## Introduction :

**Comme promis la dernière fois, nous allons aujourd'hui faire en sorte d'avoir des rendus plus vraisemblables, en mettant en œuvre les moyens offerts par OpenGL pour simuler un éclairage réaliste. Comme d'habitude, nous commencerons par quelques éléments théoriques avant de nous appuyer sur un exemple concret pour aborder la pratique.**

**Figure 1**

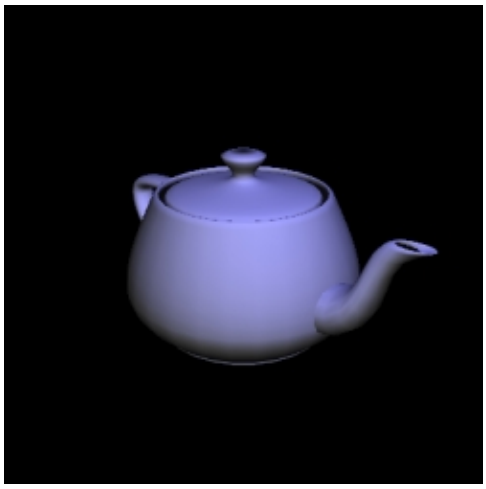

**Figure 2**

**Figure 3**

**Figure 4**

## Eclairage et matériaux réalistes :

Dans tous les programmes que nous avons écrits jusqu'à présent, nous nous sommes contentés d'affecter à chacun des sommets de nos polygones une couleur fixe. Aucun algorithme de gestion de l'illumination n'a été mis en œuvre. Un sommet dont la couleur est définie comme rouge apparaît toujours à l'écran avec le même rouge, quelque soit le point de vue depuis lequel on l'observe. Dans la réalité, un point de couleur rouge apparaîtra différemment suivant l'éclairage auquel il est soumis et l'angle sous lequel vous l'observez. OpenGL permet d'effectuer un rendu prenant en compte l'illumination des objets. Pour simuler de manière réaliste une scène tridimensionnelle il est nécessaire de reproduire les lois physiques qui régissent la lumière. Ces lois sont complexes, et il est impossible de les reproduire exactement. On se contente donc de choisir un modèle mathématique aussi proche que possible de la réalité. En fonction de l'application à laquelle il est destiné, le modèle d'illumination choisi permettra d'obtenir des résultats plus ou moins réalistes et rapides. Le modèle utilisé par le moteur de rendu de Blender donne des résultats beaucoup plus réalistes que celui d'OpenGL. En revanche, le rendu d'une scène OpenGL est bien plus rapide.

Pour étudier le modèle d'illumination utilisé par OpenGL, il nous faut aborder 3 points :

− **Les sources lumineuses** : quels sont les paramètres permettant de définir une source de lumière ?

− **Les matériaux** : Une brique ne réfléchit pas la lumière de la même manière qu'une plaque d'aluminium : le matériau est un facteur primordial pour l'éclairage d'une scène.

− **L'algorithme de remplissage** : calculer l'éclairage en chaque pixel de l'image est trop coûteux en temps. OpenGL utilise une astuce pour accélérer le rendu.

## Sources lumineuses :

Avec OpenGL, vous pouvez créer une scène contenant jusqu'à huit sources de lumière. Ces sources peuvent être de type 'omnidirectionnel' ( la source émet de la lumière uniformément dans toutes les directions, comme le soleil), ou 'spot' (la lumière n'est émise que dans un cône).

Une source de lumière est caractérisée par 10 paramètres qu'on peut classer en 4 catégories :

**1 − Les paramètres de lumière**
La lumière émise par une source est formée de trois composantes : la plus importante, la composante diffuse, est réfléchie par un objet dans toutes les directions. La composante spéculaire correspond à la lumière qui est réfléchie dans une direction privilégiée (et qui est donc à l'origine de l'effet de brillance). La composante ambiante est la plus difficile à appréhender. La lumière ambiante d'une scène est une lumière non directionnelle, que l'on peut considérer comme issue des multiples réflexions de rayons lumineux. OpenGL vous permet d'affecter une lumière ambiante pour toute la scène. La composante ambiante d'une source ajoute une contribution à cette lumière ambiante globale.
Vous avez la possibilité d'affecter une couleur à chacune des trois composantes. En affectant un bleu clair à la composante diffuse d'une source, vous donnez une teinte globale à la lumière qui en est issue. En affectant un rouge à la composante spéculaire, vous modifiez l'apparence des reflets de cette source sur les objets de la scène.

