Rendu rapide de scène 3D

Cassiopée Gossin - L1 CMI informatique
Visi 201

Tuteur : Colin Weill-duflos

Sommaire :

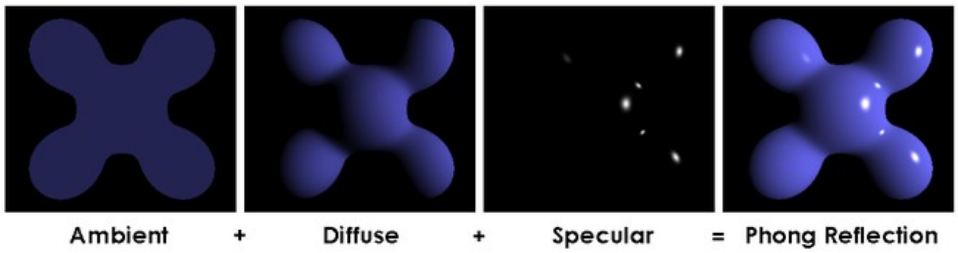
Introduction :

Un rendu de scène 3D est une méthode pour faire une projection d'une scène en 3D sur une image en 2D avec un certain point de vue qui est la caméra. Pour avoir un rendu de cette scène en temps réel, il faut implémenter des méthodes comme la rastérisation de triangles pour représenter des objets avec des triangles et le z-buffer pour appliquer de la profondeur (méthodes qui ont déjà été réalisés par autre étudiant de L1 en 2017, Tournafond Raphaël. Son travail est disponible sur le lien suivant : https://www.lama.univ-savoie.fr/mediawiki/index.php/Algorithme_de_rendu_de_sc%C3%A8ne_3D_par_Z-buffer).

Ces méthodes de rendu en temps réel (utilisés dans les jeux vidéo par exemple) sont opposés au rendu différé (utilisé pour les films par exemple) où il y a plusieurs minutes voire plusieurs heures de calculs pour une image.
Pour rendre ce rendu plus réaliste, il faut appliquer de la lumière à notre scène, donc appliquer de la luminosité et une ombre projetée aux objets de la scène. Mais il faut également la possibilité d'appliquer une texture et de représenter des objets ronds, comme une sphère.

I/ Les 3 lumières (modèle d'illumination de Phong) :

Pour rendre le modèle 3D plus réaliste, il faut ajouter de la lumière à ce rendu. Pour cela il faut utiliser le modèle d'illumination de Phong (https://fr.wikipedia.org/wiki/Ombrage_de_Phong) qui consiste en 3 composantes pour construire une lumière assez convaincante, qui sont la lumière diffuse, la lumière ambiante et la lumière spéculaire.



Explication du modèle d'illumination de Phong

1) La lumière diffuse :

La lumière diffuse représente l'illumination de l'objet qui vient directement de la lumière et qui se reflète dans toutes les directions. Le principe pour retrouver cette lumière est de savoir pour chaque point de la figure leur angle par rapport à la source de lumière (donc si le point fait face à la lumière ou non), et d'appliquer un certain taux de luminosité en conséquence. Pour cette lumière il faut donc :

- Calculer le vecteur normal de ce triangle, ce qui consiste à faire un produit vectoriel entre 2 côtés de ce triangle (vecteur N)
- Puis calculer le vecteur qui va du point jusqu'à la source de lumière, (vecteur L)
- Normaliser ces deux vecteurs pour "retirer" la distance des vecteurs et ne garder que la direction de ces vecteurs (donc ne garder que la partie qui nous intéresse). Donc diviser le vecteur N par sa norme et le vecteur L par sa norme : $N \div ||N||$ et $L \div ||L||$
- Puis faire un produit scalaire avec ceux-ci, ce qui va nous donner un résultat entre -1 et 1.
- Ensuite prendre le maximum entre le résultat et 0, car quand le résultat est négatif cela veut dire qu'il faut regarder de l'autre coté de la face illuminé pour voir la lumière. Et en prenant le maximum nous faisons le choix de ne pas laisser la lumière traverser la face illuminée. Le

maximum donne donc une valeur comprise entre 0 et 1, ce qui traduit à quel point le point est illuminé (plus ce coefficient est proche de 1, plus le point est illuminé).

