp_{3y}+cp_{3z}=1 \end{cases}$$

Donc

$$\begin{pmatrix} p_{1x} & p_{1y} & p_{1z} \\ p_{2y} & p_{2y} & p_{2z} \\ p_{3z} & p_{3y} & p_{3z} \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix}=\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Donc il suffit de calculer l'inverse de $\begin{pmatrix} p_{1x} & p_{1y} & p_{1z} \\ p_{2x} & p_{2y} & p_{2z} \\ p_{3x} & p_{3y} & p_{3z} \end{pmatrix}$, pour éviter le problème d'inversion de la matrice, j'ai choisi d'utiliser le formule de Moore-Penrose pour calculer le pseudo-inverse.

Et donc

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix}=\begin{pmatrix} p_{1x} & p_{1y} & p_{1z} \\ p_{2x} & p_{2y} & p_{2z} \\ p_{3x} & p_{3y} & p_{3z} \end{pmatrix}^{-1}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Le point d'intersection triangle/rayon est determiné d'abord par l'obtention du point d'intersection rayon/plan de triangle, puis on teste si ce point est à l'intérieur du triangle ou pas.

La fonction **insideTriangle** de classe **Triangle** permet de renvoyer true si le point est à l'intérieur du triangle et false sinon. La solution qui nous intéresse est la solution qui donne le point d'intersection le plus proche de la caméra.

On calcule pour le point P d'intersection **rayon/plan de triangle**, si les triplets de points (P_1, P_2, P_3), (P_1, P_2, P) et (P_2, P_3, P_1), (P_2, P_3, P) et (P_3, P_1, P_2), (P_3, P_1, P) sont tous orienté dans la même direction.

Voici la figure qu'on a obtenue après l'exécution du code **figure4()**.

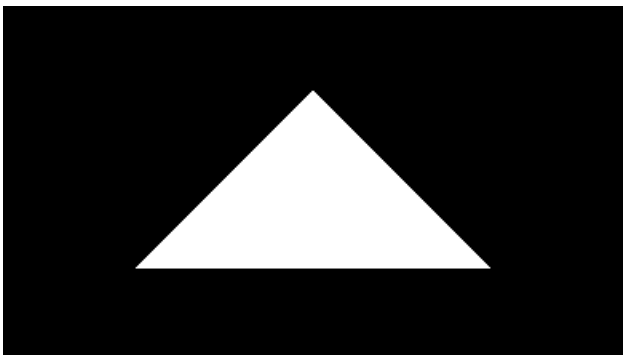


Figure 4. Triangle

On colorie le triangle par interpolation des couleurs (rouge, vert, bleu), les coefficients de l'interpolation sont calculé à partir des coefficients de barycentre $(\alpha, \beta, 1-\alpha-\beta)$ de point P d'intersection de triangle/rayon.

$$P=\alpha P_1+\beta P_2+(1-\alpha-\beta)P_3$$

Voici la figure qu'on a obtenue après l'excution du code **figure5()**.

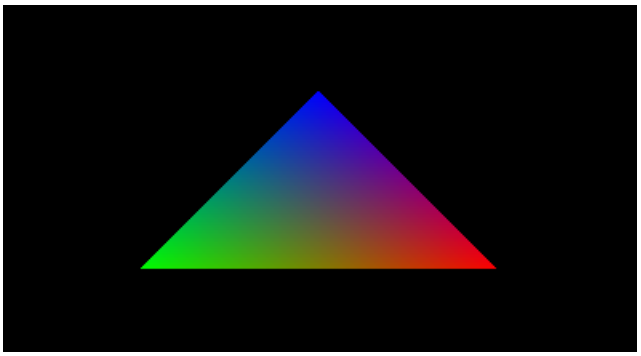


Figure 5. Interpolation de couleurs.

6 Textures

L'objectif de cette partie est d'ajouter des textures à nos sphères.

Pour rester dans le même objectif de ne rien installer, j'ai créé une fonction Matlab **PNG2PPM.*m*** pour transformer une image (jpg, png, tif ...) en ppm.

La lecture de l'image ppm en C++ est comme la lecture de fichier, on utilise la bibliothèque **fstream** pour la lecture de l'image .ppm

Ainsi j'ai créé une classe **Texture** pour enregistrer dans un « vector » les valeurs des pixels de l'image donnée à l'entrée.

Pour mapper la texture de l'image ppm sur la sphère, on utilise les coordonnées sphériques afin de trouver (u, v) position de pixel de la texture 2D.

Au point P d'intersection rayon/sphère

On calcule le vecteur N normal au point d'intersection.

$$u = W \times \left(\frac{1}{2} + \frac{\text{acrtan2}(N_z, N_x)}{2\pi} \right)$$

$$v = H \times \left(\frac{1}{2} - \frac{\arcsin(N_y)}{\pi} \right)$$

Où (W, H) la taille de l'image de la texture.

Voici la figure qu'on a obtenue après l'exécution du code **figure6()**.

Figure 6. texture appliquée à une sphère.