Université de Bordeaux 1 - LP DAWIN

MATHS POUR L'IMAGE : ALGÈBRE LINÉAIRE ET GÉOMÉTRIE

Projet de programmation : Rendu par lancer de rayon : CORRECTION

fourer@labri.fr - Novembre 2011

1 Utilisation du code existant et compréhension de son fonctionnement

La bibliothèque Qt permet de réaliser des projets multiplateformes à partir d'un code unique en C++. L'ensemble du code source est réparti dans 4 dossiers :
- **src** : contient les fichiers source en C++ `.cpp`
- **include** : contient les fichiers d'entête `.h`
- **lib** : bibliothèques externes pouvant être précompilées ou compilées en même temps que le projet (pour une meilleure compatibilité multiplateforme)
- **bin** : contient l'éxécutable créé par l'éxécution de `qmake & make`.

L'architecture du projet dans sa version initiale contient 4 types (classes) différents :
- Scène (`Scene.h`) : définit une scène virtuelle avec des lumières, une caméra et des objets géométriques,
- Objets géométriques (`Gobject.h`) : cone, cube, cylindre, plan, quadrilatère, sphère, triangle,
- Lumières (`Light.h`) : éclairage de type : ponctuel (`PLight`) ou de zone (`ALight`), un éclairage de zone correspond à un ensemble de lumières ponctuelles positionnées sur une surface (quad par exemple),
- Matériaux (`Material.h`) : définit les matériaux appliqués aux objets géométriques (contient les caractéristiques : couleur diffuse/spéculaire, exposant spéculaire, coefficient diffus/spéculaire, ...).

Les autres fichiers contiennent les structures élémentaires utilisées pour calculer le rendu de la scène :

- Rayon (`Ray.h`) : défini par une origine `Vec3f ori()` et une direction `Vec3f ori()`.
- Caméra (`Structure.h`) : défini par sa position, une direction et un vecteur "up" (pour fixer l'angle de rotation autour de la direction).
- Couleur (`Structure.h`) : vecteur de type (R,G,B).

La bibliothèque "math" (/lib/math) fournit quelques fonctions de manipulation des vecteurs et des matrices :
- `Vec-X-T` : vecteur de dimension X, de type T (e.g. Vec3f, vecteur de taille 3 réels de type float).

- `Mat-X-T` : matrice carrée de dimension X de type T (e.g. Mat3f, matrice de taille 3 réels de type float)

et la classe `Conv` contient les fonctions de conversion de base (pour les changements de repère).

2 Calculs d'intersection rayon-objet et calcul des normales

2.1 Sphère

```
float Sphere :: hit(Ray r)
{
 //méthode algébrique: lente mais efficace ...
 float t1 = -1;
 float t2 = -1;

 //on résoud l'equation x^2+y^2+z^2 = r^2+ epsilon
 // Le EPSILON est une constante très petite non obligatoire, mais qui permet de
     réduire les effets de bord
```

```
 float a = 1;
 float b = 2 * (r.dir().x() * ( r.ori().x() - Px ) + r.dir().y() * ( r.ori().y() -
     Py )
  + r.dir().z() * ( r.ori().z() - Pz ));
 float c =  pow(r.ori().x() - Px, 2) + pow(r.ori().y() - Py, 2) + pow(r.ori().z() -
     Pz, 2) - pow(radius,2) + EPSILON;
 float delta = b * b - 4 * a * c;  //discriminant (cf. cours de Lycée)

 if( delta > 0) //Il existe au moins une solution réelle
 {
  float rd =  sqrt(delta);

  t1 = (-b - rd) / 2;
  t2 = (-b + rd) / 2;

  if( t1 > SEPSILON && t1 < t2)
   return t1;

  if( t2 > SEPSILON && t2 < t1)
   return t2;
 }

 return NO_INTERSECTION;
}

// La normale correspond au vecteur unitaire directeur (donc de norme 1) partant du
    centre
// de la sphère jusqu'au point d'intersection.
Vec3f Sphere :: normale(Ray r)
{
 if(!r.intersection())
  return UNDEF_NORMAL;

 Vec3f normale = (r.getInterP() - Vec3f(Px, Py, Pz) ).normal();

 //on regarde si le rayon est à l'intérieur ou à l'extérieur de la sphère
 float angle = -r.dir().dot(normale);
 return angle<0?-normale:normale;
}
```

2.2 Plan

```
float Plane :: hit(Ray r)
{
 //équation : A(rox+trdx) + B(roy+trdy) + C (roz+trdz) + D = 0
 Vec3f norm(nx,ny,nz);

 if(norm.dot(r.dir()))
 {
  float t = -(nc + norm.dot(r.ori())) / norm.dot(r.dir());
  return t>EPSILON?t:NO_INTERSECTION;
 }
 return NO_INTERSECTION;
}

//la normale est donnée par l'équation du plan
Vec3f Plane :: normale(Ray r)
{
 Vec3f normale = Vec3f(nx,ny,nz).normal();
 float angle = -r.dir().dot(normale);
 //on vérifie si on est au dessus ou en dessous du plan
```

```
 return angle<0?-normale:normale;
}
```

