OpenGL (Partie 2)

Visualisation en OpenGL

Processus de visualisation

Quatre transformations utilisées au cours du processus de création d'une image:

(1) Transformation d'affichage (Viewport)

Permet de fixer la taille et la position de l'image sur la fenêtre d'affichage.

(2) Transformation de projection (Projection)

Permet de fixer les caractéristiques optiques de la caméra de visualisation (type de projection, ouverture, ...).

(3) Transformation de visualisation (View)

Permet de fixer la position et l'orientation de la caméra de visualisation.

(4) Transformation de modélisation (Model)

Permet de créer la scène à afficher par création, placement et orientation des objets qui la composent.

Les transformations de visualisation et de modélisation, de même nature, n'en forment qu'une pour OpenGL (transformation GL_MODELVIEW). Cette transformation fait partie de l'environnement OpenGL.

La transformation de projection existe en tant que telle dans OpenGL, et fait elle aussi partie de l'environnement OpenGL (transformation GL_PROJECTION).

Chacune de ces deux transformations peut être modifiée indépendamment de l'autre. Cela permet d'obtenir une

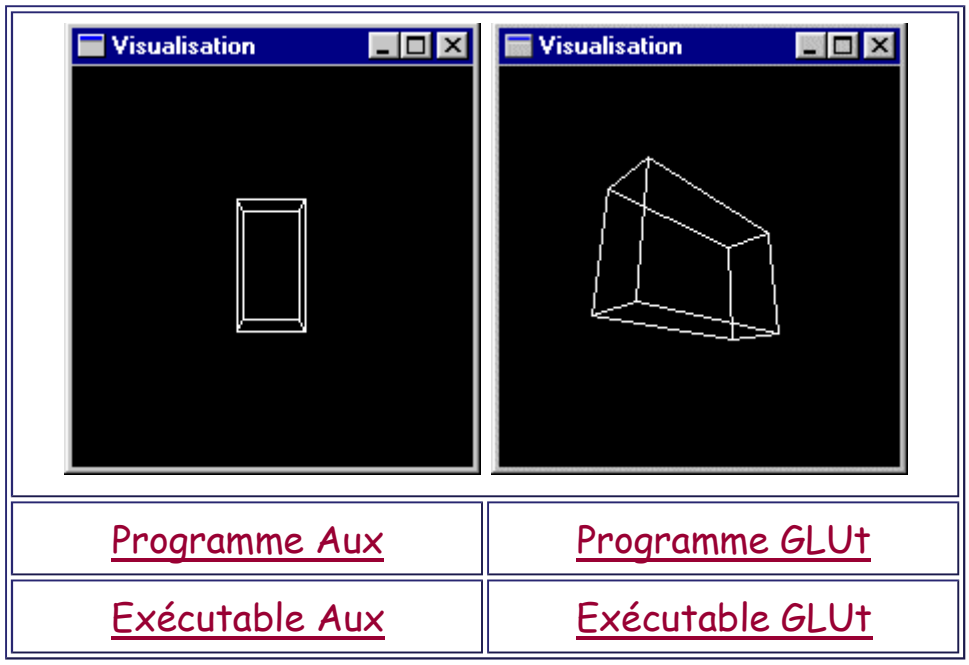

RETOUR

indépendance des scènes modélisées vis à vis des caractéristiques de la "caméra" qui les visualise.

La transformation d'affichage est elle aussi paramétrable.

Exemple

```
// w et h variables globales de type int

static float rotx = 10.0f ;
static float roty = 20.0f ;
static float rotz = 30.0f ;

glViewport(0,0,w,h);                          // 1
glMatrixMode(GL_PROJECTION);                  // 2
glLoadIdentity();                             // 2
glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);        // 2
glMatrixMode(GL_MODELVIEW);                   // 3-4
glLoadIdentity();                             // 3
glTranslatef(0.0,0.0,-5.0);                   // 3
glRotatef(rotz,0.0,0.0,1.0);                  // 3
glRotatef(rotx,1.0,0.0,0.0);                  // 3
glRotatef(roty,0.0,1.0,0.0);                  // 3
glScalef(1.0,2.0,3.0);                        // 4
glutWireCube(1.0);                            // 4
```



Choix de la transformation OpenGL de travail

- `void glMatrixMode(GLenum mode);`

  mode: Transformation sur laquelle (GL_MODELVIEW ou GL_PROJECTION) les

transformations géométriques à venir vont être composées de manière incrémentale.

Pour réaliser un affichage (construction d'une image), glMatrixMode est généralement appelé successivement une fois sur chacun des deux paramètres de manière à établir les transformations géométriques (matrices) modelview et projection. Ces appels sont habituellement réalisés au sein de la fonction reshape si les librairies Aux ou GLUt sont utilisées. Dans la fonction display de Aux ou GLUt, l'habitude veut que l'on ne travaille qu'en modelview (sauf exception, par exemple dans le cas de la programmation d'une animation de la caméra).

Transformations géométriques

Transformations d'utilité générale applicables en modelview ou en projection

- `void glLoadIdentity(void);`

  Affecte la transformation courante avec la transformation identité.

- `void glLoadMatrix{f d}(const TYPE *m);`

  Affecte la transformation courante avec la transformation caractérisée mathématiquement par la matrice m (16 valeurs en coordonnées homogènes).

- `void glMultMatrix{f d}(const TYPE *m);`

  Compose la transformation courante par la transformation de matrice m (16 valeurs en coordonnées homogènes).

- `void glTranslate{f d}(TYPE x,TYPE y,TYPE z);`

  Compose la transformation courante par la translation de vecteur (x,y,z).
  Transformation très utilisée en modélisation.

- **`void glRotate{f d}(TYPE a,TYPE dx,TYPE dy,TYPE dz);`**

  Compose la transformation courante par la rotation d'angle a degrés autour de l'axe (dx,dy,dz) passant par l'origine.
  Transformation très utilisée en modélisation.

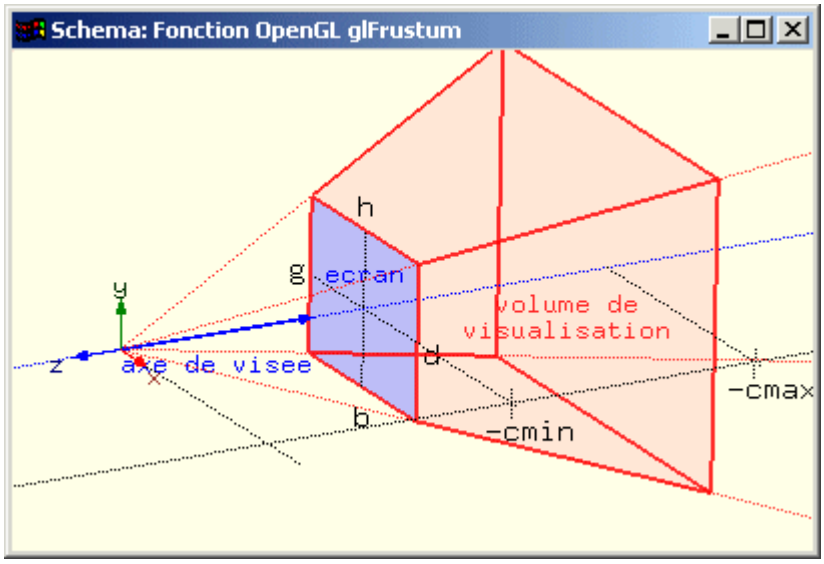
- **`void glScale{f d}(TYPE rx,TYPE ry,TYPE rz);`**

  Compose la transformation courante par la transformation composition des affinités orthogonales d'axes x, y et z, de rapports respectifs rx, ry et rz selon ces axes.
  Transformation très utilisée en modélisation.

Transformations de projection

- **`void glFrustum(GLdouble g, GLdouble d, GLdouble b, GLdouble h, GLdouble cmin, GLdouble cmax);`**

  Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée de sommet l'origine O (position de la caméra virtuelle de visualisation), orientée selon l'axe -z (orientation de la caméra virtuelle de visualisation) et de base supérieure définie par la diagonale (g,b,-cmin) (d,h,-cmin).
  cmin et cmax sont les distances entre l'origine et les plans de clipping en z "near" (-cmin en z) et "far" (-cmax en z). Etant des distances, cmin et cmax doivent avoir une valeur positive. Ces valeurs doivent aussi respecter cmin < cmax. Ces valeurs étant comptées selon l'axe -z, seuls les objets placés en z négatif peuvent être visibles.

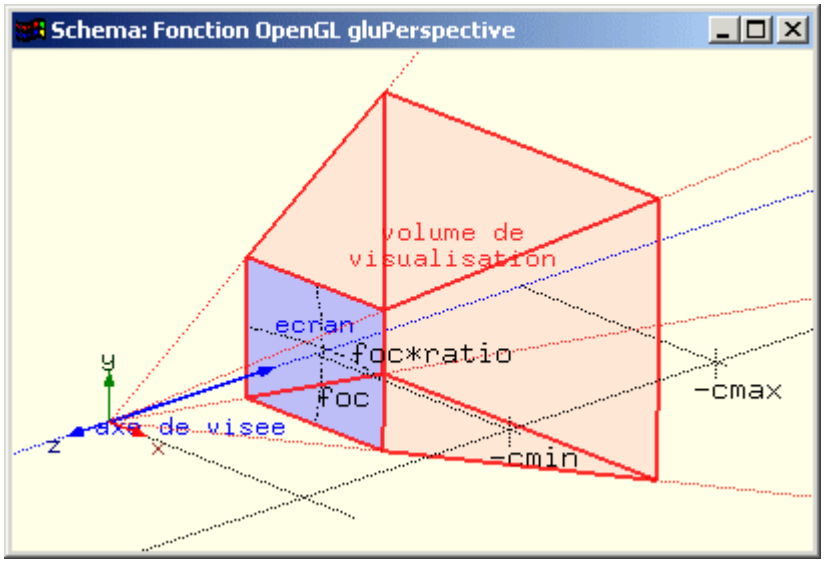


Programme GLUt

Exécutable GLUt

- **`void gluPerspective(GLdouble fov, GLdouble ratio, GLdouble cmin, GLdouble cmax);`**

  Compose la transformation courante par la transformation de projection en perspective de volume de visualisation défini par la pyramide tronquée de sommet l'origine O (position de la caméra virtuelle de visualisation), orientée selon l'axe -z (orientation de la caméra virtuelle de visualisation), possédant l'angle fov comme angle d'ouverture verticale, l'apect-ratio ratio (rapport d'ouverture largeur/hauteur) et les plans de clipping near et far en z = -cmin et z = -cmax.
  cmin et cmax doivent avoir une valeur positive et respecter cmin < cmax car il s'agit de distances à l'origine (comptées selon l'axe -z).s selon l'axe -z).



