OpenGL (Partie 3)

Placage de texture

Introduction

Les textures sont utilisables en OpenGL. Elles permettent d'éviter de rendre les objets trop complexes par facettisation.

Etapes de l'utilisation d'une texture:

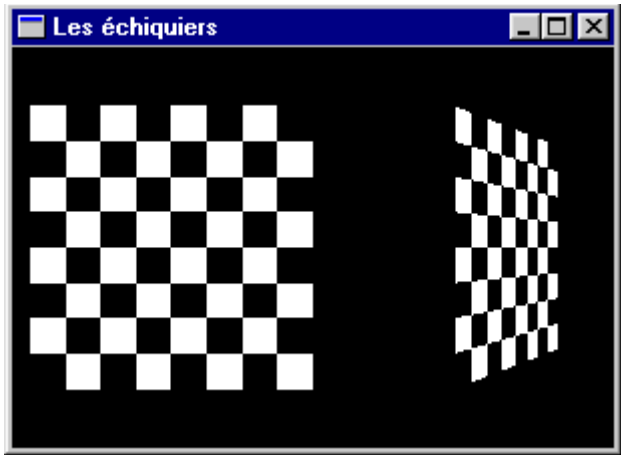
(1) spécification de la texture,

(2) spécification de la technique à utiliser pour appliquer la texture à chaque pixel,

(3) autorisation du plaquage,

(4) dessin de la scène qui sera implicitement texturée au moyen des paramètres définis en (1) et (2).

Exemple



Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

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

RETOUR

```
#define LI 64
#define LH 64
GLubyte image[LI][LH][3];

void makeImage(void) {
  int i,j,r,c;
  for( i = 0 ; i < LI ; i++ ) {
    for( j = 0 ; j < LH ; j++ ) {
      c = (((i&0x8)==0)^
           ((j&0x8)==0))*255;
      image[i][j][0] =(GLubyte) c;
      image[i][j][1] =(GLubyte) c;
      image[i][j][2] =(GLubyte) c; } }
}

void myinit(void) {
  glClearColor(0.0,0.0,0.0,0.0);
  glEnable(GL_DEPTH_TEST);
  glDepthFunc(GL_LESS);
  makeImage();
  glPixelStorei(GL_UNPACK_ALIGNMENT,1);
  glTexImage2D(GL_TEXTURE_2D,0,3,
               LI,LH,0,GL_RGB,
               GL_UNSIGNED_BYTE,
               &image[0][0][0]);
  glTexParameterf(GL_TEXTURE_2D,
                  GL_TEXTURE_WRAP_S,
                  GL_CLAMP);
  glTexParameterf(GL_TEXTURE_2D,
                  GL_TEXTURE_WRAP_T,
                  GL_CLAMP);
  glTexParameterf(GL_TEXTURE_2D,
                  GL_TEXTURE_MAG_FILTER,
                  GL_NEAREST);
  glTexParameterf(GL_TEXTURE_2D,
                  GL_TEXTURE_MIN_FILTER,
                  GL_NEAREST);
  glTexEnvf(GL_TEXTURE_ENV,
            GL_TEXTURE_ENV_MODE,
            GL_DECAL);
  glEnable(GL_TEXTURE_2D);
  glShadeModel(GL_FLAT);
}

void display(void) {
  glClear(GL_COLOR_BUFFER_BIT|
          GL_DEPTH_BUFFER_BIT);
  glBegin(GL_QUADS);
  glTexCoord2f(0.0,0.0);
  glVertex3f(-2.0,-1.0,0.0);
```

```
  glTexCoord2f(0.0,1.0);
  glVertex3f(-2.0,1.0,0.0);
  glTexCoord2f(1.0,1.0);
  glVertex3f(0.0,1.0,0.0);
  glTexCoord2f(1.0,0.0);
  glVertex3f(0.0,-1.0,0.0);
  glTexCoord2f(0.0,0.0);
  glVertex3f(1.0,-1.0,0.0);
  glTexCoord2f(0.0,1.0);
  glVertex3f(1.0,1.0,0.0);
  glTexCoord2f(1.0,1.0);
  glVertex3f(2.41421,1.0,-1.41421);
  glTexCoord2f(1.0,0.0);
  glVertex3f(2.41421,-1.0,-1.41421);
  glEnd();
  glFlush();
}

void myReshape(int w,int h) {
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0,
                 1.0*w/(float)h,
                 1.0,30.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0,0.0,-3.6);
}

int main(int argc,char** argv) {
  auxInitDisplayMode(AUX_SINGLE|
                     AUX_RGB|
                     AUX_DEPTH);
  auxInitPosition(0,0,500,500);
  auxInitWindow(argv[0]);
  myinit();
  auxReshapeFunc(myReshape);
  auxMainLoop(display);
}
```

Commandes

(1) Spécification de la texture

- `void glTexImage2D(GLenum target,GLint level,GLint component,GLsizei l,GLsizei h,GLint b,GLenum format,GLenum type,const GLvoid *pix);`

Définit une texture 2D.

target: GL_TEXTURE_2D

level: 0 pour une texture à 1 niveau

component: nombre de composantes à gérer

1 -> rouge
2 -> R et A
3 -> R, V et B
4 -> R, V, B et A

l et h: dimension de l'image représentant la texture. l et h sont obligatoirement des puissances de 2.

b: largeur du bord (usuellement 0). Si b <> 0 alors l et h sont incrémentés de 2xb.

format et type: format et type de données de l'image (voir glReadPixels)

pix: image à placer stockée dans un tableau

- **`void glTexImage1D(GLenum target,GLint level,GLint component,GLsizei l,GLint b,GLenum format,GLenum type,const GLvoid *pix);`**

Définit une texture 1D.

target: **GL_TEXTURE_1D**

...

(2) Spécification des techniques de placage

Echantillonnage

Après transformation en coordonnées écran, les pixels de la texture (texels) correspondent rarement à des pixels de l'image (un pixel = un morceau de texel -> agrandissement, un pixel = plusieurs texels -> réduction)
-> Visualisation claire des texels s'ils sont trop grands à

l'écran
-> non visualisation des texels existant entre deux pixels s'ils sont trop petits.

OpenGL propose le choix de la ou des techniques à employer pour effectuer les calculs d'"échantillonnage" nécessaire à la détermination de la couleur d'affichage dans les cas d'agrandissement (resp. réduction) quand moins (resp. plus) d'un texel correspond à un pixel de l'image.

- **`void glTexParameter{if}{v}(GLenum target,GLenum pn, TYPE param);`**

  Configuration du placage de texture

  target: **GL_TEXTURE_2D** ou **GL_TEXTURE_1D**

  pn: paramètre sur lequel est effectuée la configuration

  param: valeur adoptée

pn	*param*
GL_TEXTURE_MAG_FILTER	**GL_NEAREST** **GL_LINEAR**
GL_TEXTURE_MIN_FILTER	**GL_NEAREST** **GL_LINEAR** ...