Les paramètres de lumière diffuse, spéculaire et ambiante sont difficiles à appréhender pour le novice. Comme rien ne vaut la pratique, la meilleure chose à faire est d'expérimenter. Je vous conseille de modifier le programme exemple de façon à pouvoir faire varier les paramètres de lumière d'une des sources par l'intermédiaire du clavier et observez l'influence de chacune des composantes sur le rendu de la scène.

**2 − Le paramètre de position**
Comme son nom l'indique, il permet de définir la position de la source de lumière dans la scène.

**3 − Les paramètres de spot**
L'angle de coupure, illustré sur la figure 1, permet de définir le type de source que vous souhaitez utiliser : si l'angle vaut 180°, vous utilisez une source de lumière omnidirectionnelle. Si l'angle est compris entre 0 et 90°, vous définissez un spot, et vous pouvez alors modifier les deux autres paramètres de spot : la direction dans laquelle il pointe et

l'exposant du spot. Ce dernier paramètre permet de faire varier la concentration de la lumière à l'intérieur du cône définissant le spot. Si l'exposant vaut 0, la lumière est répartie également dans toutes les directions définies par le cône. Plus vous faites croître la valeur de l'exposant, plus la lumière sera concentrée autour de l'axe donné par le paramètre de direction.

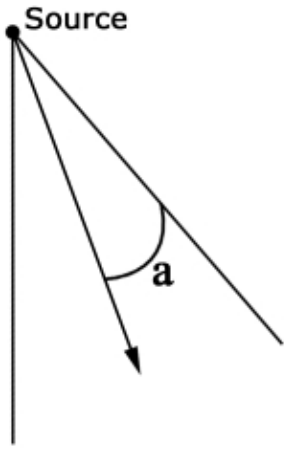


**Figure 5 : Le paramètre angle de coupure 'a' d'un spot**

**4 − Les paramètres d'atténuation**
Ils permettent de prendre en compte le phénomène physique suivant : plus un objet est éloigné d'une source de lumière, moins il est éclairé par cette dernière. Les paramètres d'atténuation sont au nombre de 3 : le facteur d'atténuation constante, le facteur d'atténuation linéaire et le facteur d'atténuation quadratique. Si on note respectivement Ac, Al et Aq ces trois coefficients, l'intensité reçue par un point P situé à une distance d de la source lumineuse est divisée par Ac+Al*d+Aq*d*d. Par défaut, Ac=1 et Al=Aq=0, ce qui correspond à une atténuation nulle (division par 1).

## Matériau :

La spécification d'un matériau pour un objet se fait par l'intermédiaire de 5 paramètres :
− La couleur diffuse
− La couleur spéculaire
− La couleur ambiante
− La couleur émise
− Le coefficient de brillance

Dans la réalité, un rayon lumineux est constitué d'un ensemble d'ondes de longueurs différentes. A chaque longueur d'onde correspond une couleur (visible ou non). Un objet frappé par un rayon lumineux va absorber certaines longueurs d'onde et réfléchir les autres. Si l'herbe est verte lorsqu'elle est éclairée par la lumière du soleil, c'est parce qu'elle ne réfléchit que les ondes dont la couleur est verte.

En informatique, une couleur est représentée par un triplet de composantes rouge, verte, et bleue. Par synthèse additive, on arrive à recréer à partir de ces trois composantes la plupart des couleurs visibles. Pour définir un le comportement d'un objet vis−à−vis d'une couleur RVB, il suffit de dire quel pourcentage de chaque composante est réfléchi. Ainsi, si on affecte à un matériau les pourcentages (R=1, V=0.5, B=0), soit (R=100%, V=50%, B=0%), et si on l'éclaire avec une lumière blanche (R=1,V=1,B=1), le matériau va réfléchir un rayon de couleur (R=1,V=0.5,B=0), et votre objet vous paraîtra orange. Cet exemple montre bien la double signification physique du paramètre : on peut l'interpréter comme le pourcentage réfléchi de chaque composante de couleur, ou bien comme la couleur du rayon réfléchi lorsque l'objet est éclairé par une source de lumière blanche. Vous remarquerez que si vous éclairez cet objet avec une lumière bleue (R=0,V=0,B=1), le matériau va renvoyer une rayon de couleur noire (R=0,V=0,B=0) qui correspond à une absence de lumière.