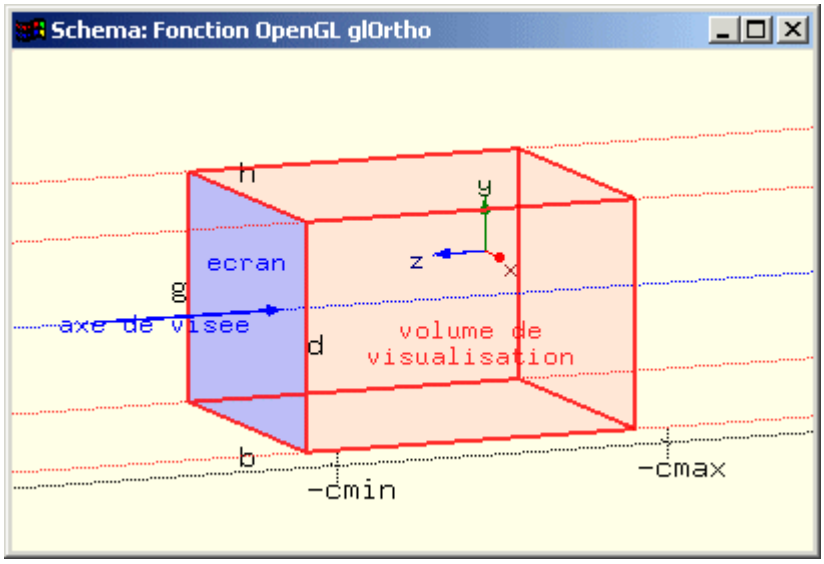
Programme GLUt

Exécutable GLUt

Exemple:
gluPerspective(40.0,1.5,50.0,100.0)
Cet appel de fonction configure une caméra de visualisation en perspective placée à l'origine (c'est obligatoire) et orientée selon l'axe -z (c'est obligatoire) avec un angle d'ouverture verticale de 40.0°, un angle d'ouverture horizontale de 40.0*1.5 = 60.0°, un plan de clipping qui élimine tous les objets ou morceaux d'objet situés en z > -50.0 et un plan de clipping qui élimine tous les objets ou morceaux d'objet situés en z < -100.0.
Ces valeurs sont considérées dans le repère courant au moment de l'appel de fonction.

- **`void glOrtho(GLdouble g, GLdouble d, GLdouble b, GLdouble h, GLdouble cmin, GLdouble cmax);`**

  Compose la transformation courante par la transformation de projection orthographique selon l'axe -z et définie par le volume de visualisation parallélépipèdique (g,d,b,h,-cmin,-cmax). cmin et cmax peuvent être positifs ou négatifs mais doivent respecter cmin < cmax.

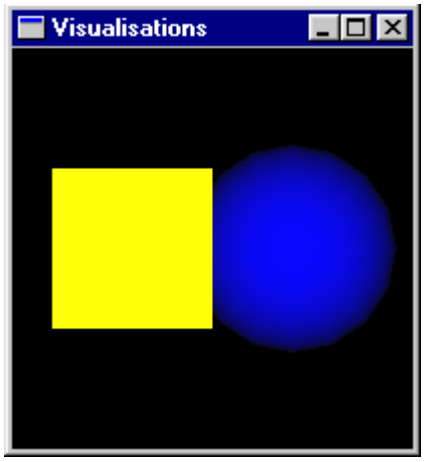


Programme GLUt

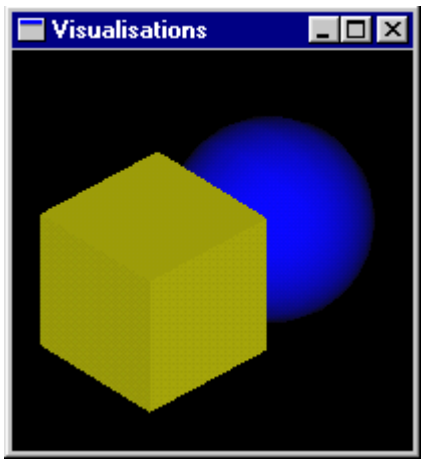
Exécutable GLUt

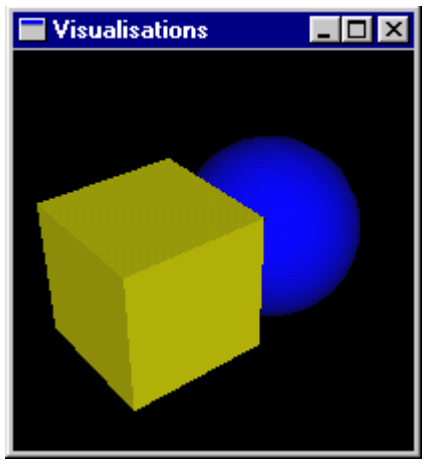
Exemples

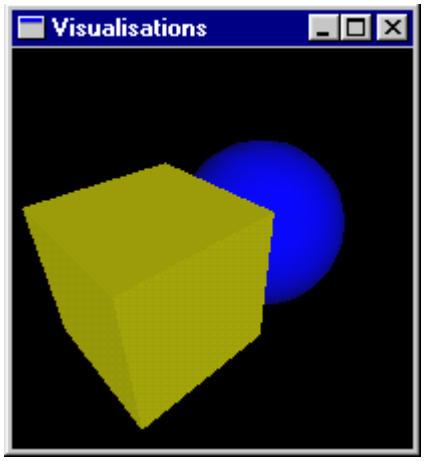
Visualisation en projection parallèle
et projection en perspective

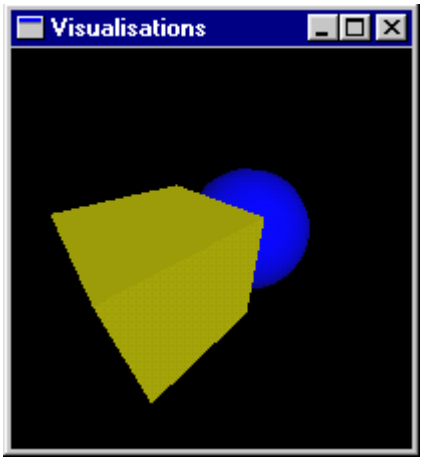


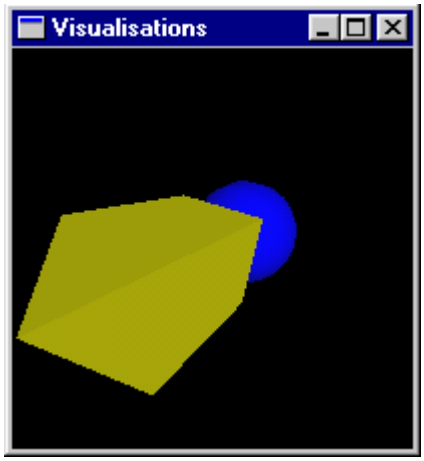
Projections orthographique et en perspective



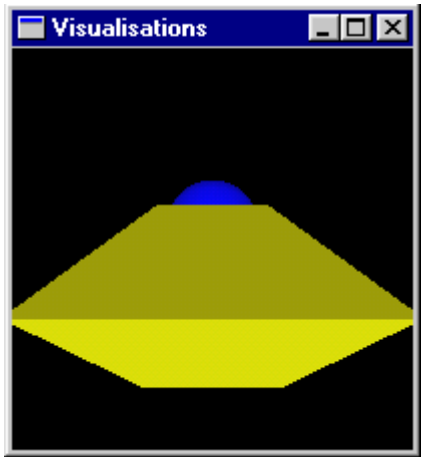


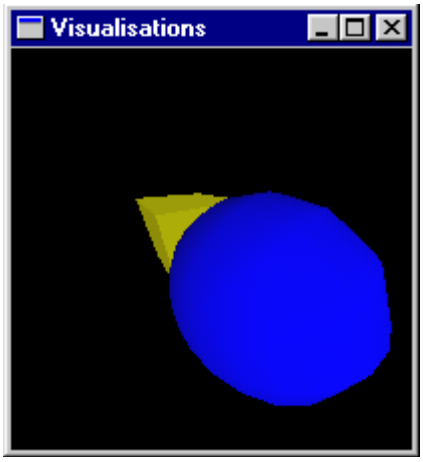






Projection en perspective avec rapprochement de la scène





Projection en perspective: très grosses déformations

Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

OpenGL est peu souple en terme de gestion des caméras:
- Elles sont obligatoirement orientées avec la direction de visualisation -z (0.0, 0.0, -1.0).

- Les caméras de projection en perspective sont obligatoirement placées à l'origine.

Il revient au développeur de placer et d'orienter la scène devant la caméra pour fournir le point de vue souhaité.
Le code est toujours structuré pour que l'ordre d'exécution soit (cf exemple tout en haut de cette page):
- Définition et paramétrage de la caméra (mode GL_PROJECTION).
- Changement de repère de placement et d'orientation de la scène devant la caméra (mode GL_MODELVIEW).
- Modélisation de la scène (mode GL_MODELVIEW).

Transformation spécifique à la visualisation (view)

- `void gluLookAt(GLdouble ex, GLdouble ey, GLdouble ez, GLdouble cx, GLdouble cy, GLdouble cz, GLdouble upx, GLdouble upy, GLdouble upz);`

  Compose la transformation courante (généralement GL_MODELVIEW) par la transformation donnant un point de vue depuis la position (ex,ey,ez) avec une direction de visualisation passant par la position (cx,cy,cz).
  (upx,upy,upz) indique la direction du repère courant (fréquemment le repère global) qui devient la direction y (0.0, 1.0, 0.0) dans le repère écran.

  gluLookAt est une fonction de la bibliothèque GLU qui permet de faciliter la résolution du problème du placement d'une caméra ailleurs qu'à l'origine.

  Exemple:
  gluLookAt(10.0,15.0,10.0,3.0,5.0,-2.0,0.0,1.0,0.0)
  place la caméra en position (10.0,15.0,10.0),
  l'oriente pour qu'elle vise le point de position (3.0,5.0,-2.0) et
  visualise la direction (0.0,1.0,0.0) de telle manière qu'elle apparaît verticale dans la fenêtre de visualisation.