GL_TEXTURE_MAG_FILTER: méthode d'agrandissement

GL_TEXTURE_MIN_FILTER: méthode de réduction

GL_NEAREST: choix du texel le plus proche du centre du pixel considéré et utilisation de sa couleur
-> plus grande rapidité mais affichage de moins bonne qualité

GL_LINEAR: choix et moyennage pondéré du carré de 2x2 texels le plus proche du centre du pixel
-> rapidité moins importante mais affichage de meilleure qualité

Répétition et seuillage de texture

Les coordonnées textures sont définies dans les images sources entre les coordonnées 0 et 1. Au delà elles peuvent être répétées ou seuillées.

Répétition: toute coordonnée supérieure à 1 ou inférieure à 0 voit sa partie entière ignorée.

Seuillage: toute coordonnée supérieure à 1 (resp. inférieure à 0) est ramenée à 1 (resp. à 0).

- **`void glTexParameter{if}{v}(GLenum target,GLenum pn, TYPE param);`**

  Configuration du placage de texture

  target: **GL_TEXTURE_2D** ou **GL_TEXTURE_1D**

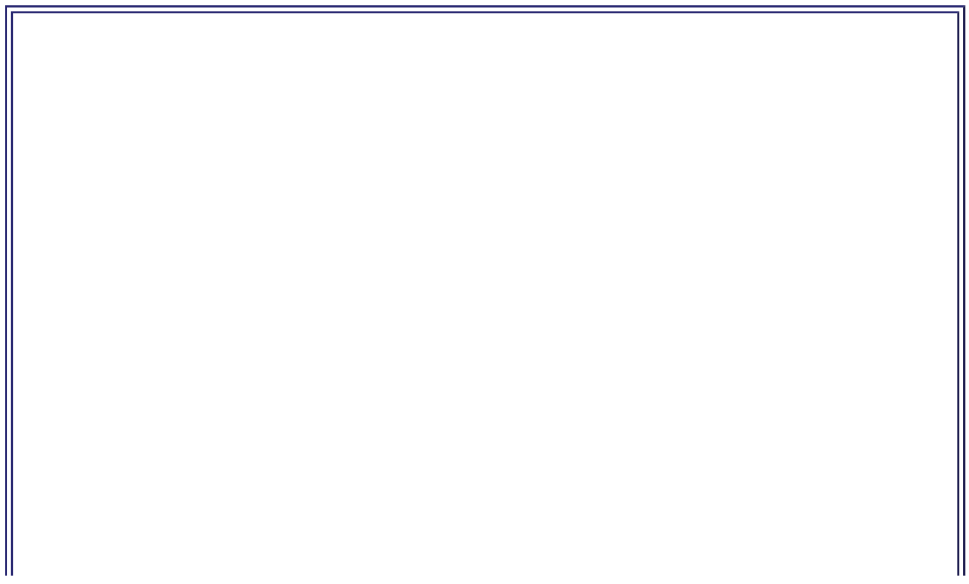
  pn: paramètre sur lequel est effectuée la configuration

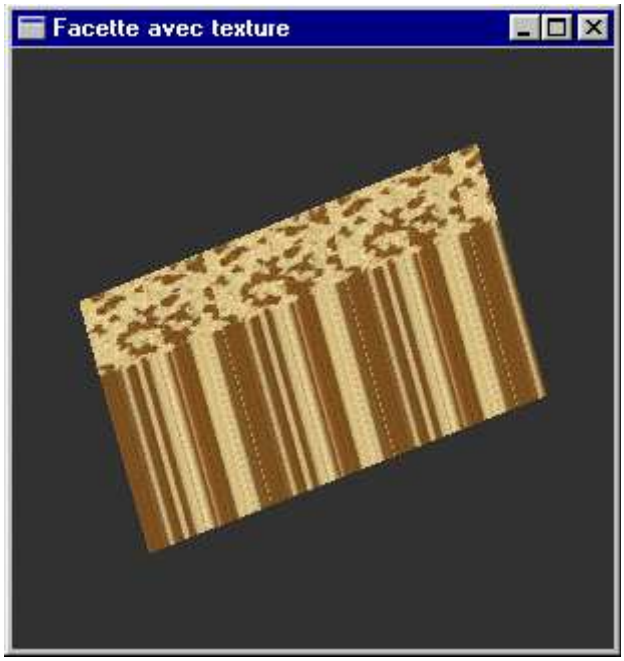
  param: valeur adoptée

pn	*param*
GL_TEXTURE_WRAP_S	GL_CLAMP GL_REPEAT
GL_TEXTURE_WRAP_T	GL_CLAMP GL_REPEAT
GL_TEXTURE_BORDER_COLOR	Quatre valeurs entre 0 et 1

- **GL_CLAMP**: seuillage

- **GL_REPEAT**: répétition





Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

(3) Autorisation du placage

- `glEnable(int mode);`

  mode: **GL_TEXTURE_2D** ou **GL_TEXTURE_1D**

(4) Coordonnées de la texture

Pour chaque sommet définissant un objet, les coordonnées lui correspondant dans la texture doivent être précisées. Les coordonnées dans la texture des pixels à l'intérieur des facettes sont obtenues par interpolation entre ces valeurs.

1, 2, 3 ou 4 coordonnées (s,t,r,q) suivant le type de texture utilisée:

- s affecté pour les textures 1D (t et r à 0, q à 1)
- s et t affectés pour les textures 2D (r à 0, q à 1)
- s, t et r affectés pour les textures 3D (q à 1)

q est l'équivalent de w en coordonnées homogènes.

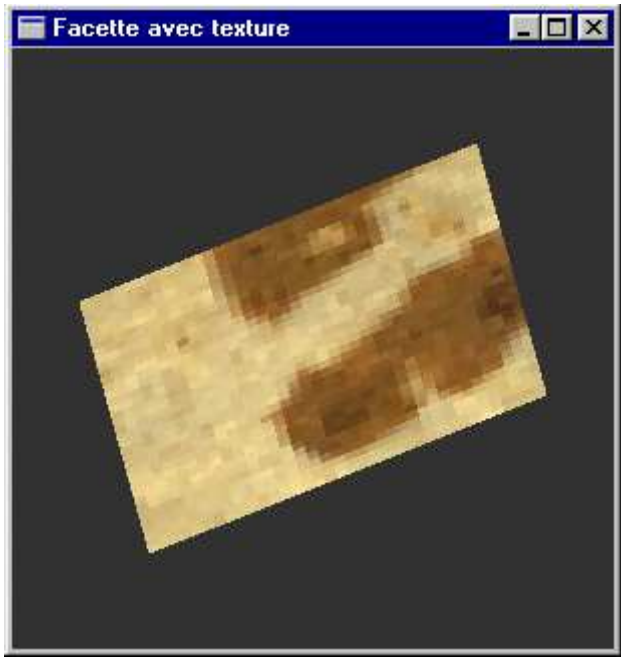
- `void glTexCoord{1234}{sifd}{v}(TYPE coords);`