2.3 Cube

```
//Un cube est composé de 6 quadrilatères
// il suffit de tester si l'un d'entre eux intersecte le rayon
float Quad :: hit(Ray r)
{
 if(!is_valid())
  return NO_INTERSECTION;

 if(!is_ready())
  return NO_INTERSECTION;

 //si rayon non perpendiculaire avec la normale du quad
 if(w.dot(r.dir()))
 {
  //on calcule le rayon dans la base du quad (changement de repere)
  Vec3f rop = Conv().change_repere_inv(r.ori(), o, chg_inv);
  Vec3f rdp = Conv().change_repere_inv(r.dir(), Vec3f(0,0,0), chg_inv);

  //intersection quand z == 0 et 0 <= x <= Xmax et y <= y <= Ymax

  //on résoud : rop.z() + rdp.z * t == 0
  float t = - rop.z() / rdp.z();
  float xi = rop.x() + rdp.x() * t;
  float yi = rop.y() + rdp.y() * t;

  //on vérifie si x and y sont dans les limites
  if( t > EPSILON && xi >= EPSILON && xi <= xmax+EPSILON && yi >= EPSILON && yi <=
      ymax+EPSILON )
   return t;
 }
 return NO_INTERSECTION;
}

//donné par la base du quad
Vec3f Quad :: normale(Ray r)
{
 if(!is_valid())
  return UNDEF_NORMAL;

 if(!is_ready())
  return UNDEF_NORMAL;

 Vec3f normale = w;

 //test de la direction
 float angle = -r.dir().dot(normale);
 return angle < 0?-normale:normale;
}
```

3 Implémentation de la BRDF de Phong

```
 for(unsigned int i=0 ; i<lum_list.size();++i){
   Light* l = lum_list[i];
   //vecteur L normalisé
   Vec3f L = (l->get_coords() - r.getInterP()).normal();
   //on créé le rayon lumineux
```

```
    Ray Rlight = Ray(P.x(), P.y(), P.z(),
                     L.x(), L.y(), L.z(), 1);

    if( check_hit(Rlight) != true )    //on vérifie si il n y a pas d'objet entre la
        lumière et le point d'intersection (sinon ombre)
    {
      //vecteur vers la camera (direction opposée de celle du rayon primaire initial)
      Vec3f V = (-r.dir()).normal();
      //normale au point d'intersection
      Vec3f N = (o->normale(r)).normal();
      //rayon réfléchi
      float s = L.dot(N);
      Vec3f R =  (N * 2.0f * s - L).normal();

      //Coefficient et couleur diffuse et spéculaire donnés par le matériau
      float Id = m->Dalbedo * s * l->get_radiance();
      float Is = m->Salbedo * m->Salbedo * pow( R.dot(V), m->exponent) *
          l->get_radiance();

      // On ajoute la contribution lumineuse au pixel calculé
      c += m->get_difColor() * Id + m->get_specColor() * Is;
    }
  }//for: on somme toutes les lumières
  c.check_correct_value();  //on vérifie que la couleur ne dépasse pas
      (R_\text{max},G_{\text{max}},B_{\text{max}}).
```

4 Transformations géométriques

Pour réaliser cette question, il faut lire et comprendre la fonction `Scene::parse_xml()` et la modifier de manière à interpréter les balises suivantes (par exemple) :

- `<rotation axis="1" angle="45" objectid="3" />` : Rotation d'un angle de 45 degrés autour de l'axe 1 appliqué à l'objet '3'

- `<translation axis="2" value="10" objectid="22"  />` : Translation de 10 unités dans l'axe 2 appliqué à l'objet 22

- `<homothetie factor="0.4" objectid="11"  />` : Homothétie de facteur 0.4 appliqué à l'objet 11 .

Il faut ensuite ajouter un champ `id` pour chaque objet et mettre à jour la classe `Gobject.h`. Pour appliquer une transformation, il suffit de calculer la matrice correspondante (c.f. cours) puis de l'appliquer aux coordonnées de l'objet :

- pour la sphère la rotation n'a pas d'effet puisqu'elle n'est pas orientée, l'homothétie s'applique à son rayon (radius) et la translation aux coordonnées du centre.
- pour un plan, la rotation s'applique à la normale du plan donnée par son équation : (ici $\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$), l'homothétie et la translation n'ont pas d'effet car un plan est infini et son origine (dans cette application) est fixe : $(0, 0, 0)$.
- pour un cube, n'importe quelle transformation s'applique à chacun de ses sommets par un produit matriciel.