Ces valeurs sont considérées dans le repère courant (fréquemment le repère global).

Dans la pratique, gluLookAt place et oriente la scène devant la caméra de telle manière que la caméra semble placée et orientée selon les paramètres d'entête.
-> L'appel à gluLookAt doit être exécuté en mode GL_MODELVIEW (donc après la définition de la caméra en mode GL_PROJECTION) et doit être placé dans le code avant la création de la scène pour porter sur elle.

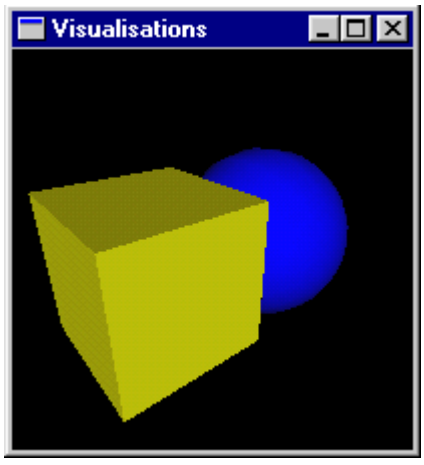
Transformation d'affichage

- `void glViewport(GLint x,GLint y,GLsizei l,GLsizei h);`

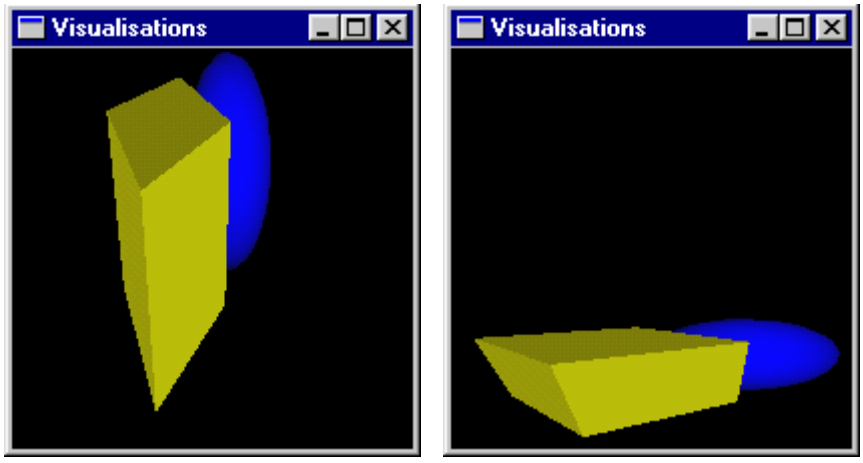
x, y: Position du coin supérieur gauche (en pixels)
l,h: Résolution (en pixels)

Définit le rectangle de pixels de la fenêtre d'affichage dans lequel l'image calculée sera affichée.

Changement de viewport



Utilisation de toute la surface de la fenêtre d'affichage



Utilisation d'une partie seulement
de la fenêtre d'affichage

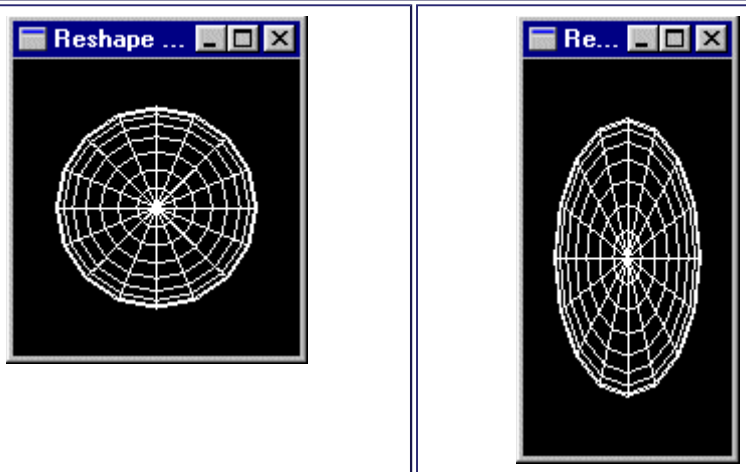
Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

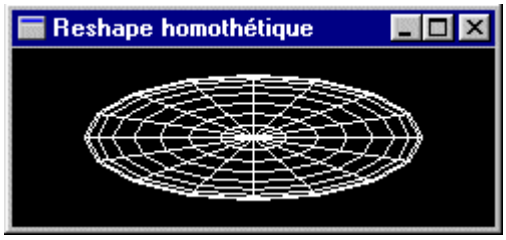
Problème

L'affichage des scènes fait appel à un repère virtuel associé au viewport de visualisation. Les dimensions fournies au moment de l'exécution de la fonction de projection (GL_PROJECTION) définissent ce repère qui peut ne pas être homothétique au viewport.

Exemple: `glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0)` associe le repère virtuel de diagonale (-1.0,-1.0) - (1.0,1.0) au viewport d'affichage. Si celui-ci n'a pas des résolutions en x et y identiques, les objets dessinés sont déformés.

Adaptation de la transformation de projection
au viewport d'affichage





Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Solution: Adapter aux dimensions du viewport les dimensions données pour la définition de la projection.

Exemple: Pour un viewport défini par
`glViewport(0,0,w,h)`
on pourra utiliser
`gluPerspective(65.,(double) w/h,1.0,20.0)`
ce qui revient à adapter le ratio de gluPerspective au ratio du viewport.

Utilisation du reshape : homothétique	
	
Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Opérations diverses liées à la modélisation et à la visualisation

- `void glPushMatrix(void);`

  Empile la matrice courante dans la pile de matrices. ATTENTION, il existe deux piles, une en mode GL_MODELVIEW et une en mode GL_PROJECTION.

- `void glPopMatrix(void);`

  Dépile la matrice en haut de pile du mode courant et remplace la matrice courante par celle-ci.

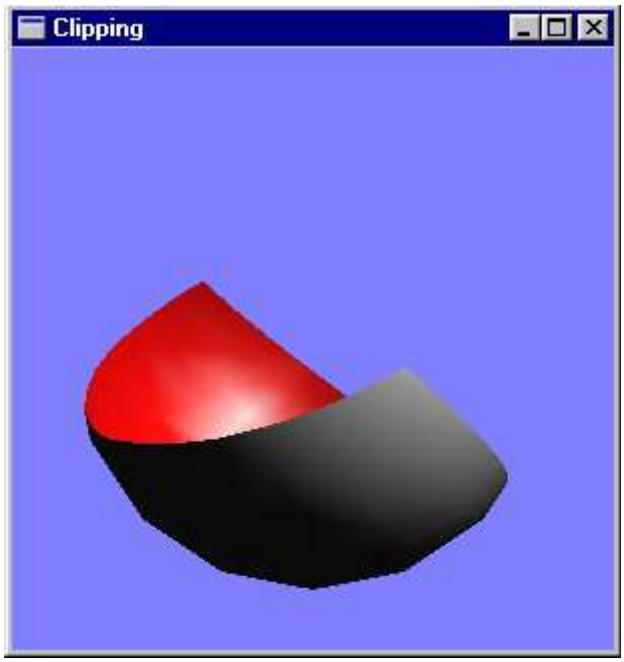
- `void glClipPlane(GLenum plane,const GLdouble *equ);`

  Définit un plan de clipping 3D (découpage).

  equ: tableau de quatre coefficients (a,b,c,d) qui définissent l'équation ax+by+cz+d = 0 du plan de clipping.

  plane: **GL_CLIP_PLANEi** avec i compris entre 0 et 5.

  glClipPlane(plane,equ) est à utiliser en association avec glEnable(plane) et glDisable(plane) pour autoriser/interdire la gestion du plan de clipping plane.