  Configuration d'une position de sommet dans une texture

coords: coordonnées dans la texture

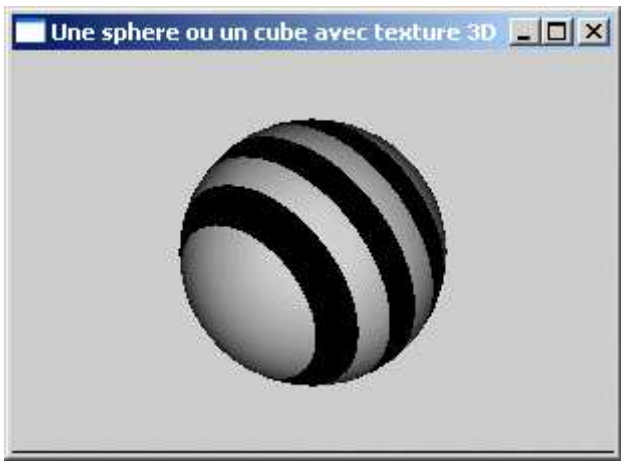
1: remplit s, t=0, r=0, q=1
2: remplit s et t, r=0, q=1
3: remplit s, t et r, q=1
4: remplit s,t,r et q

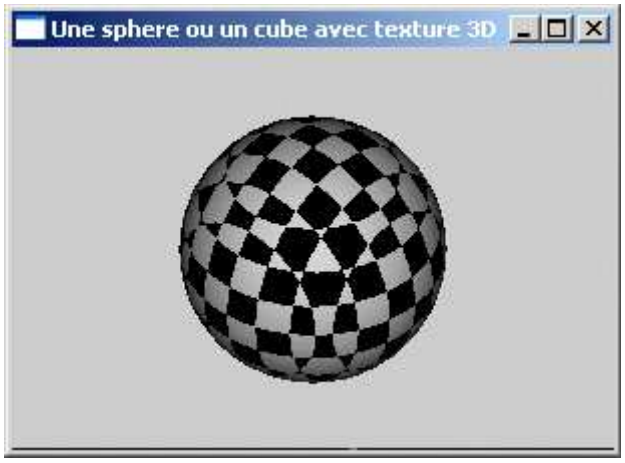
glTexCoord s'emploie comme glNormal et permet de fixer la position dans une texture du prochain sommet défini.

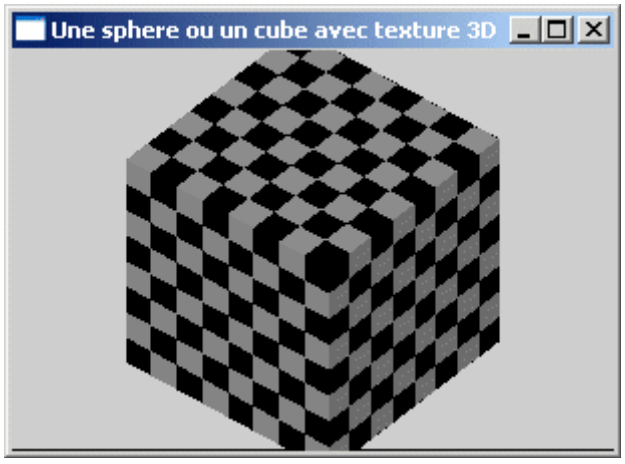


Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Depuis OpenGL 1.2, il est possible d'employer des textures 3D. Le mode opératoire et les fonctions correspondantes sont une simple généralisation des versions 1D et 2D.







Le programme GLUt

Exécutable GLUt

Génération automatique des coordonnées de texture

OpenGL permet, dans une certaine mesure, de générer automatiquement les coordonnées de texture.

Les courbes et surfaces lissées

Introduction

Modélisation et rendu de B-Splines et de surfaces basées sur les courbes de Bézier.

"évaluateurs" de courbe ou de surface de Bézier
-> spécification des points de contrôle pour une courbe ou une surface.

Pour les surfaces, calcul automatique possible des normales utilisées pour les éclairages.

Affichage réalisé par subdivision en facettes
-> précision aussi importante que désirée.

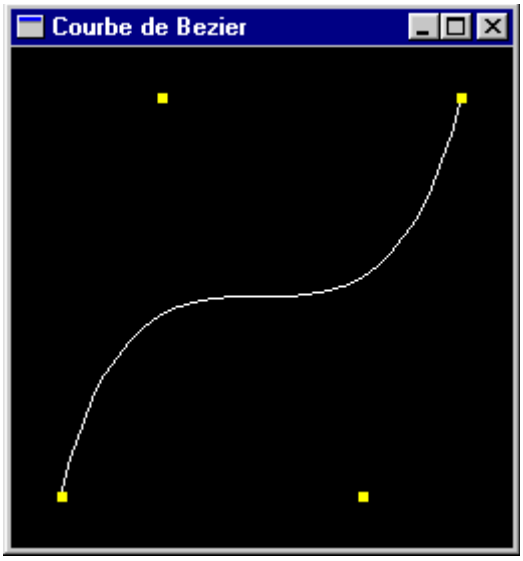
Interface pour l'utilisation des NURBS dans la librairie GLU.

Interface pour l'utilisation de certaines quadriques dans la librairie GLU.

Exemples

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

GLfloat points[4][3] = {
  {-4.0,-4.0,0.0},{-2.0,4.0,0.0},
  { 2.0,-4.0,0.0},{ 4.0,4.0,0.0}};

void display(void) {
  int i;
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0,1.0,1.0);
  glBegin(GL_LINE_STRIP);
  for( i = 0 ; i <= 30 ; i++ )
    glEvalCoord1f((GLfloat) i/30.0);
  glEnd();
  glPointSize(5.0);
  glColor3f(1.0,1.0,0.0);
  glBegin(GL_POINTS);
  for( i = 0 ; i < 4 ; i++ )
    glVertex3fv(&points[i][0]);
  glEnd();
  glFlush();
}
```

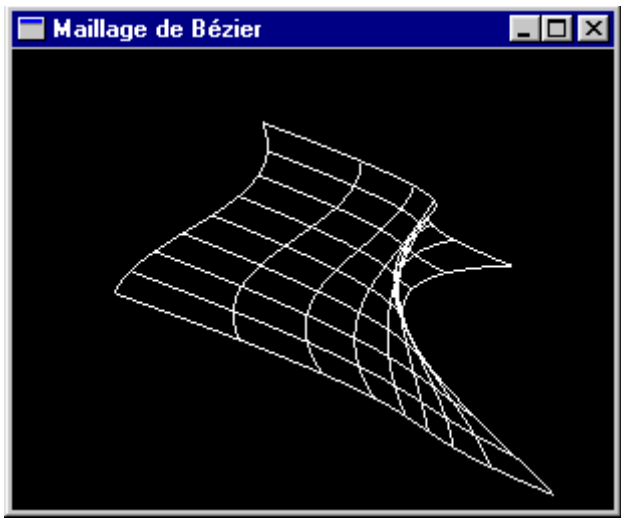
```
void myReshape(int w,int h) {
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if( w <= h )
    glOrtho(-5.,5.,-5.*h/w,5.*h/w,
            -5.,5.);
    else
    glOrtho(-5.*w/h,5.*w/h,-5.,5.,
            -5.,5.);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void myinit(void) {
  glClearColor(0.0,0.0,0.0,1.0);
  glMap1f(GL_MAP1_VERTEX_3,0.0,1.0,3,4,
          &points[0][0]);
  glEnable(GL_MAP1_VERTEX_3);
  glShadeModel(GL_FLAT);
}