Puisque nous avons décomposé la lumière émise par une source en lumière diffuse, spéculaire et ambiante, on peut spécifier le comportement du matériau pour chacune de ces composantes, ce qui explique les trois premiers paramètres du matériau : couleur diffuse, couleur spéculaire et couleur ambiante.

Le paramètre de couleur émise correspond à une composante de lumière supplémentaire, qui permet de prendre en compte le fait que certains objets peuvent émettre eux−même de la lumière. Si vous modélisez une ampoule allumée, vous pourrez attribuer du blanc ou du jaune à la couleur émise par le matériau de l'objet. Pour les matériaux classiques, la couleur émise est noire. Cependant, cette couleur émise n'est pas considérée comme une source de lumière pour les autres objets de la scène.

Pour expliquer le dernier paramètre, le coefficient de brillance, revenons à ce que nous avons vu concernant la lumière spéculaire : elle est à l'origine du phénomène de brillance qui crée des tâches de lumière intense sur les objets (voir figure 4 et 5). La composante spéculaire de la lumière est réfléchie dans une direction privilégiée. Dans la réalité, cette réflexion ne se fait jamais de manière parfaite, et le coefficient de brillance permet de modéliser cette imperfection. Sa signification physique est un étalement des taches spéculaires sur les objets lorsqu'on diminue la valeur du coefficient.

## L'algorithme d'éclairage :

Pour des raisons d'efficacité, OpenGL ne calcule pas la couleur de chaque pixel d'un polygone : soit il remplit chaque polygone avec un couleur unie (mode de remplissage 'Flat'), soit il utilise un algorithme de Gouraud (mode 'Smooth'), dont le principe est le suivant : pour un polygone donné, la couleur de chacun des sommets est calculée, et le polygone est rempli avec un dégradé entre ces différentes couleurs.

Pour calculer correctement la réflexion des rayon lumineux en un point, OpenGL a besoin de connaître la perpendiculaire à la surface de l'objet au point considéré. On appelle cette donnée une normale. Nous aurons l'occasion de revenir longuement sur la question des normales dans un prochain didacticiel, et dans le programme exemple, nous utiliserons une théière générée avec ses normales par glut.

## L'exemple :

Passons maintenant au programme exemple. Aujourd'hui, il affiche à l'écran une théière générée par glut et éclairée par 2 sources lumineuses différentes. Vous avez la possibilité de tourner autour de la théière avec les touches 'a' et 'z', et vous pouvez faire varier certains paramètres d'éclairage avec d'autres touches (jetez un coup d'œil à la fonction de rappel clavier() pour connaître toutes les variables modifiables).

## Paramètres d'éclairage :

L'architecture du programme est classique et les seules nouveautés concernent l'utilisation du modèle d'illumination. La phase d'initialisation de l'éclairage commence par la spécification du mode remplissage des polygones avec :

```
glShadeModel(GL_SMOOTH);
```

Ensuite on indique à OpenGL qu'on souhaite utiliser le calcul d'éclairage, en activant la variable d'état GL_LIGHTING :

```
glEnable(GL_LIGHTING);
```

Nous avons vu qu'OpenGL permet d'utiliser jusqu'à huit sources de lumière. Ces huit lampes sont indexées par les constantes GL_LIGHT0 à GL_LIGHT7. Il faut activer chacune des sources qu'on souhaite utiliser (2 dans notre cas) :

```
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
```

Vient ensuite le paramétrage des lampes. Il se fait avec une unique fonction, glLightfv(), dont le prototype est le suivant :

void glLightfv(GLenum lampe,GLenum nomparam, GLType param)

'lampe' désigne la lampe dont on veut modifier un propriété. 'nomparam' est le nom du paramètre à modifier. Il s'agit d'une des dix propriétés de source lumineuse que nous avons évoqué :
− GL_DIFFUSE
− GL_AMBIENT
− GL_SPECULAR
− GL_POSITION
− GL_SPOT_CUTOFF
− GL_SPOT_DIRECTION
− GL_SPOT_EXPONENT
− GL_CONSTANT_ATTENUATION
− GL_LINEAR_ATTENUATION
− GL_QUADRATIC_ATTENUATION

'param' désigne la valeur à affecter au paramètre choisi. Vous remarquerez que les paramètres sont passés sous forme de tableaux.
La définition de la position des sources de lumières se trouve dans la fonction d'affichage. En effet, tout comme les sommets des polygones, les paramètres de position et de direction d'une source subissent les transformations contenues dans la matrice de modélisation−visualisation. Il faut donc placer judicieusement la déclaration de ces deux paramètres. Les deux spots que nous utilisons sont omnidirectionnels (car nous ne modifions pas la valeur par défaut de GL_SPOT_CUTOFF qui vaut 180), et donc le paramètre direction ne nous est pas utile.