- Puis ajouter ce coefficient (kd) au calcul des couleurs RGB.

La formule est la suivante : $kd = \max ((N \div ||N||) . (L \div ||L||) , 0)$

Exemples de triangles ayant la lumière diffuse selon des sources de lumière de différentes coordonnées (précisées en dessous de chaque triangle) :

(1., 0.5, -0.5)
Illuminé de face

(2., 0., -0.5)
Illuminé vers la pointe bleue

(0., 2., -0.5)
Illuminée vers la pointe rouge

2) La lumière ambiante :

La lumière ambiante représente la lumière qui retourne sur la figure après avoir rebondi sur d'autres objets. C'est donc une lumière assez faible qui est présente de manière uniforme sur l'objet. Le principe pour retrouver cette lumière est simplement d'ajouter un coefficient arbitraire assez faible (ka, entre 0 et 1) au calcul des couleurs RGB. Le coefficient choisi est ici de 0,1.

Triangle ayant la lumière ambiante (coefficient = 0.1)

3) La lumière spéculaire (approximation de Blinn-Phong) :

La lumière spéculaire est la lumière renvoyée dans la direction de la caméra (donc les reflets), donc plus le vecteur de lumière va sur la camera, plus le point est clair voire blanc. Pour retrouver cette lumière il faut utiliser l'approximation de Blinn-Phong (https://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_reflection_model), qui se compose de 2 formules :

- Le vecteur H dit "Halfway" qui représente la "moitié" du parcours entre le vecteur light et le vecteur vue :

$$H = (L+V) \div (||L+V||)$$

où L est le vecteur "light" qui est le vecteur qui va du point jusqu'à la source de lumière, et V est le vecteur "vue" qui est le vecteur qui va du point jusqu'à la camera. On divise par la norme pour avoir un vecteur normalisé.

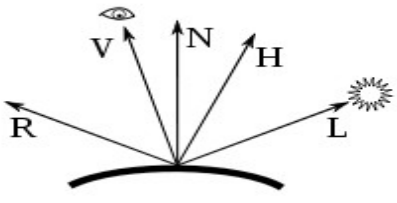


Schéma explicatif de l'approximation de Blinn-Phong

- Et le coefficient de la lumière spéculaire s'obtient avec la formule :

  (H.N)**α

où H est le vecteur "halfway", N est la normale du triangle auquel le point appartient et α est une puissance qui dépend du matériaux de l'objet (plus α est élevé et plus le matériau est lisse et donc plus spéculaire sera petit et lumineux. A l'inverse, plus α est petit et plus le spéculaire sera large et diffus).

Puis ajouter ce coefficient (ks) au calcul des couleurs RGB.

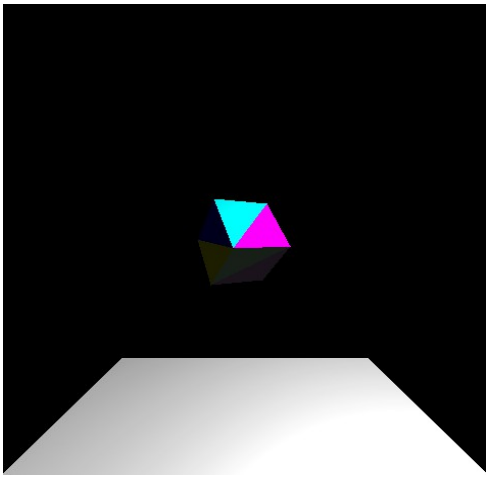
Triangle ayant la lumière spéculaire (coordonnées de la source de lumière : (2., 0. -0.5,), alpha = 10)

4) La formule finale :

La formule finale pour le calcul des couleurs en ajoutant les 3 modèles de lumières expliqués précédemment est la suivante : (R, G, B représentent les composantes RGB du point dont on veut ajouter la luminosité)

couleur = ((kd + ka + ks) * R , (kd + ka + ks) * G , (kd + ka + ks) * B)

Triangle ayant toutes les composantes de lumière (diffuse, ambiante et spéculaire)