int main(int argc,char** argv) {
  auxInitDisplayMode(AUX_SINGLE|
                     AUX_RGB);
  auxInitPosition(0,0,500,500);
  auxInitWindow(argv[0]);
  myinit();
  auxReshapeFunc(myReshape);
  auxMainLoop(display);
}
```



Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

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

GLfloat pts[4][4][3] = {
  { {-1.5,-1.5,4.0},{-0.5,-1.5,2.0},
    {0.5,-1.5,-1.0},{1.5,-1.5,2.0} },
  { {-1.5,-0.5,1.0},{-0.5,-0.5,3.0},
    {0.5,-0.5,0.0},{1.5,-0.5,-1.0} },
  { {-1.5,0.5,4.0},{-0.5,0.5,0.0},
    {0.5,0.5,3.0},{1.5,0.5,4.0} },
  { {-1.5,1.5,-2.0},{-0.5,1.5,-2.0},
    {0.5,1.5,0.0},{1.5,1.5,-1.0} } };

void display(void) {
  int i,j;
  glClear(GL_COLOR_BUFFER_BIT|
          GL_DEPTH_BUFFER_BIT);
  glColor3f(1.0,1.0,1.0);
  glPushMatrix();
  glRotatef(85.0,1.0,1.0,1.0);
  for( j = 0 ; j <= 8 ; j++ ) {
    glBegin(GL_LINE_STRIP);
    for( i = 0 ; i <= 30 ; i++ )
      glEvalCoord2f((float)i/30.0,
                    (float)j/8.0);
    glEnd();
    glBegin(GL_LINE_STRIP);
    for(i = 0; i <= 30; i++)
      glEvalCoord2f((float)j/8.0,
                    (float)i/30.0);
    glEnd(); }
  glPopMatrix();
  glFlush();
}

void myinit(void) {
  glClearColor(0.0,0.0,0.0,1.0);
  glMap2f(GL_MAP2_VERTEX_3,
          0,1,3,4,
          0,1,12,4,
          &pts[0][0][0]);
  glEnable(GL_MAP2_VERTEX_3);
  glMapGrid2f(20,0.0,1.0,20,0.0,1.0);
  glEnable(GL_DEPTH_TEST);
  glShadeModel(GL_FLAT);
}

void myReshape(int w,int h) {
  glViewport(0,0,w,h);
```

```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if ( w <= h )
    glOrtho(-4.0,4.0,
            -4.0*(float)h/(float)w,
             4.0*(float)h/(float)w,
            -4.0,4.0);
    else
    glOrtho(-4.0*(float)w/(float)h,
             4.0*(float)w/(float)h,
            -4.0,4.0,-4.0,4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

int main(int argc,char** argv) {
  auxInitDisplayMode(AUX_SINGLE|
                     AUX_RGB|
                     AUX_DEPTH);
  auxInitPosition(0,0,500,500);
  auxInitWindow(argv[0]);
  myinit();
  auxReshapeFunc(myReshape);
  auxMainLoop(display);
}
```



Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Commandes

Courbes

- `void glMap1{fd}(GLenum target,TYPE u1,TYPE u2,GLint stride,GLint ordre,const TYPE *p);`

Définit un évaluateur 1D.

target: ce que représentent les points de contrôle (voir ci-dessous)

u1,u2: intervalle de la variable u définissant la courbe (généralement de 0 à 1)

stride: nombre de valeurs simple ou double précision par point de contrôle dans le tableau p

ordre: degré + 1 (nombre de points de contrôle)

p: tableau des points de contrôle

target	*Valeurs possibles*
GL_MAP1_VERTEX_3	coordonnées x, y et z
GL_MAP1_VERTEX_4	coordonnées x, y, z et w
GL_MAP1_INDEX	indexe de couleurs
GL_MAP1_COLOR_4	R, V, B et A
GL_MAP1_NORMAL	coordonnées de normales
GL_MAP1_TEXTURE_COORD_1	coordonnée s de texture
GL_MAP1_TEXTURE_COORD_2	coord. s et t de texture
GL_MAP1_TEXTURE_COORD_3	coord. s, t et r de texture
GL_MAP1_TEXTURE_COORD_4	coord. s, t, r et q de texture

- `void glEvalCoord1{fd}{v}(TYPE u);`

Evalue l'évaluateur de la courbe activée.

u: valeur du paramètre d'évaluation

Surfaces

- `void glMap2{fd}(GLenum target,TYPE u1,TYPE u2,GLint us,GLint uo,TYPE v1,TYPE v2,GLint vs,GLint vo,const TYPE *p);`

Définit un évaluateur 2D.

target: ce que représentent les points de contrôle

u1,u2 et v1,v2: intervalles des variables u et v

us et vs: nombres de nombres simple ou double précision par bloc suivant u et v

uo et vo: degré + 1 en u et en v

p: tableau des points de contrôle

- **`void glEvalCoord2{fd}{v}(TYPE u,TYPE V);`**

  Evalue l'évaluateur de la surface 2D activée.

  u,v: valeurs des paramètres d'évaluation

Evaluation globale

OpenGL propose un mécanisme permettant l'utilisation automatique d'un évaluateur 1D sur l'ensemble d'un intervalle particulier avec n points générés.

Pour un évaluateur 2D, on fournit deux intervalles avec deux nombres de points à générer.

-> génèration d'une suite de points, de segments ou de polygones pouvant définir une primitive graphique.

- **`void glMapGrid1{fd}(GLint n,TYPE u1,TYPE u2);`**

  Définit une grille d'évaluation sur l'intervalle [u1,u2] avec n pas répartis régulièrement entre ces bornes.