## Paramètres de matériaux :

Le système d'affection des propriétés de matériau utilise le principe de machine à états. OpenGL gère un matériau courant. Lorsqu'un polygone est décrit, il se voit affecter le matériau courant. La modification du matériau courant se fait avec la fonction GlMaterialfv() :

Void glMaterialfv(GLenum face, GLenum nomparam, Gltype param) ;

'face' indique la face (avant ou arrière) dont on souhaite modifier les paramètres. Nous n'avons pas encore abordé les considérations de face, et nous nous satisferons de la valeur GL_FRONT_AND_BACK. Tout comme pour glLightfv(), nomparam désigne la propriété qu'on souhaite changer, et 'param' est un tableau contenant la nouvelle valeur à affecter à 'nomparam'. Les valeur de 'nomparam' possibles sont :
−GL_AMBIENT
−GL_DIFFUSE
−GL_SPECULAR
−GL_EMISSION
−GL_SHININESS (i.e. coefficient de brillance)

## Conclusion :

C'est tout pour aujourd'hui. Nous n'avons pas abordé absolument tout ce qui concerne l'éclairage. Les fonctions manquantes seront vues au fur est à mesure de nos progrès. Comme je vous l'ai déjà dit, je vous conseille de bidouiller le programme afin de pouvoir faire varier d'autres paramètres d'éclairage, notamment ceux concernant les composantes diffuse, spéculaire et ambiante des sources de lumière. Si vous êtes flemmards, vous pouvez vous rabattre sur un modeleur 3D, créer une sphère éclairée par une lampe et faire varier les paramètres de la source lumineuse et de l'objet.

A la rentrée, nous nous intéresserons aux problèmes de placage de textures sur des objets. Nous aborderons l'utilisation d'images JPEG avec la libjpeg et la synthèse de textures procédurales. Bonnes Vacances !

## Références :

| | |
|---|---|
| **OpenGL 1.2** | Woo, Neider, Davis et Shreiner − Campus Press Référence<br>La traduction française de la dernière édition du livre de référence en matière de programmation OpenGL |
| **Eclairage et rendu numériques** | Jeremy Birn − Campus Press.<br>Orienté pratique, cet ouvrage vous apprendra à créer des rendus de<br>qualité. |
| **Introduction à l'Infographie** | Foley, Van Dam, Feiner et Hughes − Vuibert<br>La bible de l'informatique graphique. |
| **www.opengl.org** | Le site officiel d'OpenGL. Tout y est : présentation, documents de spécification, liens vers des didacticiels, bibliographie |
| **www.mesa3d.org** | Le site de Mesa, l'implémentation libre d'OpenGL la plus utilisée sous Linux |
| **reality.sgi.com/mjk/glut3** | La page de glut. Vous y trouverez le manuel de référence glut |
| **http://www.linuxgraphic.org/section3d/openGL/index.html** | La section OpenGL du site Linuxgraphic.org. Un tout nouveau forum attend vos questions. |

## Code source :

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.14159265

void affichage();
void clavier(unsigned char touche,int x,int y);
void reshape(int x,int y);
void calcTableCosSin();

int angle=45;
float Sin[360],Cos[360];
GLfloat L0pos[]={ 0.0,2.0,−1.0};
GLfloat L0dif[]={ 0.3,0.3,0.8};
GLfloat L1pos[]={ 2.0,2.0,2.0};
GLfloat L1dif[]={ 0.5,0.5,0.5};
GLfloat Mspec[]={0.5,0.5,0.5};
GLfloat Mshiny=50;

int main(int argc,char **argv)
{
  /* initialisation de glut et creation
     de la fenetre */
  glutInit(
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowPosition(200,200);
  glutInitWindowSize(400,400);
  glutCreateWindow("light1");