Cube ayant toutes les composantes de lumière

II/ Ombre projetée avec shadow map :

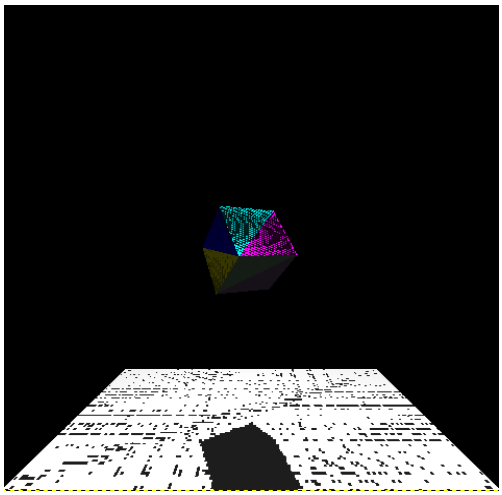
Pour compléter l'ajout de lumières à nos objets 3D, il faut ajouter des ombres projetées à notre scène. Pour cela il faut utiliser le shadow mapping (https://www.opengl-tutorial.org/fr/intermediate-tutorials/tutorial-16-shadow-mapping/). Cette méthode consiste tout d'abord mettre une caméra à la place de la lumière pour savoir ce qu'elle voit de la scène. Puis à récupérer les différents z_buffers (les profondeurs) de tous les points que la lumière "voit". Pour cela il faut utiliser un code similaire à celui déjà créé (toute la procédure de rastérisation des triangle et de z-buffer) mais cette fois ci il n'est pas utile de faire les calculs pour les couleurs et les lumières, ce qui va créer l'image de la lumière (appellation arbitraire). (le code est celui dans le fichier raster_ombre2.py).

Rendu de l'image de la lumière pour un cube avec un sol (source de lumière au dessus de la scène)

Il faut ensuite, pour chaque point dont on fait le calcul des couleur, retrouver sa projection sur l'image de la lumière et comparer le z_buffer de ce point et celui qui est déjà marqué sur l'image de la lumière (à l'endroit de la projection). Et si le z_buffer de ce point est supérieur ou égale au z_buffer de l'image de la lumière, alors ce point est vu par la lumière et le calcul de la luminosité se fait normalement, et sinon seule la lumière ambiante compte dans le calcul de la luminosité.

Acné d'ombre :

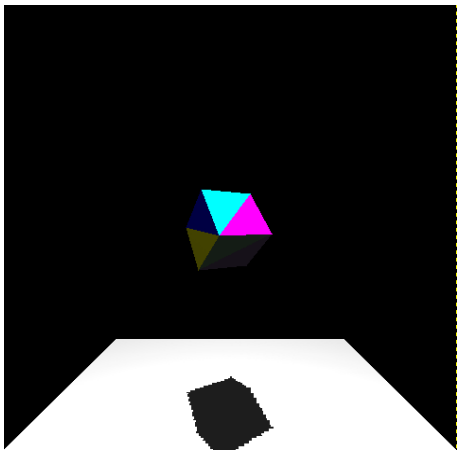
L'un des problèmes qui survient avec le shadow mapping est l'acné d'ombre. Ce phénomène est facilement visible et décrit par l'image.



Exemple d'acné d'ombre

Ce phénomène survient notamment lorsque la lumière atteint une surface plate en biais, donc les vecteurs qui vont de la source de lumière ne sont pas perpendiculaires à la surface plate. Cet événement s'explique car deux points proches sur la surface de l'objet sont tous les deux projetés sur le même pixel sur l'image de la lumière. Donc la caméra de la lumière voit pour ce pixel le point le plus proche comme étant dans la lumière, et celui qui est légèrement plus éloigné comme étant dans l'ombre. Ce qui va créer ces traits sombres sur la surface de l'objet quand le point de vue repasse sur la caméra "normale" (celle que le spectateur voit).

Pour résoudre ce problème, il faut mettre une marge pour aussi considéré dans la lumière les points qui ont un z-buffer légèrement supérieur à celles qu'il y a sur l'image de la lumière. La marge choisi ici est de 0,01. Donc à présent le comparatif des z-buffers se fait tel que le point est considéré comme étant dans la lumière si le z-buffer du point + 0,01 est supérieur ou égale au z-buffer du pixel sur l'image de la lumière.