- **`void glEvalMesh1(GLenum mode,GLint p1,GLint p2);`**

  Applique la grille d'évaluation courante à tous les évaluateurs 1D activés.

  mode: **GL_POINT** ou **GL_LINE** pour dessiner des points ou une ligne polygonale lissant les points de contrôle

  p1, p2: indices de début et de fin des points à évaluer

- `void glMapGrid2{fd}(GLint nu,TYPE u1,TYPE u2,GLint nv,TYPE v1,TYPE v2);`

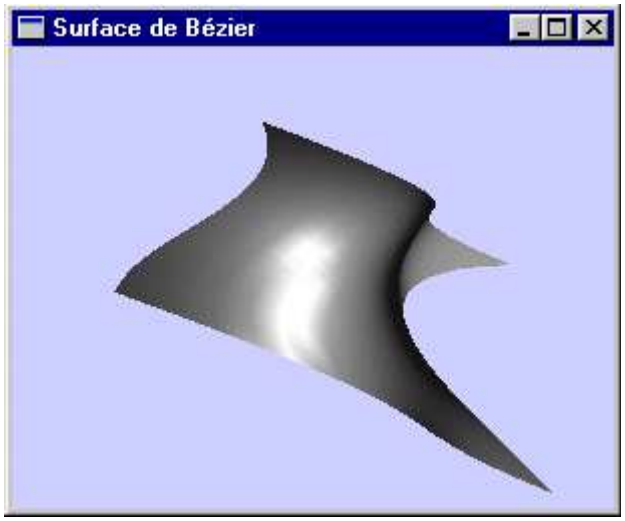
- `void glEvalMesh2(GLenum mode,GLint p1,GLint p2,GLint q1,GLint q2);`

  Définition identique à la version 1D sauf pour mode: **GL_POINT**, **GL_LINE** ou **GL_FILL** (remplissage de la surface)

  Autres exemples

```
GLfloat pts[4][4][3] = {
  { {-1.5,-1.5,4.0},{-0.5,-1.5,2.0},
    {0.5,-1.5,-1.0},{1.5,-1.5,2.0}},
  { {-1.5,-0.5,1.0},{-0.5,-0.5,3.0},
    {0.5,-0.5,0.0},{1.5,-0.5,-1.0}},
  { {-1.5,0.5,4.0},{-0.5,0.5,0.0},
    {0.5,0.5,3.0},{1.5,0.5,4.0}},
  { {-1.5,1.5,-2.0},{-0.5,1.5,-2.0},
    {0.5,1.5,0.0},{1.5,1.5,-1.0}}};

void initlights(void) {
  GLfloat amb[] = {0.2,0.2,0.2,1.0};
  GLfloat pos[] = {0.0,0.0,2.0,1.0};
  GLfloat dif[] = {0.6,0.6,0.6,1.0};
  GLfloat spec[] = {1.0,1.0,1.0,1.0};
  GLfloat shininess[] = {50.0};
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0,GL_AMBIENT,
            amb);
  glLightfv(GL_LIGHT0,GL_POSITION,
            pos);
  glMaterialfv(GL_FRONT,GL_DIFFUSE,
            dif);
  glMaterialfv(GL_FRONT,GL_SPECULAR,
               spec);
  glMaterialfv(GL_FRONT,GL_SHININESS,
               shininess);
}

void display(void) {
  glClear(GL_COLOR_BUFFER_BIT|
          GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glRotatef(85.0,1.0,1.0,1.0);
  glEvalMesh2(GL_FILL,0,20,0,20);
  glPopMatrix();
```

```
  glFlush();
}

void myinit(void) {
  glClearColor(0.0,0.0,0.0,1.0);
  glEnable(GL_DEPTH_TEST);
  glMap2f(GL_MAP2_VERTEX_3,0,1,
          3,4,0,1,12,4,&pts[0][0][0]);
  glEnable(GL_MAP2_VERTEX_3);
  glEnable(GL_AUTO_NORMAL);
  glEnable(GL_NORMALIZE);
  glMapGrid2f(20,0.0,1.0,20,0.0,1.0);
  initlights();
}
```

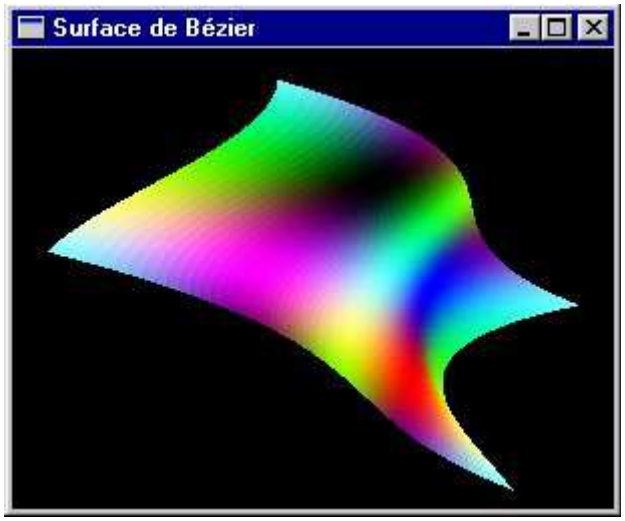


Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

```
#define iw 64
#define ih 64
GLubyte im[3*iw*ih];
GLfloat pts[4][4][3] = {
  { {-1.5,-1.5,4.0},{-0.5,-1.5,2.0},
    {0.5,-1.5,-1.0},{1.5,-1.5,2.0}},
  { {-1.5,-0.5,1.0},{-0.5,-0.5,3.0},
    {0.5,-0.5,0.0},{1.5,-0.5,-1.0}},
  { {-1.5,0.5,4.0},{-0.5,0.5,0.0},
    {0.5,0.5,3.0},{1.5,0.5,4.0}},
  { {-1.5,1.5,-2.0},{-0.5,1.5,-2.0},
    {0.5,1.5,0.0},{1.5,1.5,-1.0}}};
GLfloat texpts[2][2][2] = {
  {{0.0,0.0},{0.0,1.0}},
  {{1.0,0.0},{1.0,1.0}}};

void display(void) {
  glClear(GL_COLOR_BUFFER_BIT|
```

```
            GL_DEPTH_BUFFER_BIT);
  glColor3f(1.0,1.0,1.0);
  glEvalMesh2(GL_FILL,0,20,0,20);
  glFlush();
}

void makeImage(void) {
  int i,j;
  float ti,tj;
  for( i = 0 ; i < iw ; i++ ) {
    ti = 2.0*3.14159265*i/iw;
    for( j = 0 ; j < ih ; j++ ) {
      tj = 2.0*3.14159265*j/ih;
      im[3*(ih*i+j)]=127*(1+sin(ti));
      im[3*(ih*i+j)+1]=127*
                       (1+cos(2*tj));
      im[3*(ih*i+j)+2]=127*
                       (1+cos(ti+tj));
    }
  }
}

void myinit(void) {
  glMap2f(GL_MAP2_VERTEX_3,
          0,1,3,4,0,1,12,4,
          &pts[0][0][0]);
  glMap2f(GL_MAP2_TEXTURE_COORD_2,
          0,1,2,2,0,1,4,2,
          &texpts[0][0][0]);
  glEnable(GL_MAP2_TEXTURE_COORD_2);
  glEnable(GL_MAP2_VERTEX_3);
  glMapGrid2f(20,0.0,1.0,20,0.0,1.0);
  makeImage();
  glTexEnvf(GL_TEXTURE_ENV,
            GL_TEXTURE_ENV_MODE,
            GL_DECAL);
  glTexParameterf(GL_TEXTURE_2D,
                  GL_TEXTURE_WRAP_S,
                  GL_REPEAT);
  glTexParameterf(GL_TEXTURE_2D,
                  GL_TEXTURE_WRAP_T,
                  GL_REPEAT);
  glTexParameterf(GL_TEXTURE_2D,
                 GL_TEXTURE_MAG_FILTER,
                 GL_NEAREST);
  glTexParameterf(GL_TEXTURE_2D,
                 GL_TEXTURE_MIN_FILTER,
                 GL_NEAREST);
  glTexImage2D(GL_TEXTURE_2D,0,3,
               iw,ih,0,
```

```
                    GL_RGB,
                    GL_UNSIGNED_BYTE,
                    im);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_NORMALIZE);
    glShadeModel(GL_FLAT);
}
```