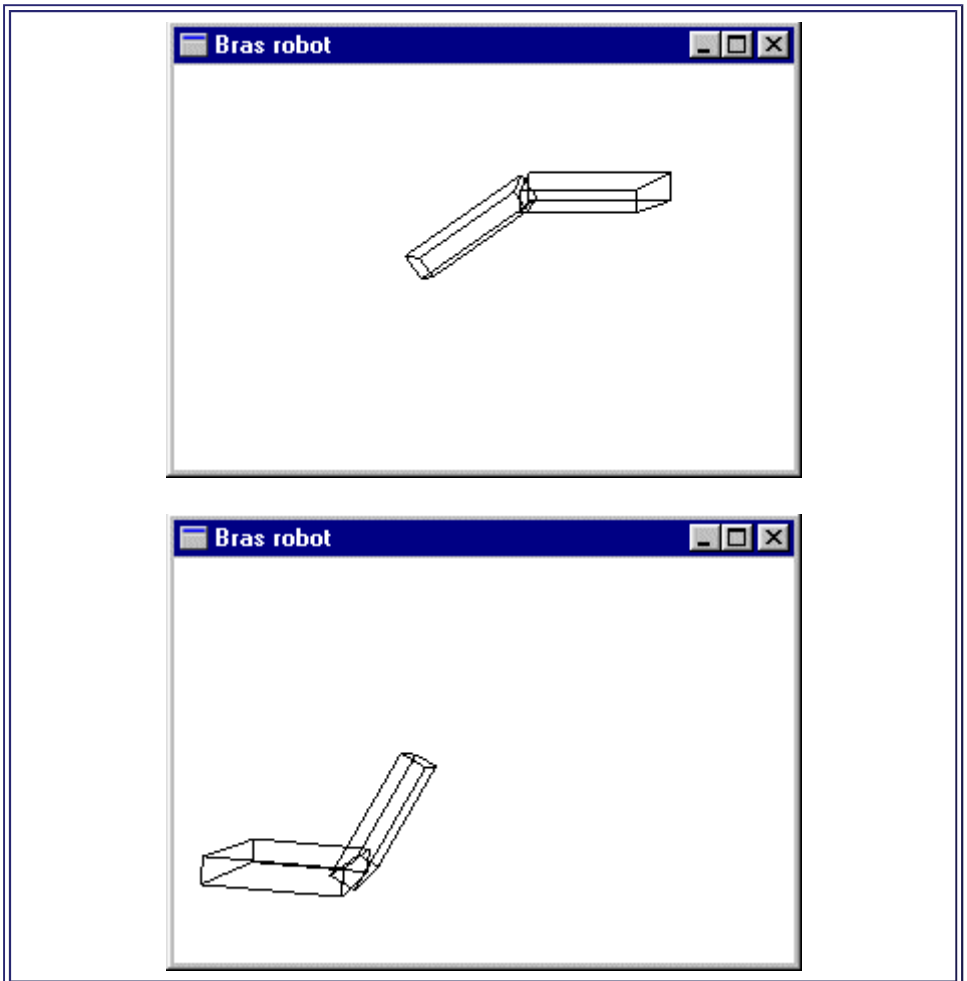
Découpage

Programme Aux / Programme GLUt
Exécutable Aux / Exécutable GLUt

Exemples

```
#include <GL/gl.h>
#include <GL/glu.h>
```

```
#include "glaux.h"

static int shoulder = 0, elbow = 0;

void elbowAdd(void) {
  elbow = (elbow + 5) % 360;
}

void elbowSubtract(void) {
  elbow = (elbow - 5) % 360;
}

void shoulderAdd(void) {
  shoulder = (shoulder + 5) % 360;
}

void shoulderSub(void) {
  shoulder = (shoulder - 5) % 360;
}

void display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0,1.0,1.0);
  glPushMatrix();
  glTranslatef(-1.0,0.0,0.0);
  glRotatef((GLfloat) shoulder,
            0.0,0.0,1.0);
  glTranslatef(1.0,0.0,0.0);
  auxWireBox(2.0,0.4,1.0);
  glTranslatef(1.0,0.0,0.0);
  glRotatef((GLfloat) elbow,
            0.0,0.0,1.0);
  glTranslatef(1.0,0.0,0.0);
  auxWireBox(2.0,0.4,1.0);
  glPopMatrix();
  glFlush();
}

void myinit(void) {
  glShadeModel(GL_FLAT);
}

void myReshape(int w,int h) {
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(65.,(float)w/(float)h,
                 1.0,20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
```

```
  glTranslatef(0.0,0.0,-5.0);
}

int main(int argc,char** argv) {
  auxInitDisplayMode(AUX_SINGLE|
                     AUX_RGB|
                     AUX_DIRECT);
  auxInitPosition(0,0,400,400);
  auxInitWindow(argv[0]);
  myinit();
  auxKeyFunc(AUX_LEFT,shoulderSub);
  auxKeyFunc(AUX_RIGHT,shoulderAdd);
  auxKeyFunc(AUX_UP,elbowAdd);
  auxKeyFunc(AUX_DOWN,elbowSubtract);
  auxReshapeFunc(myReshape);
  auxMainLoop(display);
}
```

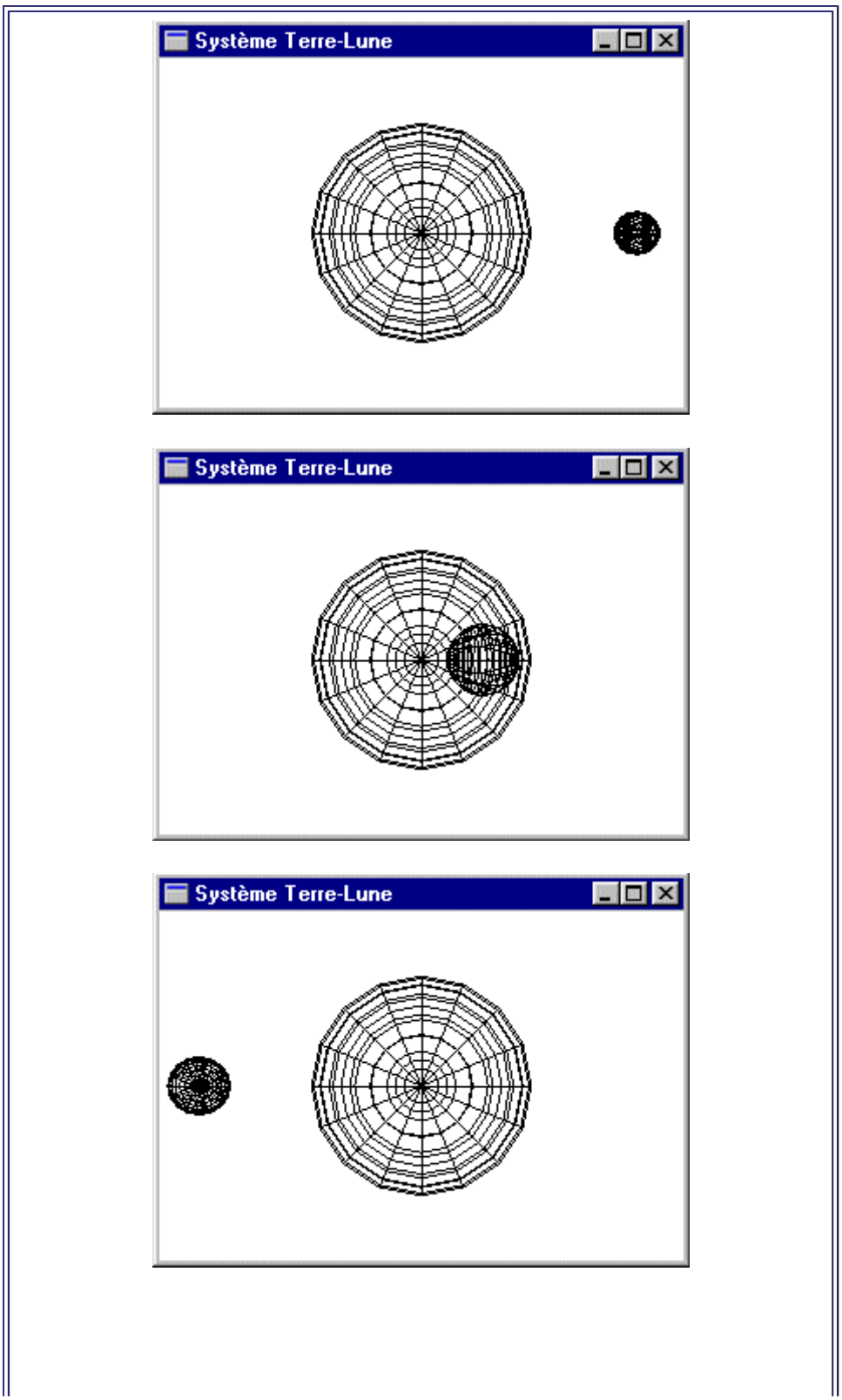


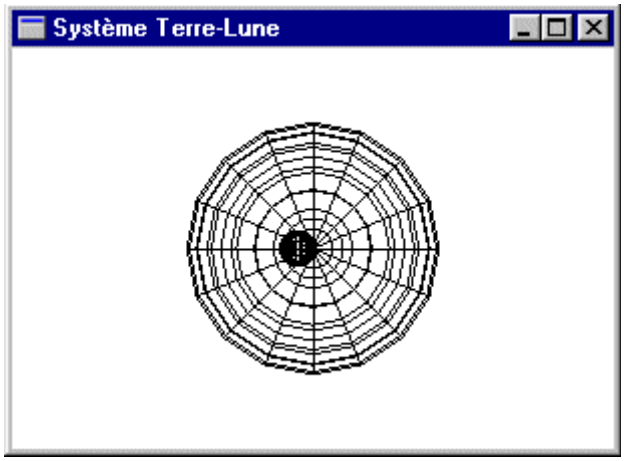
Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

```
#include <GL/gl.h>
#include <GL/glu.h>
#include "glaux.h"
```

```
static int year = 0, day = 0;

void dayAdd(void) {
  day = (day + 10) % 360;
}

void daySubtract(void) {
  day = (day - 10) % 360;
}

void yearAdd(void) {
  year = (year + 5) % 360;
}

void yearSubtract(void) {
  year = (year - 5) % 360;
}

void display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0,1.0,1.0);
  glPushMatrix();
  auxWireSphere(1.0);
  glRotatef((GLfloat) year,
            0.0,1.0,0.0);
  glTranslatef(2.0,0.0,0.0);
  glRotatef((GLfloat) day,
            0.0,1.0,0.0);
  auxWireSphere(0.2);
  glPopMatrix();
  glFlush();
}

void myinit(void) {
  glShadeModel(GL_FLAT);
}

void myReshape(int w,int h) {
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0,
                 (float) w/(float) h,
                 1.0,20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0,0.0,-5.0);
}

int main(int argc,char** argv) {
```

```
    auxInitDisplayMode(AUX_SINGLE|
                       AUX_RGB);
    auxInitPosition(0,0,500,500);
    auxInitWindow(argv[0]);
    myinit();
    auxKeyFunc(AUX_LEFT,yearSubtract);
    auxKeyFunc(AUX_RIGHT,yearAdd);
    auxKeyFunc(AUX_UP,dayAdd);
    auxKeyFunc(AUX_DOWN,daySubtract);
    auxReshapeFunc(myReshape);
    auxMainLoop(display);
}
```





Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Les listes d'affichage

Définition

Une liste d'affichage est une suite de commandes OpenGL stockées pour une utilisation future.

Quand une liste d'affichage est invoquée, les commandes qu'elle contient sont exécutées dans l'ordre où elles ont été stockées.

Les commandes d'une liste d'affichage sont les mêmes que les commandes d'affichage immédiat.