Rendu final du cube avec un sol pour l'ombre projetée

III/ La texture :

Pour rendre ce modèle encore plus réaliste, il faut ajouter une texture à nos triangles. Le principe est de "plaquer" la texture sur chaque triangle qui compose la scène. Donc de faire correspondre chaque point du triangle avec un pixel de l'image de la texture et de lui associer sa couleur. Pour appliquer cette texture, il faut utiliser 3 pixels de la texture que l'on associe aux 3 sommets de chaque triangle. Puis il faut leurs appliquer les coordonnées barycentriques du point recherché pour retrouver la position du pixel correspondant sur la texture. Et enfin récupérer la couleur de ce pixel pour l'associer au point.
(La description des coordonnées barycentriques est sur le lien suivant : https://www.lama.univ-savoie.fr/mediawiki/index.php/Algorithme_de_rendu_de_sc%C3%A8ne_3D_par_Z-buffer)

La formule :

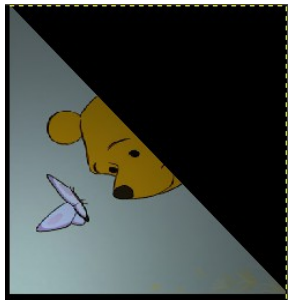
La formule est donc la suivante :

- Les 3 points sur la texture : A(a, b) , B(c, d) , C(e, f).

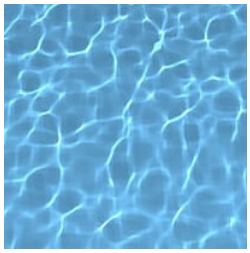
Les coordonnées choisis ici sont : A(0, 0) , B(largeur_texture, 0) , C(0, hauteur_texture), mais n'importe quel combinaisons de point sur la texture peut fonctionner.

- L'association avec les coefficients barycentriques (cfA, cfB, cfC) :
  x = cfA*a + cfB*c + cfC*e
  y = cfA*b + cfB*d + cfC*f
- On récupère la couleur avec "texture[x][y]" et on l'applique dans le calcul des couleurs.

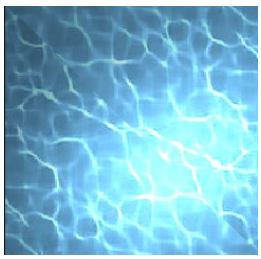
Exemple 1 de texture possible



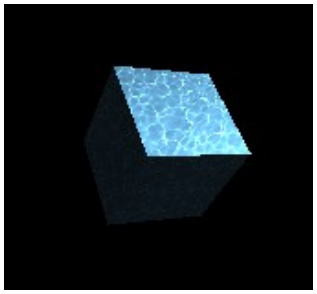
Application de cette texture sur un triangle avec le travail de luminosité



Exemple 2 de texture possible

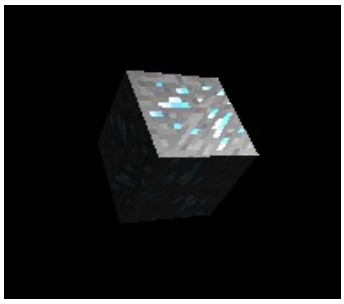


Application de cette texture sur un carré avec le travail de luminosité



Application de cette texture sur un cube avec la luminosité

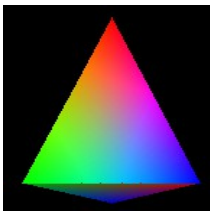
Exemple 3 de texture possible



Application de cette texture sur un cube avec la luminosité

IV/ La sphère :

Pour représenter au mieux des objets réalistes, il faut inclure des objets ronds, comme une sphère. Pour générer une sphère avec des triangles, il faut tout d'abord créer une pyramide composée uniquement de triangles équilatéraux, et dont tous les sommets sont a une distance de 1 du centre de la pyramide (on utilise une distance de 1 pour avoir un vecteur qui va du centre aux sommets qui est de 1, donc un vecteur normalisé). Puis de prendre chaque triangle de cette pyramide et de les diviser en 4 triangles équilatéraux (cf.schéma).