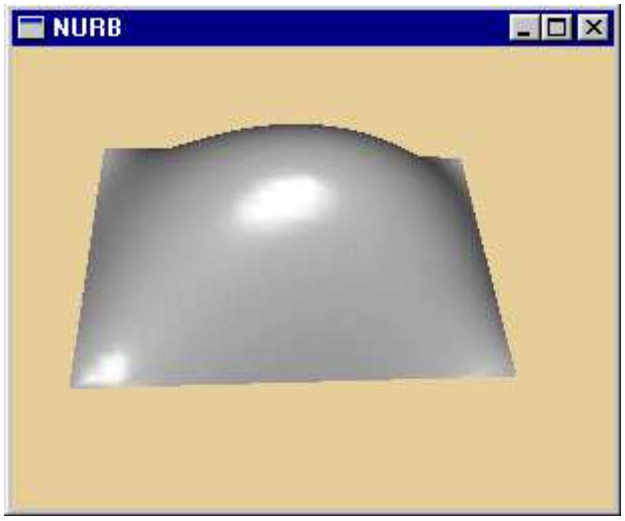


Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

L'interface GLU pour les NURBS

Introduction

La bibliothèque GLU propose un support pour les courbes et les surfaces NURBS.



Programme Aux	Programme GLUt
Exécutable Aux	Exécutable GLUt

Pour l'affichage, ces courbes et ces surfaces sont discrétisées en lignes et polygones.

Exemple

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

GLfloat ctlpts[4][4][3];
GLUnurbsObj *theNurb;

void display(void) {
  GLfloat knots[8] = {0.0,0.0,0.0,0.0,
                      1.0,1.0,1.0,1.0};
  glClear(GL_COLOR_BUFFER_BIT|
          GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glRotatef(330.0,1.,0.,0.);
  glScalef(0.5,0.5,0.5);
  gluBeginSurface(theNurb);
  gluNurbsSurface(theNurb,8,knots,
                  8,knots,4*3,3,
                  &ctlpts[0][0][0],4,4,
                  GL_MAP2_VERTEX_3);
  gluEndSurface(theNurb);
  glPopMatrix();
  glFlush();
}

void init_surface(void) {
  int u,v;
  for( u = 0 ; u < 4 ; u++ ) {
    for( v = 0 ; v < 4 ; v++ ) {
      ctlpts[u][v][0] = 2.*(u-1.5);
      ctlpts[u][v][1] = 2.*(v-1.5);
      if ((u==1 || u==2)&&
          (v==1 || v==2))
        ctlpts[u][v][2] = 3.0;
        else
        ctlpts[u][v][2] = -3.0; } }
  }

void myinit(void) {
  GLfloat dif[] = {0.7,0.7,0.7,1.0};
  GLfloat spe[] = {1.0,1.0,1.0,1.0};
  GLfloat shininess[] = {100.0};
  glClearColor(0.0,0.0,0.0,1.0);
```

```
  glMaterialfv(GL_FRONT,GL_DIFFUSE,
               dif);
  glMaterialfv(GL_FRONT,GL_SPECULAR,
               spe);
  glMaterialfv(GL_FRONT,GL_SHININESS,
               shininess);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glDepthFunc(GL_LESS);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_AUTO_NORMAL);
  glEnable(GL_NORMALIZE);
  init_surface();
  theNurb = gluNewNurbsRenderer();
  gluNurbsProperty(theNurb,
                GLU_SAMPLING_TOLERANCE,
                25.0);
  gluNurbsProperty(theNurb,
                   GLU_DISPLAY_MODE,
                   GLU_FILL);
}