5 Résultats (avec bonus) :

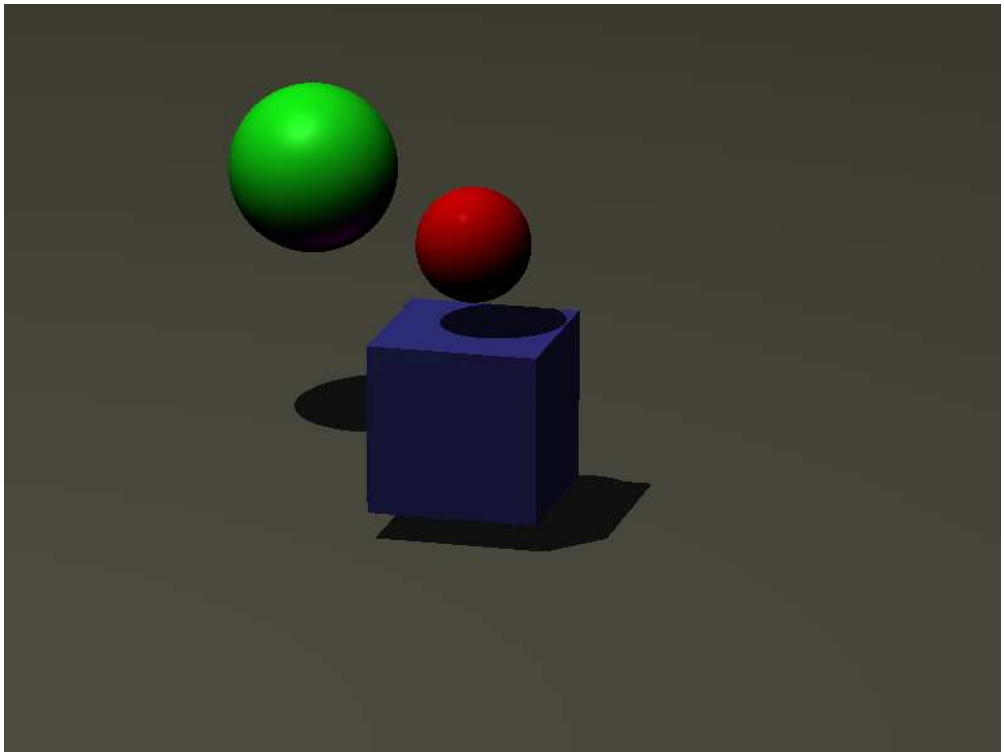

FIG. 1 – Les fichiers scene.xml et camera.xml fournis donnent cela...

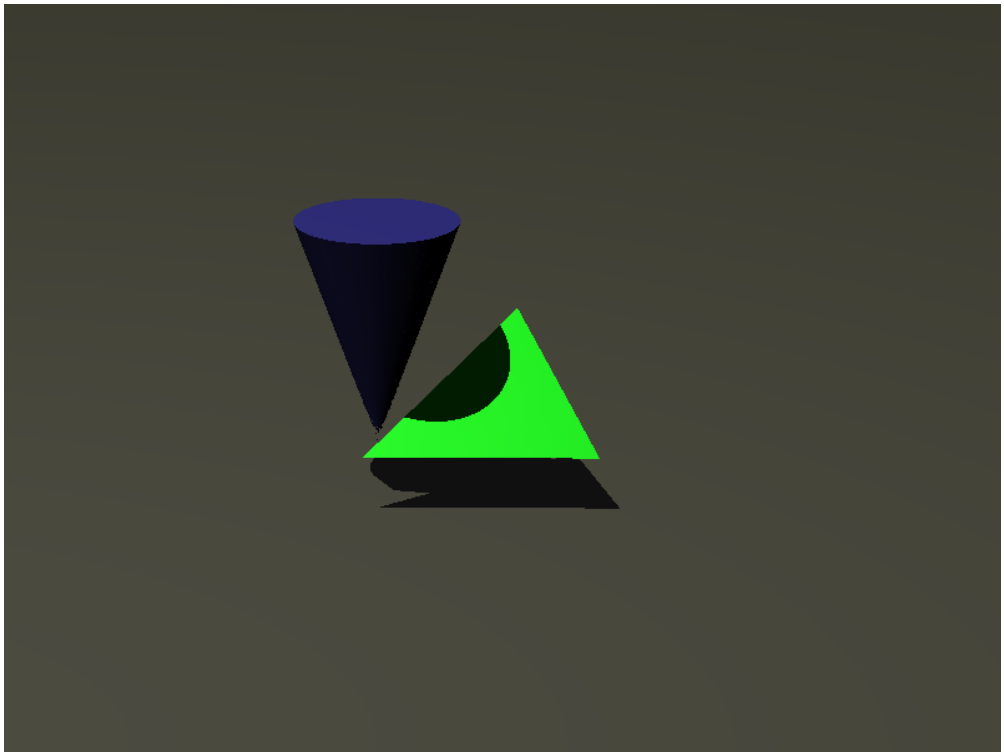

FIG. 2 – Et voici la partie bonus.