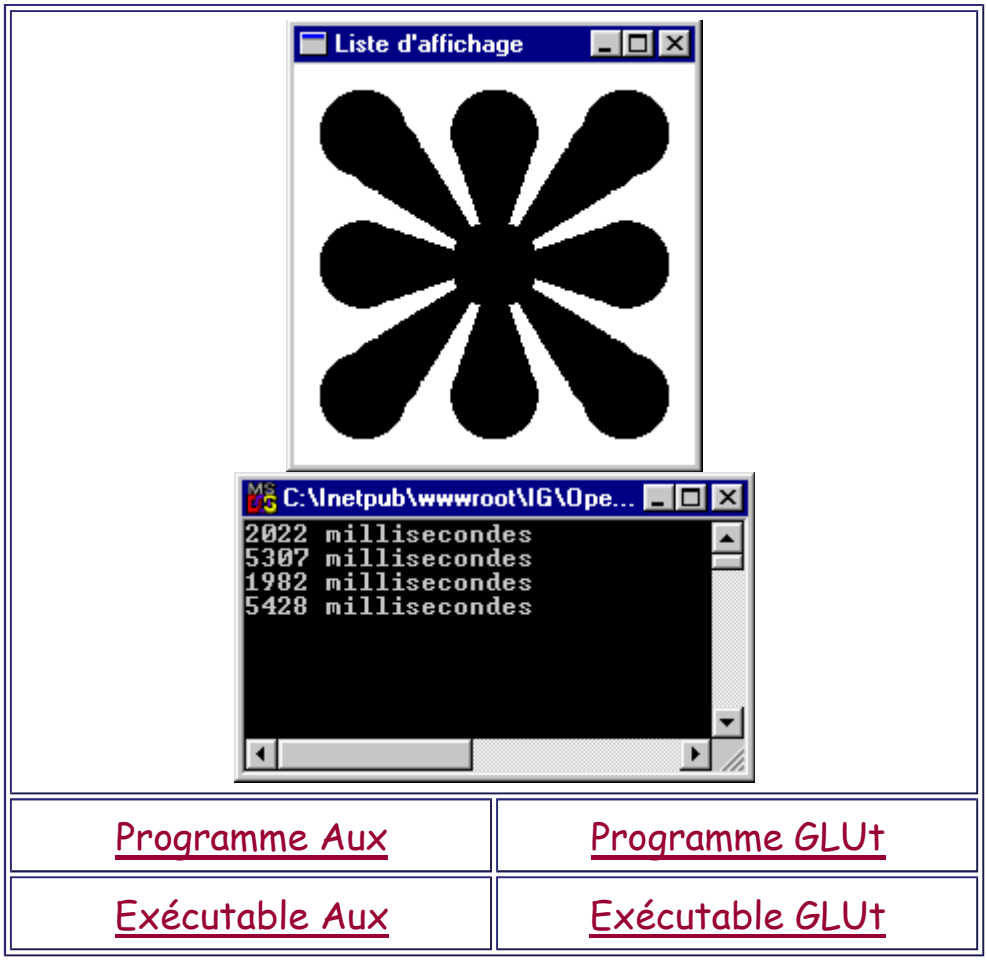
Exemple

Dessin d'un polygone régulier de 100 sommets.

```
void cercle(){
  GLint i ;
  GLfloat cosinus,sinus ;
  glBegin(GL_POLYGON) ;
  for ( i = 0 ; i < 100 ; i++ ) {
    cosinus = cos(i*2*PI/100.0);
    sinus = sin(i*2*PI/100.0);
    glVertex2f(cosinus,sinus) ; }
  glEnd() ;
}
```

Cette méthode sans liste d'affichage est inefficace car tout appel à cercle() entraîne l'exécution des calculs de sinus et cosinus.

```
#define CERCLE 1
void cercle() {
  GLint i ;
  GLfloat cosinus,sinus ;
  glNewList(CERCLE,GL_COMPILE) ;
  glBegin(GL_POLYGON) ;
  for ( i = 0 ; i < 100 ; i++ ) {
    cosinus = cos(i*2*PI/100.0);
    sinus = sin(i*2*PI/100.0);
    glVertex2f(cosinus,sinus) ; }
  glEnd() ;
  glEndList() ;
}

glCallList(CERCLE) ;
```



Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Fonctionnalités

Les listes d'affichages sont des macros et, à ce titre, ne peuvent pas être modifiées ou paramétrées.

Les listes d'affichage optimisent les temps de calcul et d'affichage car elles permettent la suppression de calculs redondants sur:

- les opérations matricielles,

- les conversions de formats d'image bitmap,
- les calculs de lumière,
- les calculs de propriétés des matériaux de surface,
- les calculs de modèle d'éclairage,
- les textures,
- les motifs de tracé.

L'inconvénient principal des listes d'affichage est qu'elles peuvent occuper beaucoup de place en mémoire.

Syntaxe des commandes

Commandes principales

- `void glNewList(GLuint liste,GLenum mode);`

  Indique le début de la définition d'une liste d'affichage.

  liste: entier positif unique qui identifiera la liste.

  mode: **GL_COMPILE** ou **GL_COMPILE_AND_EXECUTE** suivant que l'on désire seulement stocker la liste ou la stocker et l'exécuter simultanément.

- `void glEndList(void);`

  Indique la fin d'une liste d'affichage.

  Toutes les commandes depuis le dernier glNewList ont été placées dans la liste d'affichage (quelques exceptions).

- `void glCallList(GLuint liste);`

  Exécute la liste d'affichage référencée par liste.

Commandes auxiliaires

- `GLuint glGenLists(GLsizei nb);`

  Alloue nb listes d'affichage contiguës, non allouées auparavant.

L'entier retourné est l'indice qui donne le début de la zone libre de références.

- `GLboolean glIsList(GLuint liste);`

  Retourne vrai si liste est utilisé pour une liste d'affichage, faux sinon.

- `void glDeleteLists(GLuint l,GLsizei nb);`

  Détruit les nb listes à partir de la liste l.

Exemple

```
#include <GL/gl.h>
#include <GL/glu.h>
#include "glaux.h"

GLuint listName = 1;

void myinit(void) {
  glNewList(listName,GL_COMPILE);
  glColor3f(1.0,0.0,0.0);
  glBegin(GL_TRIANGLES);
  glVertex2f(0.0,0.0);
  glVertex2f(1.0,0.0);
  glVertex2f(0.0,1.0);
  glEnd();
  glTranslatef(1.5,0.0,0.0);
  glEndList();
  glShadeModel(GL_FLAT);
}

void drawLine(void) {
  glBegin(GL_LINES);
  glVertex2f(0.0,0.5);
  glVertex2f(15.0,0.5);
  glEnd();
}

void display(void) {
  GLuint i;
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0,1.0,0.0);
  for ( i = 0 ; i < 10 ; i++ )
    glCallList(listName);
  drawLine();
  glFlush();
}
```

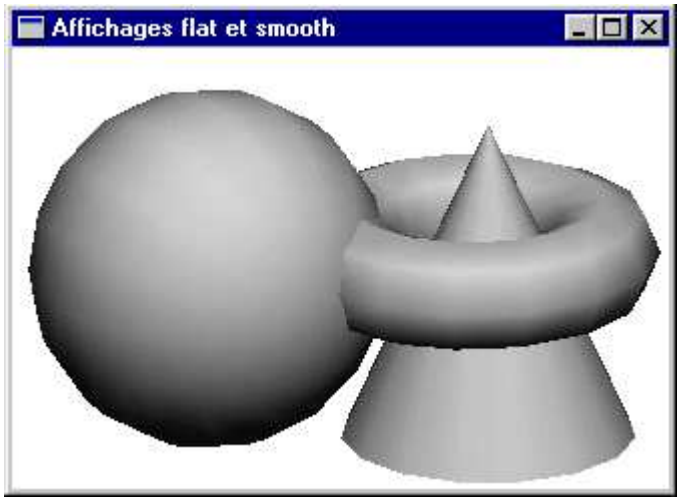
```
void myReshape(int w,int h) {
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if ( w <= h )
    gluOrtho2D(0.,2.,
               -0.5*(float)h/(float)w,
               1.5*(float)h/(float)w);
    else
    gluOrtho2D(0.,
               2.*(float)w/(float)h,
               -.5,1.5);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

int main(int argc,char** argv) {
  auxInitDisplayMode(AUX_SINGLE|
                     AUX_RGB);
  auxInitPosition(0,0,400,50);
  auxInitWindow(argv[0]);
  myinit();
  auxReshapeFunc(myReshape);
  auxMainLoop(display);
}
```

Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

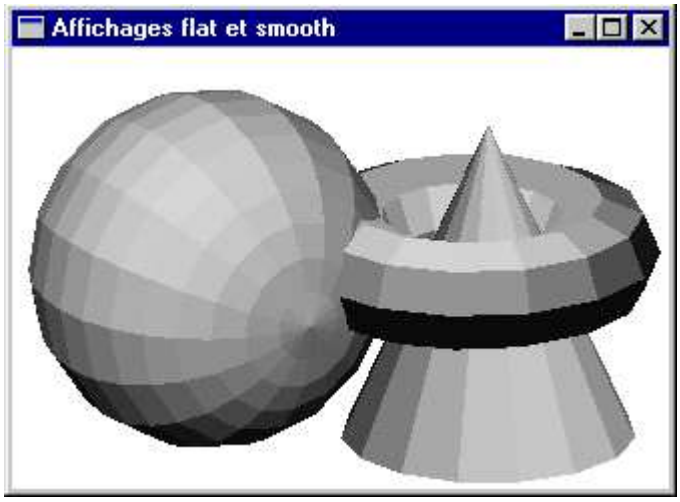
Mode de lissage

- `void glShadeModel(GLenum mode)`

  mode: mode de lissage (**GL_FLAT**: uni, **GL_SMOOTH**: Gouraud)



Lissé (utilisé par défaut)



Non lissé

Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Les calculs d'illumination

Les matériaux et les lumières

La couleur d'un objet dépend des pourcentages (coefficients) des lumières rouges, vertes et bleues (RVB) le touchant qu'il réfléchit par réflexions ambiante, diffuse et spéculaire et de la "lumière" qu'il "émet" intrinsèquement (pour lui-même).

Réflexions ambiante et diffuse, lumière émise -> couleur intrinsèque d'un objet

Réflexion spéculaire -> couleurs des lumières de la scène (généralement blanches, grises ou jaunes)

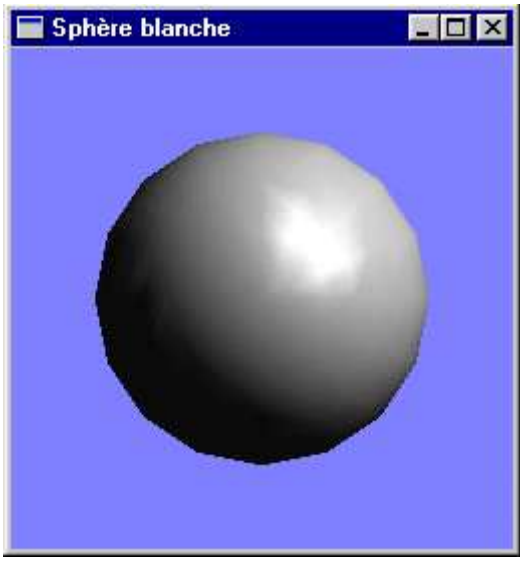
Modélisation de l'éclairage reçu sur un objet en un point vu par l'observateur par trois catégories de lumières:

- de la lumière ambiante provenant de réflexions multiples et donc uniformément de toutes les directions (réfléchie dans toutes les directions et donc visible de partout),
- de la lumière diffuse provenant d'une direction particulière (réfléchie dans toutes les directions et donc visible de partout),
- de la lumière spéculaire provenant d'une direction particulière (réfléchie dans une direction préférentielle et donc visible seulement si l'observateur est situé dans cette direction).