void myReshape(int w,int h) {
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45.0,
                (float)w/(float)h,
                 3.0,8.0);
  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();
  auxReshapeFunc(myReshape);
  auxMainLoop(display);
}
```

Commandes

- `GLUnurbsObj *gluNewNurbsRenderer(void);`

  Crée un nouvel objet NURBS repéré par le pointeur **GLUnurbsObj*** retourné.

- `void gluNurbsProperty(GLUnurbsObj *obj,GLenum attrb, GLfloat v);`

  Contrôle les attributs de la NURBS obj.

  attrb: **GLU_SAMPLING_TOLERANCE**, **GLU_AUTO_LOAD_MATRIX**, **GLU_CULLING** ou **GLU_DISPLAY_MODE**

  v: valeur à affecter à l'attribut

attrb	*Rôle*
GLU_SAMPLING_TOLERANCE	Valeur maximale de longueur d'un segment de ligne ou de polygone
GLU_AUTO_LOAD_MATRIX	non renseigné
GLU_DISPLAY_MODE	Rendu filaire ou en polygones pleins: • **GLU_FILL** • **GLU_OUTLINE_POLYGONE** • **GLU_OUTLINE_PATCH**
GLU_CULLING	Amélioration les performances par non affichage des NURBS en dehors du volume visualisé.

- `void gluBeginSurface(GLUnurbsObj *obj);`

  Commence la création de la surface NURBS obj.

- `void gluNurbsSurface(GLUnurbsObj *obj,GLint ukc, GLfloat *ukn,GLint vkc, GLfloat *vkn,GLint us, GLint vs,GLfloat *ctl,GLint uo,GLint vo,GLenum type);`

  Décrit les sommets (ou normales ou coordonnées de texture) de la NURBS obj.

  ukc et vkc: non renseigné

  ukn et vkn: non renseigné

us: nombre de flottants entre deux points de contrôle suivant la direction paramétrique u

vs: nombre de flottants entre deux points de contrôle suivant la direction paramétrique v

ctl: tableau des points de contrôle

uo et vo: ordres de la NURBS suivant u et v (degré + 1) (nombre de sommets)

type: **GL_MAP2_VERTEX_3** ou **GL_MAP2_VERTEX_4** pour des points de contrôle à trois coordonnées ou à quatre.

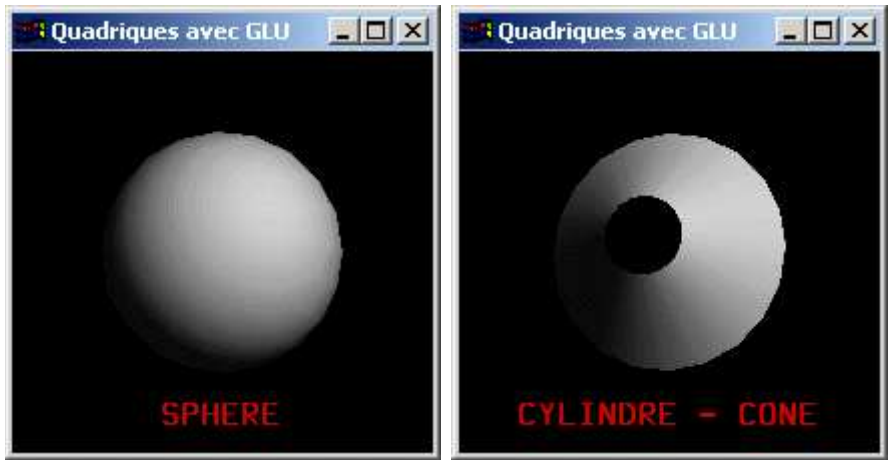
- `void gluEndSurface(GLUnurbsObj *obj);`

  Finit la définition de la surface NURBS obj.

L'interface GLU pour les quadriques

Introduction

La bibliothèque GLU propose un support pour certaines surfaces quadriques.



Programme GLUt

Exécutable GLUt

La sélection d'objets

Introduction

OpenGL propose un mécanisme permettant la sélection d'objets à l'écran.

Cette sélection peut être interactive si la zone de sélection est petite et proche de la souris.

Mode opératoire

- Dessin de la scène

- Entrée en mode "sélection"

- Second dessin de la scène (tant que l'on est en mode sélection, le frame buffer ne change pas)

- Sortie du mode sélection

- OpenGL retourne la liste des primitives qui ont intersectées le volume de visualisation au cours du second dessin.

Gestion de la liste de primitives

Le résultat d'une sélection est un tableau de "noms" entiers et de données associées. Chaque nom correspond à une ou plusieurs primitives de dessin.

Chaque primitive qui coupe la zone de visualisation génère un flag de sélection.

Le tableau de "noms" subit une mise à jour réalisant une sauvegarde de la "pile de noms" dans le tableau.

Le tableau retourné est utilisé pour déterminer quelles primitives ont été sélectionnées.

Commandes

Initialisation

- `void glSelectBuffer(GLsizei taille,GLuint *buff);`

  Définit le tableau à utiliser pour le retour des données sélectionnées.

  taille: nombre maximum de données mémorisées

  buff: tableau d'entiers non signés

- `GLint glRenderMode(GLenum mode);`

  Passe l'application en mode rendu, sélection ou feedback

  mode: **GL_RENDER**, **GL_SELECT** ou **GL_FEEDBACK**

  Valeur retournée: nombre d'objets sélectionnés si on demande **GL_SELECT** alors que l'on est déjà en mode sélection.

Création de la pile de noms

- `void glInitNames(void);`

  Vide la pile de noms.

- `void glPushName(GLuint nom);`

  Empile nom sur la pile de noms.

- `void glPopName(void);`

Dépile un nom de la pile de noms.

- `void glLoadName(GLuint nom);`

  Remplace la valeur en haut de la pile de noms par nom.

Exploitation de la pile de noms

Le tableau retourné permet l'interprétation des flags de sélection.

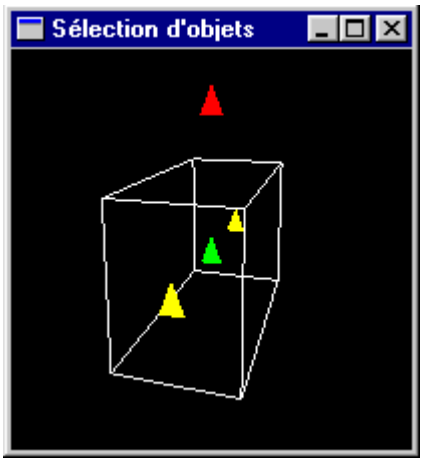
Chaque flag a généré dans ce tableau 4 enregistrements:

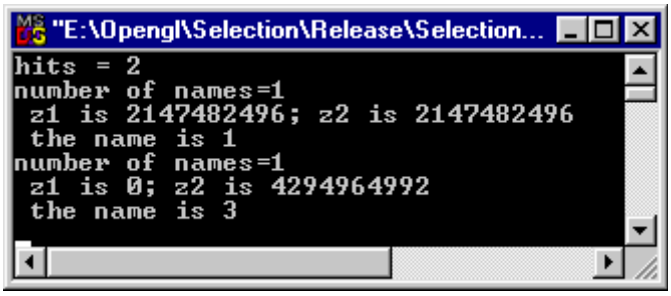
- le nombre de noms dans la pile quand le flag intervient,

- le minimum et le maximum des coordonnées écran z des sommets des primitives qui ont intersectée le volume de visualisation depuis le dernier flag (valeurs comprises entre 0 et 1 multipliées par 2^{32}-1).

- le contenu de la pile de nom au moment du flag.

Exemple





Programme Aux | Programme GLUt

Exécutable Aux | Exécutable GLUt

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

#define BUFSIZE 512

void drawTriangle(GLfloat x1,
                  GLfloat y1,
                  GLfloat x2,
                  GLfloat y2,
                  GLfloat x3,
                  GLfloat y3,
                  GLfloat z) {
  glBegin(GL_TRIANGLES);
  glVertex3f(x1,y1,z);
  glVertex3f(x2,y2,z);
  glVertex3f(x3,y3,z);
  glEnd();
}

void drawViewVol(GLfloat x1,
                 GLfloat x2,
                 GLfloat y1,
                 GLfloat y2,
                 GLfloat z1,
                 GLfloat z2) {
  glColor3f(1.0,1.0,1.0);
  glBegin(GL_LINE_LOOP);
  glVertex3f(x1,y1,-z1);
  glVertex3f(x2,y1,-z1);
  glVertex3f(x2,y2,-z1);
  glVertex3f(x1,y2,-z1);
  glEnd();
  glBegin(GL_LINE_LOOP);
  glVertex3f(x1,y1,-z2);
  glVertex3f(x2,y1,-z2);
  glVertex3f(x2,y2,-z2);
  glVertex3f(x1,y2,-z2);
  glEnd();
  glBegin(GL_LINES);
  glVertex3f(x1,y1,-z1);
  glVertex3f(x1,y1,-z2);
  glVertex3f(x1,y2,-z1);
  glVertex3f(x1,y2,-z2);
  glVertex3f(x2,y1,-z1);
  glVertex3f(x2,y1,-z2);
```

```
  glVertex3f(x2,y2,-z1);
  glVertex3f(x2,y2,-z2);
  glEnd();
}

void drawScene(void) {
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0,1.333,
                 0.01,100.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(7.5,7.5,12.5,
            2.5,2.5,-5.0,
            0.0,1.0,0.0);
  glColor3f(0.0,1.0,0.0);
  drawTriangle(2.0,2.0,3.0,
               2.0,2.5,3.0,-5.0);
  glColor3f(1.0,0.0,0.0);
  drawTriangle(2.0,7.0,3.0,
               7.0,2.5,8.0,-5.0);
  glColor3f(1.0,1.0,0.0);
  drawTriangle(2.0,2.0,3.0,
               2.0,2.5,3.0,0.0);
  drawTriangle(2.0,2.0,3.0,
               2.0,2.5,3.0,-10.);
  drawViewVol(0.0,5.0,0.0,
              5.0,0.0,10.0);
}

void processHits(GLint hits,
                 GLuint buffer[]) {
  unsigned int i,j;
  GLuint names,*ptr;
  printf("hits = %d\n",hits);
  ptr =(GLuint *) buffer;
  for ( i = 0 ; i < hits ; i++ ) {
    names = *ptr;
    printf("nb of names=%d\n",names);
    ptr++;
    printf(" z1 is %u;",*ptr);
    ptr++;
    printf(" z2 is %u\n",*ptr);
    ptr++;
    printf(" the name is ");
    for( j = 0 ; j < names ; j++ )
      printf("%d ",*ptr);
    ptr++;
    printf("\n"); }
}
```

```
void myinit(void) {
  glDepthFunc(GL_LESS);
  glEnable(GL_DEPTH_TEST);
  glShadeModel(GL_FLAT);
}

void selectObjects(void) {
  GLuint selectBuf[BUFSIZE];
  GLint hits,viewport[4];
  glSelectBuffer(BUFSIZE,selectBuf);
  glRenderMode(GL_SELECT);
  glInitNames();
  glPushName(-1);
  glPushMatrix();
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0.0,5.0,0.0,5.0,0.0,10.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glLoadName(1);
  drawTriangle(2.0,2.0,3.0,
               2.0,2.5,3.0,-5.0);
  glLoadName(2);
  drawTriangle(2.0,7.0,3.0,
               7.0,2.5,8.0,-5.0);
  glLoadName(3);
  drawTriangle(2.0,2.0,3.0,
               2.0,2.5,3.0,0.0);
  drawTriangle(2.0,2.0,3.0,
               2.0,2.5,3.0,-10.);
  glPopMatrix();
  glFlush();
  hits = glRenderMode(GL_RENDER);
  processHits(hits,selectBuf);
}

void display(void) {
  glClearColor(0.0,0.0,0.0,0.0);
  glClear(GL_COLOR_BUFFER_BIT|
          GL_DEPTH_BUFFER_BIT);
  drawScene();
  selectObjects();
  glFlush();
}

int main(int argc,char** argv) {
  auxInitDisplayMode(AUX_SINGLE|
                     AUX_RGB|
                     AUX_DEPTH);
```

```
  auxInitPosition(0,0,200,200);
  auxInitWindow(argv[0]);
  myinit();
  auxMainLoop(display);
}
```