Pyramide de base pour la sphère

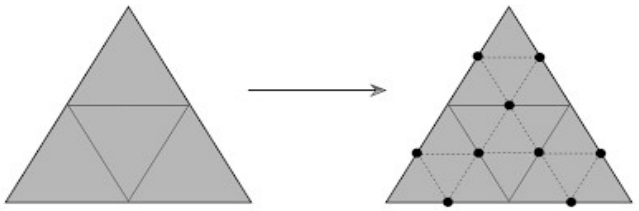


Schéma explicatif des divisions de triangle

Puis d'éloigner tous les points crées par ces nouveaux triangles de 1 par rapport au centre de la sphère, pour avoir le même éloignement entre le centre et tous les points qui composent la sphère. Puis de répéter l'opération sur chaque triangle plusieurs fois pour obtenir une sphère (cf.schéma).

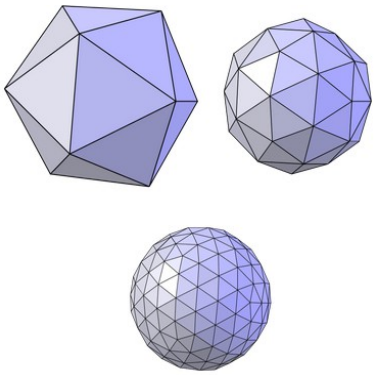


Schéma explicatif de création d'une sphère

1) Division des triangles :

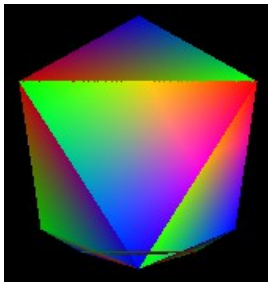
Pour diviser chaque triangles en 4 triangles équilatéraux, il faut simplement prendre l'un des coté du triangle et de trouver le milieu de ce coté pour trouver un nouveau point, puis de faire la même chose pour tous les cotés du triangle. Le point du milieu des cotés se retrouve de la manière suivante :

- Les 2 points du coté du triangle : A(xa,ya,za), B(xb,yb,zb)
- Le milieu est donc : ((xa+xb)/2, (ya+yb)/2, (za+zb)/2)

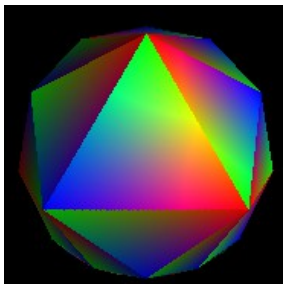
2) Éloigner les points :

Ensuite pour mettre tous les points à 1 du centre de la sphère, il faut simplement prendre le vecteur qui va du centre à ce point et de le normaliser (ce qui va donc lui donner une norme de 1). Puis il faut décaler le point du centre par ce vecteur normalisé.
Nous avons donc la formule suivante :

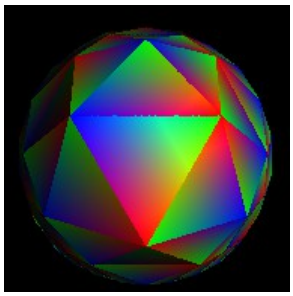
- Le centre : C = (a,b,c)
- Le vecteur qui va du centre au point : V = (d,e,f)
- Les coordonnées du point décalé sont donc : P = (a+d, b+e, c+f)



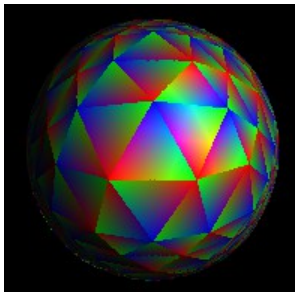
Sphère : 1 division de triangle



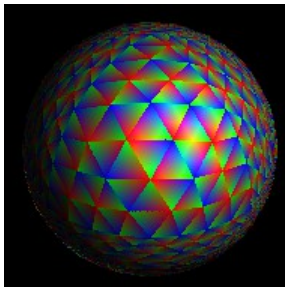
Sphère : 2 divisions de triangle



Sphère : 3 divisions de triangle



Sphère : 4 divisions de triangle



Sphère : 5 divisions de triangle

3) Les normales aux sommets :