  /* précalcul de la table des sinus et cosinus */
```

```
  calcTableCosSin();

  /* Initialisation d'OpenGL */
  glClearColor(0.0,0.0,0.0,0.0);
  glColor3f(1.0,0.0,00);
  glEnable(GL_DEPTH_TEST);

  /* Paramétrage des lumières */

  glShadeModel(GL_SMOOTH);
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_LIGHT1);
  glLightfv(GL_LIGHT0,GL_DIFFUSE,L0dif);
  glLightfv(GL_LIGHT0,GL_SPECULAR,L0dif);
  glLightfv(GL_LIGHT1,GL_DIFFUSE,L1dif);
  glLightfv(GL_LIGHT1,GL_SPECULAR,L1dif);

  /* Paramétrage du matériau */
  glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,Mspec);
  glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,Mshiny);

  /* Mise en place de la perspective */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45.0,1.0,0.1,10.0);
  glMatrixMode(GL_MODELVIEW);

  /* Mise en place des fonctions de rappel */
  glutDisplayFunc(affichage);
  glutKeyboardFunc(clavier);
  glutReshapeFunc(reshape);

  /* Entree dans la boucle principale */
  glutMainLoop();
  return 0;
}



void affichage(){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  gluLookAt(4.5*Cos[angle],2.0,4.5*Sin[angle],0.0,0.0,0.0,0.0,1.0,0.0);
  glLightfv(GL_LIGHT0,GL_POSITION,L0pos);
  glLightfv(GL_LIGHT1,GL_POSITION,L1pos);
  glutSolidTeapot(1.0);
  glutSwapBuffers();
}



void clavier(unsigned char touche,int x,int y)
{
   switch (touche)
    {
    case 'z' :   /* increment de l'angle de position */
      angle+=2;
      if (angle>=360)
        angle-=360;
      glutPostRedisplay();
      break;
    case 'a' :  /* decrement de l'angle de position */
      angle-=2;
```

```
      if (angle<0)
        angle+=360;
      glutPostRedisplay();
      break;
    case 'w' : /* Lampe 0 on */
      glEnable(GL_LIGHT0);
      glutPostRedisplay();
      break;
    case 'x' : /* Lampe 0 off */
      glDisable(GL_LIGHT0);
      glutPostRedisplay();
      break;
    case 'c': /* lampe 1 on */
      glEnable(GL_LIGHT1);
      glutPostRedisplay();
      break;
    case 'v': /* lampe 1 off */
      glDisable(GL_LIGHT1);
      glutPostRedisplay();
      break;
    case 'm': /* increment reflexion speculaire */
      Mspec[0]+=0.1;
      if (Mspec[0]>1)
        Mspec[0]=1;
      Mspec[1]+=0.1;
     if (Mspec[1]>1)
        Mspec[1]=1;
      Mspec[2]+=0.1;
     if (Mspec[2]>1)
       Mspec[2]=1;
     glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,Mspec);
     glutPostRedisplay();
      break;
    case 'l': /* decrement reflexion speculaire */
      Mspec[0]-=0.1;
      if (Mspec[0]<0)
        Mspec[0]=0;
      Mspec[1]-=0.1;
     if (Mspec[1]<0)
        Mspec[1]=0;
      Mspec[2]-=0.1;
     if (Mspec[2]<0)
        Mspec[2]=0;
      glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,Mspec);
      glutPostRedisplay();
      break;
    case 'j': /* increment de la brillance */
      Mshiny-=1;
      if (Mshiny<0)
        Mshiny=0;
      glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,Mshiny);
      glutPostRedisplay();
      break;
    case 'k': /* decrement de la brillance */
      Mshiny+=1;
      if (Mshiny>128)
        Mshiny=128;
      glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,Mshiny);
      glutPostRedisplay();
      break;
     case 'q' :
      exit(0);
    }
}
```

```
void reshape(int x,int y)
{
  if (x<y)
    glViewport(0,(y-x)/2,x,x);
  else
    glViewport((x-y)/2,0,y,y);
}



void calcTableCosSin()
{
  /* calcul du tableau des sinus et cosinus */
  int i;
  for (i=0;i<360;i++) {
    Cos[i]=cos(((float)i)/180.0*PI);
    Sin[i]=sin(((float)i)/180.0*PI);
  }
}
```