Sélection sur une zone écran

Utilisation du mode de sélection pour déterminer si des objets sont désignés.

Utilisation d'une matrice spéciale en conjonction avec la matrice de projection pour restreindre le dessin à une région de l'écran.

Un clic de souris -> entrée en mode de sélection.

- `void gluPickMatrix(GLdouble x,GLdouble y,GLdouble l, GLdouble h,GLint viewport[4]);`

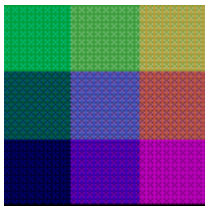
  Crée une matrice de projection qui restreint le dessin à une région de l'écran et la multiplie à la matrice courante.

  x,y: centre de la région (en coordonnées écran)

  l,h: taille de la région (en coordonnées écran)

  viewport: bords de l'écran actuel (obtenu par **glGetIntegerv (GL_VIEWPORT,GLint *viewport);**)

Exemple



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

#define BUFSIZE 512
int brd[3][3];

void myinit(void) {
  int i,j;
```

```
  for( i = 0 ; i < 3 ; i++ )
    for( j = 0 ; j < 3 ; j++ )
      brd[i][j] = 0;
      glClearColor(0.0,0.0,0.0,0.0);
}

void myReshape(int w,int h) {
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0,3.0,0.0,3.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void drawSquares(GLenum mode) {
  GLuint i,j;
  for ( i = 0 ; i < 3 ; i++ ) {
  if ( mode == GL_SELECT )
    glLoadName(i);
    for( j = 0 ; j < 3 ; j++ ) {
      if( mode == GL_SELECT )
        glPushName(j);
      glColor3f((float) i/3.0,
                (float) j/3.0,
                (float) brd[i][j]/3.0);
      glRecti(i,j,i+1,j+1);
      if ( mode == GL_SELECT )
        glPopName(); } }
}

void processHits(GLint hits,
                 GLuint buffer[]) {
  unsigned int i,j;
  GLuint ii,jj,names,*ptr;
  printf("hits = %d\n",hits);
  ptr =(GLuint *) buffer;
  for ( i = 0 ; i < hits ; i++ ) {
    names = *ptr;
    printf("nb of names=%d\n",names);
    ptr++;
    printf(" z1 is %u;",*ptr);
    ptr++;
    printf(" z2 is %u\n",*ptr);
    ptr++;
    printf(" names are ");
    for ( j = 0 ; j < names ; j++ ) {
      printf("%d ",*ptr);
    if ( j == 0 )
      ii = *ptr;
```

```
      else
      if( j == 1 )
    jj = *ptr;
    ptr++; }
  printf("\n");
  board[ii][jj] =(brd[ii][jj]+1)%3; }
}

void display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  drawSquares(GL_RENDER);
  glFlush();
}

void pickSquares(AUX_EVENTREC *event) {
  GLuint selectBuf[BUFSIZE];
  GLint hits;
  GLint viewport[4];
  int x,y;
  x = event->data[AUX_MOUSEX];
  y = event->data[AUX_MOUSEY];
  glGetIntegerv(GL_VIEWPORT,viewport);
  glSelectBuffer(BUFSIZE,selectBuf);
  glRenderMode(GL_SELECT);
  glInitNames();
  glPushName(-1);
  glMatrixMode(GL_PROJECTION);
  glPushMatrix();
  glLoadIdentity();
  gluPickMatrix((float) x,
                (float)(viewport[3]-y),
                5.0,5.0,viewport);
  gluOrtho2D(0.0,3.0,0.0,3.0);
  drawSquares(GL_SELECT);
  glMatrixMode(GL_PROJECTION);
  glPopMatrix();
  glFlush();
  hits = glRenderMode(GL_RENDER);
  processHits(hits,selectBuf);
}

int main(int argc,char** argv) {
  auxInitDisplayMode(AUX_SINGLE|
                     AUX_RGB);
  auxInitPosition(0,0,100,100);
  auxInitWindow(argv[0]);
  myinit();
  auxMouseFunc(AUX_LEFTBUTTON,
               AUX_MOUSEDOWN,
               pickSquares);
```

```
    auxReshapeFunc(myReshape);
    auxMainLoop(display);
}
```