Modéliser l'éclairage d'une scène nécessite de définir toutes les lumières y existant mais aussi d'attribuer un matériel à chaque objet de la scène.

Exemple

```
#include <GL/gl.h>

void myinit(void) {
  GLfloat diffuse[] = { 0.9,0.9,0.9,1.0 };
  GLfloat specular[] = { 1.0,1.0,1.0,1.0 };
  GLfloat shinines[] = { 50.0 };
  GLfloat l_pos[] = { 1.0,1.0,1.0,0.0 };
  glMaterialfv(GL_FRONT,GL_DIFFUSE,diffuse);
  glMaterialfv(GL_FRONT,GL_SPECULAR,specular);
  glMaterialfv(GL_FRONT,GL_SHININESS,shinines);
  glLightfv(GL_LIGHT0,GL_POSITION,l_pos);
  glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuse);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
}
```



Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Définition des propriétés d'un matériau

- `void glMaterial{i f}[v](GLenum f,GLenum pn,TYPE v);`

  Configuration des caractéristiques d'un matériau

  f: **GL_FRONT**, **GL_BACK** ou **GL_FRONT_AND_BACK** pour spécifier quelle face est configurée
  pn: Caractéristique affectée
  v: Valeurs affectées à la caractéristique pn

pn	*Rôle*
GL_AMBIENT	Couleur ambiante (coefficient ambiant)
GL_DIFFUSE	Couleur diffuse (coefficient de diffusion)
GL_AMBIENT_AND_DIFFUSE	Couleurs ambiante et diffuse
GL_SPECULAR	Couleur spéculaire (coefficient de réflexion)
GL_SHININESS	Focalisation spéculaire (0 -> 128)
GL_EMISSION	Couleur émise (coefficient d'émission)
GL_COLOR_INDEXES	Indexes des couleurs ambiante, diffuse et spéculaire

pn	*Valeur par défaut*
GL_AMBIENT	(0.2,0.2,0.2,1.0)
GL_DIFFUSE	(0.8,0.8,0.8,1.0)
GL_AMBIENT_AND_DIFFUSE	
GL_SPECULAR	(0.0,0.0,0.0,1.0)
GL_SHININESS	0.0
GL_EMISSION	(0.0,0.0,0.0,1.0)
GL_COLOR_INDEXES	(0,1,1)

Caractéristiques possiblement différentes pour les différents objets d'une scène

Pour une même primitive graphique, utilisation des mêmes caractéristiques de matériel pour chacun de ses sommets ou bien changement d'un sommet à un autre -> dégradé de matériaux

Récupération de la configuration du matériel

- `void glGetMaterial{i f}v(GLenum f,GLenum pn,TYPE *v);`

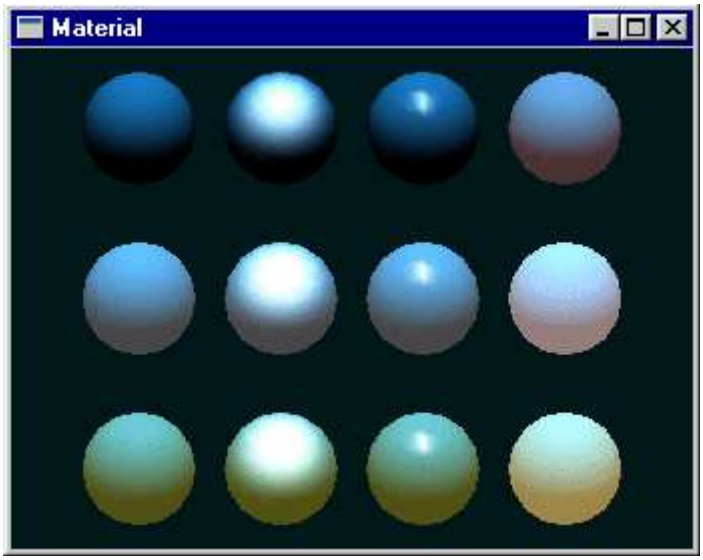
  Lecture des caractéristiques du matériel courant

  f: **GL_FRONT** ou **GL_BACK** pour spécifier quelle face est concernée par l'opération
  pn: Caractéristique lue
  v: Tableau affecté au résultat de la lecture de la caractéristique pn

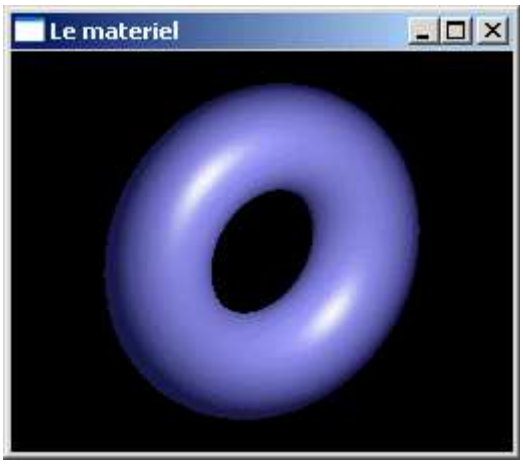
Exemple

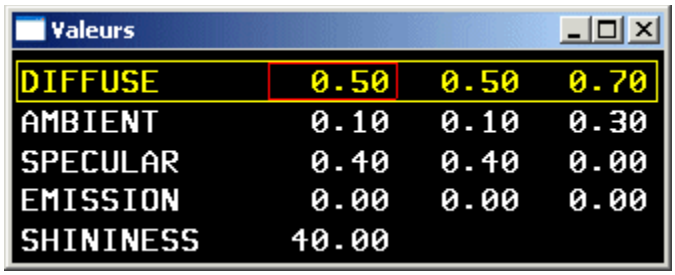
```
void myinit(void) {
  GLfloat no_mat[] = { 0.0F,0.0F,0.0F,1.0F };
  GLfloat mat_diffuse
[] = { 0.1F,0.5F,0.8F,1.0F };
  GLfloat no_shininess[] = { 0.0F };
  glMaterialfv(GL_FRONT,GL_AMBIENT,no_mat);
  glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
  glMaterialfv(GL_FRONT,GL_SPECULAR,no_mat);
  glMaterialfv
```

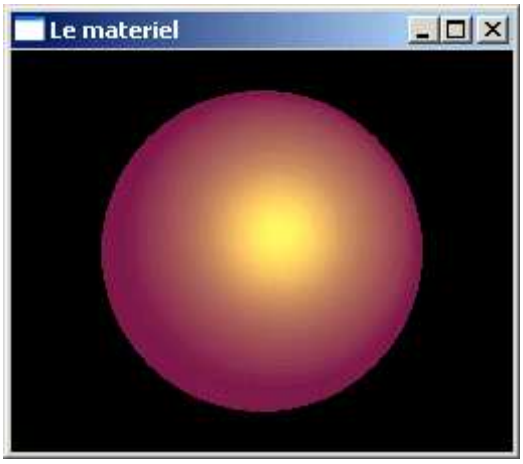
```
(GL_FRONT,GL_SHININESS,no_shininess);
  glMaterialfv(GL_FRONT,GL_EMISSION,no_mat);
```

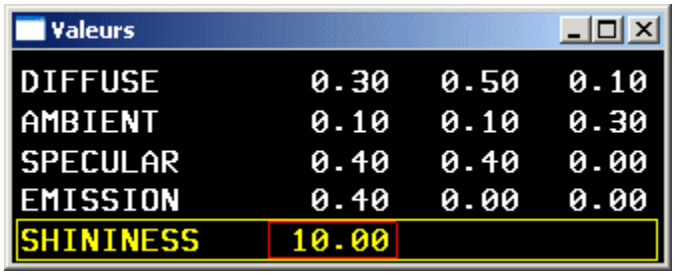


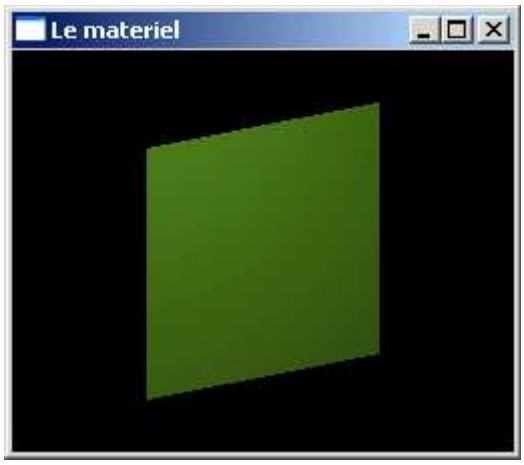
Le programme Aux	Le programme GLUt
Exécutable Aux	Exécutable GLUt

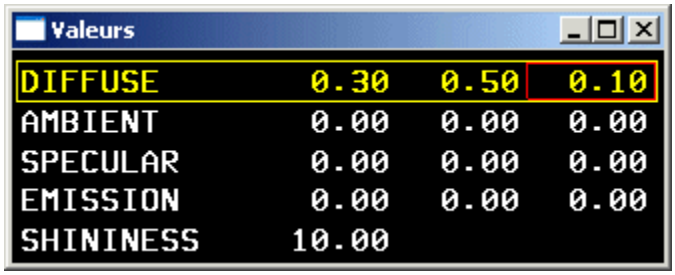












Programme GLUt

Exécutable GLUt

Configuration de sources lumineuses

Gestion d'un nombre minimum de 8 sources lumineuses désignées par les constantes GL_LIGHT0, GL_LIGHT1, GL_LIGHT2,...

Les paramètres de ces sources doivent être renseignés.