Pour rendre la sphère plus réaliste, il faut la lisser. Et pour cela il faut mettre les normales de la sphère sur les sommets de chaque triangles, au lieu de les placer sur chaque faces de chaque triangles comme c'était le cas précédemment. Pour cela il faut remarquer que les vecteurs qui relient le centre de la sphère et ses différents sommets sont tous de norme 1, ce sont donc des vecteurs normalisés. Il faut donc utiliser ces différents vecteurs et leur appliquer une translation pour les appliquer à chaque sommets de chaque triangles qui composent la sphère, ce qui crée une norme sur chaque point qui compose la sphère. Puis utiliser les coordonnées barycentriques pour créer une normale pour chaque point qui compose les triangles. Et enfin utiliser ces normales pour le calcul des couleurs et des lumières de chaque point de la sphère ce qui va la lisser.

Pour calculer la normale pour chaque point, la formule est la suivante :

- Les 3 normales des sommets du triangle dont le point appartient :

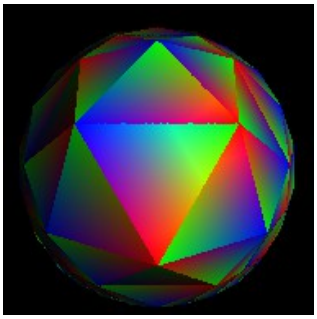
normA(a, b, c) , normB(d, e, f) , normC(g, h, i).
- L'association avec les coefficients barycentriques (cfA, cfB, cfC) :

$x = cfA*a + cfB*d + cfC*g$

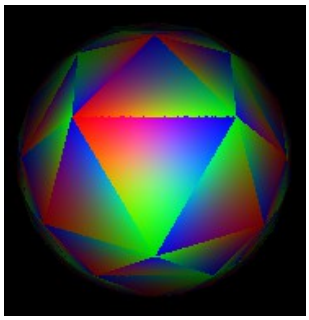
$y = cfA*b + cfB*e + cfC*h$

$z = cfA*c + cfB*f + cfC*i$

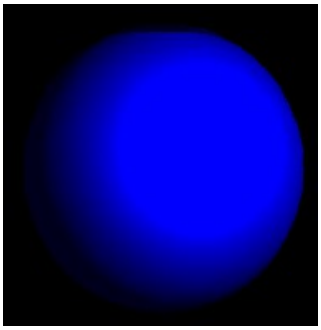
- La normale du point est donc : $N = (x, y, z)$



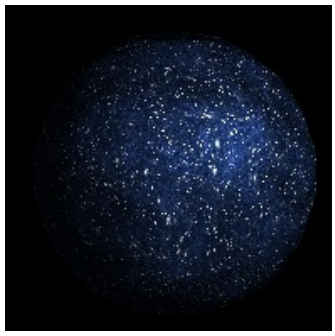
Sphère non lissé, 3 divisions de triangle



Sphère lissée



Sphère lissée monochrome



Sphère lissée avec une texture

Annexe :

1) Algorithmes finaux :

- raster.fixed.modif_2.py (algorithme principal)
- raster_ombre2.py (algorithme secondaire pour créer l'image de la lumière pour les ombres projetées)

2) Bibliothèques utilisées :

- numpy (https://numpy.org/)
- math (https://docs.python.org/fr/3.5/library/math.html)
- copy (https://docs.python.org/fr/3/library/copy.html)
- matplotlib (https://matplotlib.org/)

3) Sources :

- Travail précédent (https://www.lama.univ-savoie.fr/mediawiki/index.php/Algorithme_de_rendu_de_sc%C3%A8ne_3D_par_Z-buffer)
- Modèle d'illumination de Phong (https://fr.wikipedia.org/wiki/Ombrage_de_Phong & http://rodolphe-vaillant.fr/images/courses/phong_shading.pdf)
- Approximation de Blinn-Phong (https://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_reflection_model)
- Shadow mapping (https://www.opengl-tutorial.org/fr/intermediate-tutorials/tutorial-16-shadow-mapping/)