- `void glLight{i f}[v](GLenum nb,GLenum pname,TYPE p);`

  Configuration d'une source lumineuse

  nb: **GL_LIGHT0**, **GL_LIGHT1**, ..., **GL_LIGHT7**, soit au total 8 sources lumineuses possibles en VC++
  pname: Caractéristique affectée
  p: Valeur affectée à la caractéristique pname

pname	*Rôle*
GL_AMBIENT	(R,G,B,A): Energie colorée de la lumière ambiante ajoutée à la scène
GL_DIFFUSE	(R,G,B,A): Energie colorée de la lumière diffuse liée à la source lumineuse
GL_SPECULAR	(R,G,B,A): Energie colorée de la lumière spéculaire liée à la source lumineuse
GL_POSITION	(x,y,z,w): "Position" de la lumière • si w = 0, lumière placée à l'infini, (x,y,z) donne la direction vers la source de lumière -> lumière directionnelle • si w != 0, lumière non placée à l'infini, (x,y,z) donne la position -> lumière ponctuelle
GL_SPOT_DIRECTION	Orientation du cone d'ouverture dans le cas d'un spot
GL_SPOT_EXPONENT	Exposant (focalisation) d'un spot (plus l'exposant est grand plus le spot est focalisé sur son axe)
GL_SPOT_CUTOFF	Angle d'ouverture d'un spot

pname	*Rôle*
GL_CONSTANT_ATTENUATION	Facteur d'attenuation constant: Kc
GL_LINEAR_ATTENUATION	Facteur d'attenuation linéaire: Kl
GL_QUADRATIC_ATTENUATION	Facteur d'attenuation quadratique: Kq

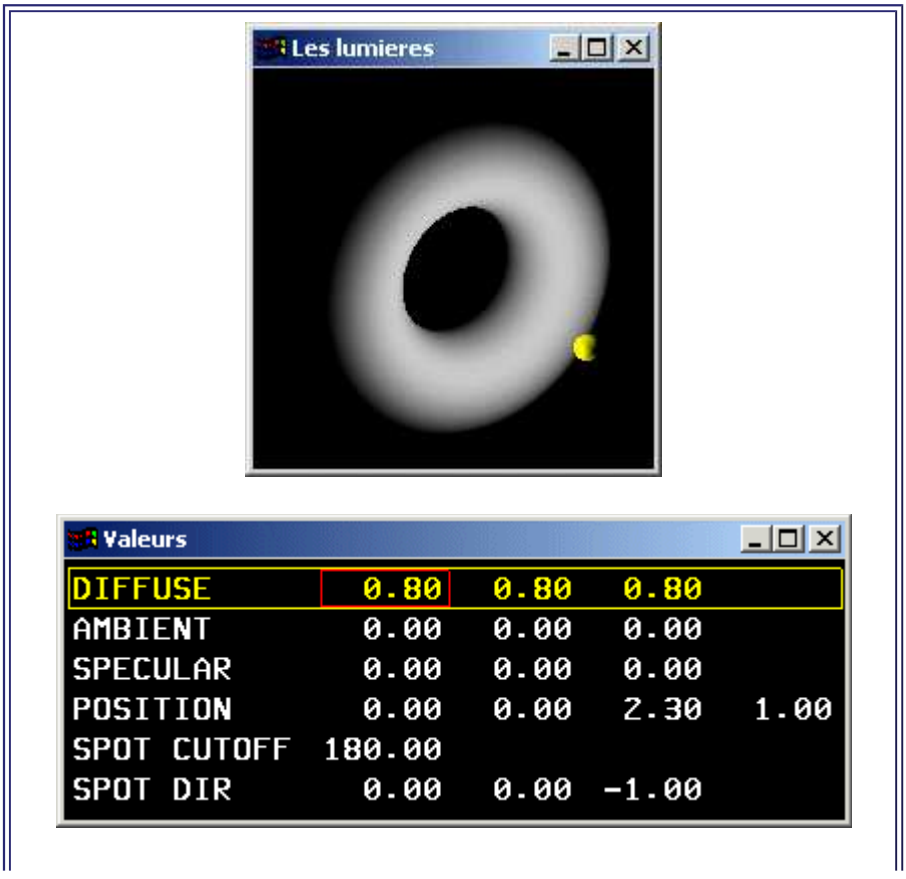
Facteur d'atténuation en fonction de la distance d entre une lumière ponctuelle et un point éclairé: $\frac{1}{(kc + d * Kl + d^2 * Kq)}$

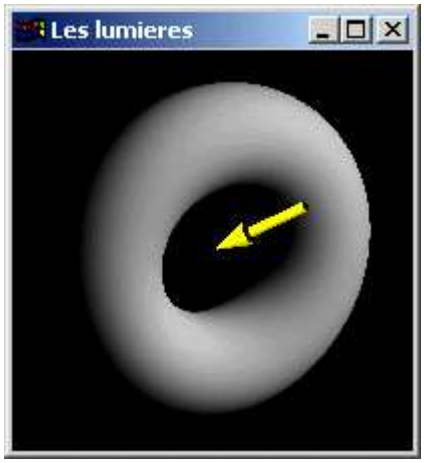
pname	*Valeur par défaut*
GL_AMBIENT	(0.0,0.0,0.0,1.0)

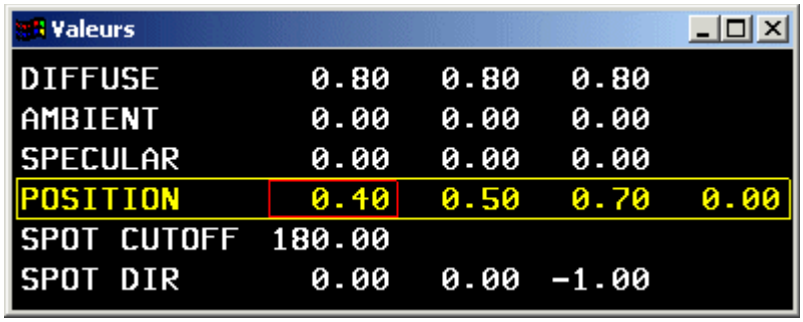
GL_DIFFUSE	GL_LIGHT0:(1.0,1.0,1.0,1.0) autres:(0.0,0.0,0.0,1.0)
GL_SPECULAR	GL_LIGHT0:(1.0,1.0,1.0,1.0) autres:(0.0,0.0,0.0,1.0)
GL_POSITION	(0.0,0.0,1.0,0.0)
GL_SPOT_DIRECTION	(0.0,0.0,-1.0)
GL_SPOT_EXPONENT	0.0
GL_SPOT_CUTOFF	180.0
GL_CONSTANT_ATTENUATION	1.0
GL_LINEAR_ATTENUATION	0.0
GL_QUADRATIC_ATTENUATION	0.0

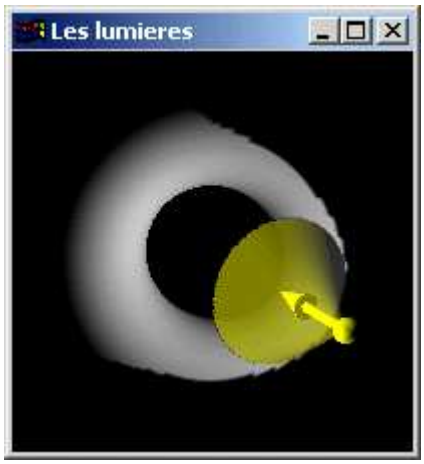
Exemple

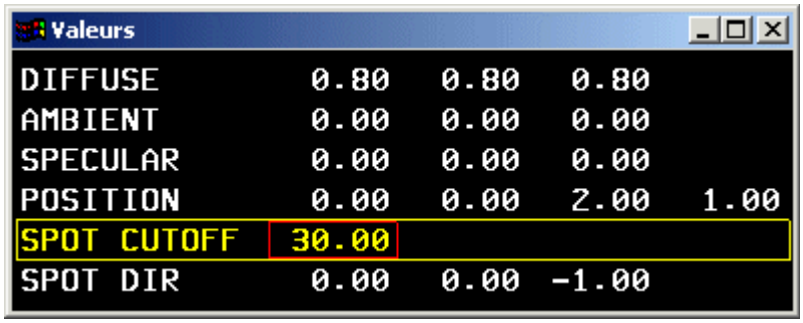
```
void myinit(void) {
  GLfloat amb[] = {0.0,0.0,0.0,1.0};
  GLfloat dif[] = {1.0,1.0,1.0,1.0};
  GLfloat spe[] = {1.0,1.0,1.0,1.0};
  GLfloat pos[] = {0.0,3.0,2.0,0.0};
  glLightfv(GL_LIGHT0,GL_AMBIENT,amb);
  glLightfv(GL_LIGHT0,GL_DIFFUSE,dif);
  glLightfv(GL_LIGHT0,GL_SPECULAR,spe);
  glLightfv(GL_LIGHT0,GL_POSITION,pos);
  ...
```







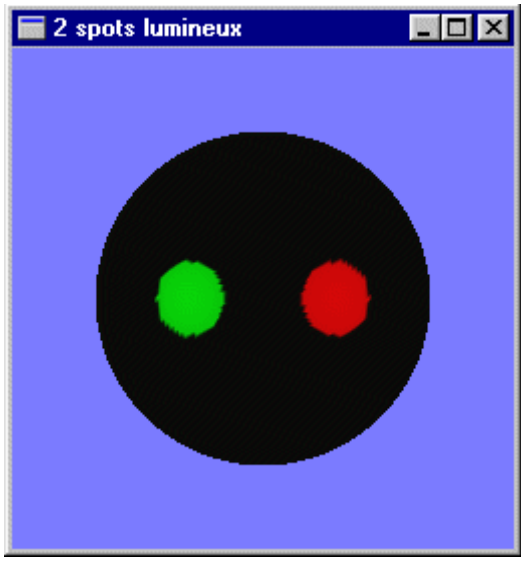




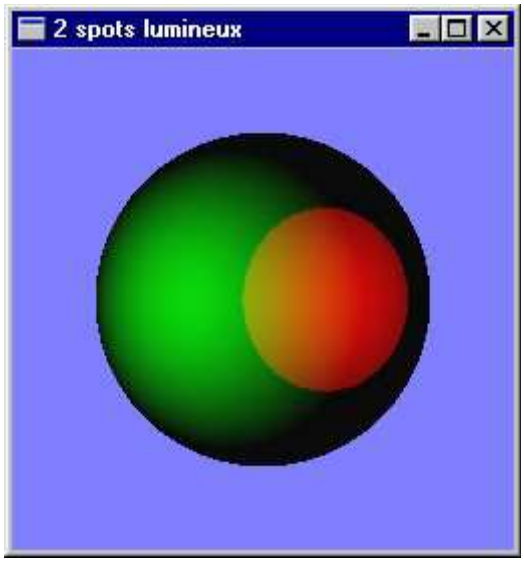
Programme GLUt

Exécutable GLUt

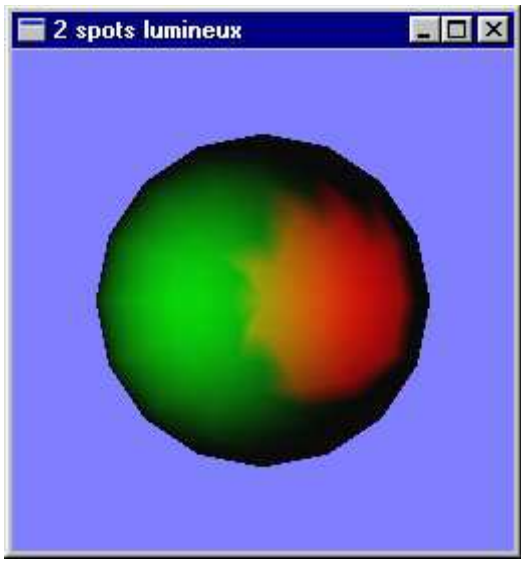
Exemples de définition de spots



Deux spots très focalisés



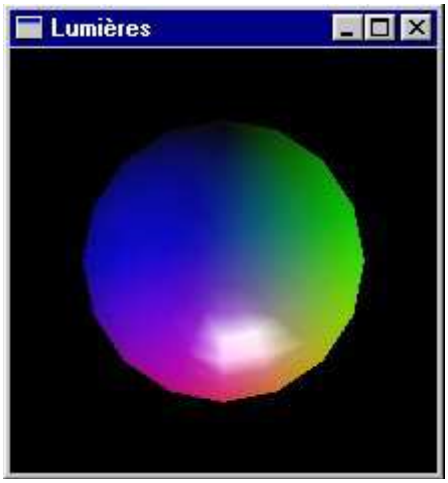
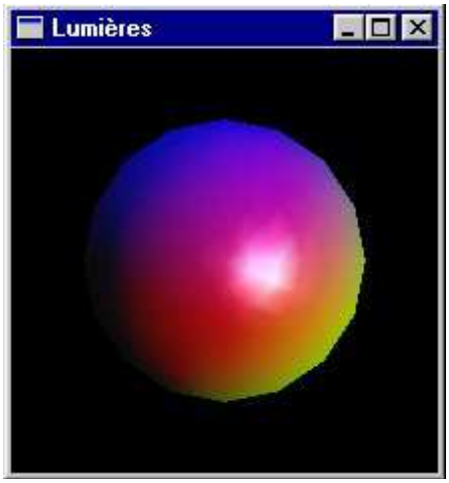
Ces deux spots moins focalisés



Aberrations liées à la facettisation des objets à courbure

Le programme Aux et GLUt

Exécutable Aux et GLUt

Obtention de lumières colorées	
	
Programme Aux	Programme GLUt et Aux
Exécutable Aux	Exécutable GLUt et Aux

Lecture de la configuration des sources lumineuses

- `void glGetLight{i f}v(GLenum nb,GLenum pname,TYPE *p);`

  Lecture de la configuration d'une des sources lumineuses

  nb: **GL_LIGHT0**, **GL_LIGHT1**, ..., **GL_LIGHT7**
  pname: Caractéristique lue
  p: Tableau destiné à la récupération de la caractéristique pname

Autorisation des calculs d'éclairage

Les calculs d'éclairage ne sont réalisés qu'après exécution de:

- `glEnable(GL_LIGHTING);`

Faute de cette autorisation, ce sont les informations de couleurs spécifiées par les fonctions glColor* qui sont prises en compte.

Chaque lumière (8 au maximum) doit être elle-aussi autorisée:

- `glEnable(GL_LIGHTi);`

Ces options peuvent être annulées au moyen de **glDisable**.

Normalisation automatique

Les normales spécifiées sur les objets subissent les transformations géométriques GL_MODELVIEW et GL_PROJECTION.
-> Problème s'il y a des scales ou une visualisation en perspective car alors une normale peut ne plus conserver sa norme égale à 1.0. Ceci peut conduire à des erreurs dans les calculs d'illumination.

- `glEnable(GL_NORMALIZE);`

  Normalisation automatique systématique avant utilisation de tout vecteur connu comme étant une normale.

Choix d'un modèle d'illumination

Le modèle d'illumination OpenGL gère les trois caractéristiques suivantes:

- l'intensité de lumière ambiante globale permettant de fixer l'éclairage minimum au sein de la scène via les composantes ambiantes des matériaux,

- la position du point de vue (position de l'observateur, local ou à l'infini) utilisé pour l'évaluation des réflexions spéculaires,

- des calculs d'éclairage différents ou non pour les deux faces des facettes modélisant les objets.

- `void glLightModel{if}[v](GLenum md,TYPE v);`

  Configuration du modèle d'illumination

  md: **GL_LIGHT_MODEL_AMBIENT**, **GL_LIGHT_MODEL_LOCAL_VIEWER**, **GL_LIGHT_MODEL_TWO_SIDE**

  v: valeur affectée à md

Le blending (transparence)

OpenGL permet d'afficher un objet avec composition de la couleur de chacun de ses pixels avec la couleur du pixel déjà présent dans l'image à cette place -> transparence.

- `glEnable(GL_BLEND);`

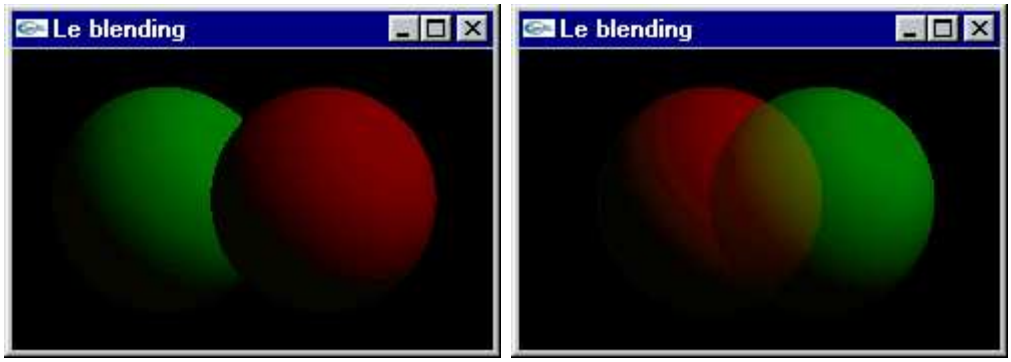
  Activation du blending.

- `glBlendFunc(GLenum sfactor,GLenum dfactor);`

  Choix de l'arithmétique de composition des pixels sources et destination (sfactor pour les pixels de l'objet affiché, dfactor pour les pixels déjà présents).

Pour le blending les valeurs habituelles de sfactor et dfactor sont **GL_SRC_ALPHA** et **GL_ONE_MINUS_SRC_ALPHA**
-> on garde 1-alpha du pixel présent et on ajoute alpha du pixel calculé
-> objet non transparent avec alpha = 1
-> objet transparent avec alpha = 0.

ATTENTION: La composition entre les pixels présents et placés implique que l'ordre de dessin des objets graphiques a une influence sur l'image finale.



	GLUt	
Deux sphères transparentes	Src Src	

Les bitmaps

Introduction

OpenGL permet la gestion de bitmaps en autorisant les opérations de transfert entre le buffer couleur et la mémoire centrale.

Buts

- gestion de fichiers
- gestion de polices de caractères (non traité)
- gestion de textures (voir plus loin)

Commandes

- `void glReadPixels(GLint x,GLint y,GLsizei l,GLsizei h,GLenum format,GLenum type,GLvoid *pixels);`

  Lecture des pixels d'un frame-buffer et stockage en mémoire

  x et y: position du coin supérieur gauche du rectangle de pixels

  l et w: dimensions du rectangle de pixels

  format: information à lire sur les pixels

  type: type du résultat à fournir

  pixels: tableau destiné à recevoir les résultats

format	*Type de données pour les pixels*
GL_COLOR_INDEX	Indexe de couleurs
GL_RGB	Couleurs RVB
GL_RGBA	Couleurs RVBA
GL_RED	Composante rouge
GL_GREEN	Composante verte
GL_BLUE	Composante bleue
GL_ALPHA	Composante alpha
GL_LUMINANCE	Luminance
GL_LUMINANCE_ALPHA	Luminance puis composante alpha

GL_STENCIL_INDEX	Stencil
GL_DEPTH_COMPONENT	Composante de profondeur

type	*Type C*
GL_UNSIGNED_BYTE	Entier 8 bits non signé
GL_BYTE	Entier 8 bits signé
GL_BITMAP	Bits dans des entiers 8 bits non signés
GL_UNSIGNED_SHORT	Entier 16 bits non signé
GL_SHORT	Entier 16 bits signé
GL_UNSIGNED_INT	Entier 32 bits non signé
GL_INT	Entier 32 bits signé
GL_FLOAT	Réel simple précision

- **`void glDrawPixels(GLsizei l,GLsizei h,GLenum f,GLenum type,GLvoid *pixels);`**

  Écriture dans un frame-buffer de pixels stockés en mémoire. Le rectangle de pixels est affiché avec son coin supérieur gauche en position raster courante

  l et w: dimensions du rectangle de pixels

  f: information à écrire sur les pixels

  type: type des données transmises

  pixels: tableau destiné à transmettre les pixels

- **`void glRasterPos{234}{sifd}{v}(TYPE x,TYPE y,TYPE z,TYPE w);`**

  Fixe la position raster courante

  Si 2 est utilisé z est fixé à 0 et w à 1. Si c'est 3, w est fixé à 1

- **`void glCopyPixels(GLint x,GLint y,GLsizei l,GLsizei h,GLenum type);`**

Copie depuis un buffer dans lui-même

x et y: position du coin supérieur gauche du rectangle source (la position du rectangle destination donnée par la position raster courante)

l et w: dimensions du rectangle source

type: buffer sur lequel agir (**GL_COLOR**, **GL_STENCIL** ou **GL_DEPTH**)

Exemple: Sauvegarde d'une image

```
void sauve(char *nom,int px,
           int py,int dx,int dy){
  GLubyte *im ;
  FILE *num ;
  int y,x,t = dx*dy*3 ;
  im =(GLubyte *) calloc(t,
                    sizeof(GLubyte));
  glReadPixels(px,py,dx,dy,GL_RGB,
               GL_UNSIGNED_BYTE,im);
  num = fopen(nom,"wb") ;
  if ( num ) {
    for ( y = dy-1 ; y >= 0 ; y-- ) {
      for ( x = 0 ; x < dx ; x++ ) {
        fwrite((void *)&im[(x+y*dx)*3],
               sizeof(GLubyte),
               3,num) ; } }
  fclose(num) ; }
  free(im) ;
}
```

Ce programme lit une portion rectangulaire (px, py, px+dx, py+dy) de la fenêtre d'affichage et la stocke dans un tableau.

Ce tableau est ensuite sauvegardé dans un fichier.

Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Suite