Cours 2

Franck HÉTROY

3A IRV, 16 novembre 2005

Rappel planning

- 05/10/2005 (1h30) : cours
  ⇝ Introduction à OpenGL
- 16/11/2005 (1h30) : cours
  ⇝ Fonctionnalités OpenGL avancées
- 23/11/2005 (1h30) : T.P.
  ⇝ Introduction à libQGLViewer et petits exercices
- 07/12/2005 (3h) : T.P.
- 04/01/2005 (3h) : T.P.

Aujourd'hui

1. Blending, antialiasing et autres effets
2. *Display lists*
3. Plaquage de textures
4. Les buffers OpenGL
5. Sélection d'objets

Livre de chevet

D. Shreiner, M. Woo, J. Neider, T. Davis

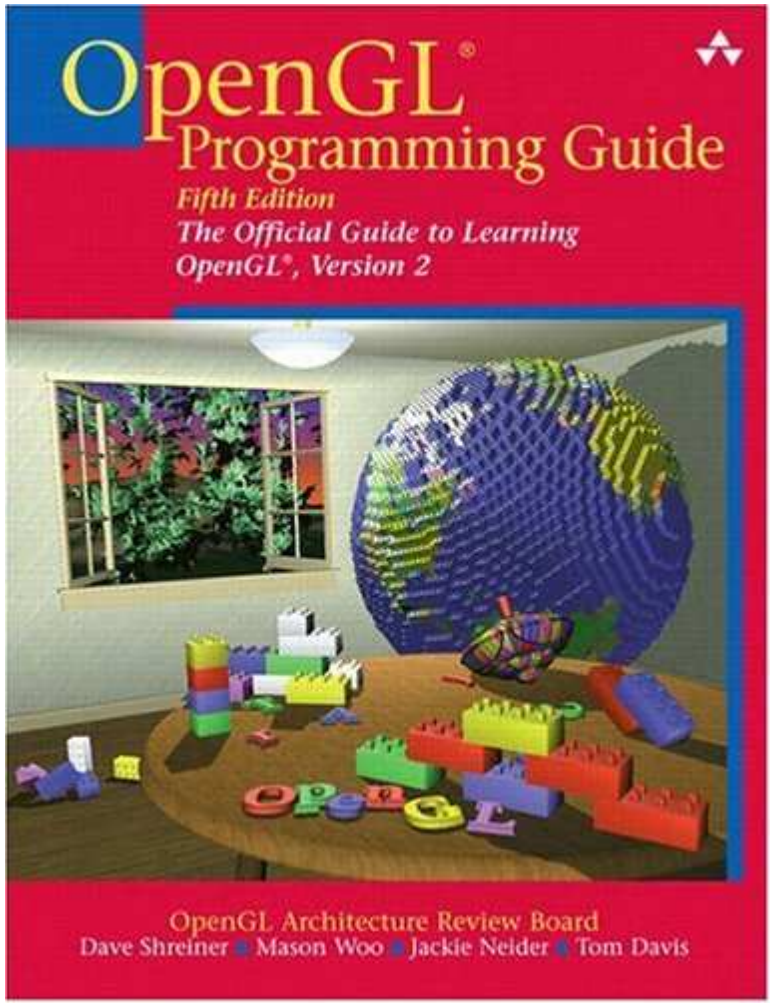
OpenGL® Programming Guide Fifth edition

Addison-Wesley

alias le red book

`http://opengl-redbook.com/`

⇝ toujours le même



OpenGL - Cours 1

Blending

- Blending = fusion, mélange
  $\rightsquigarrow$ de couleurs
- Objectif : obtenir des effets de transparence en combinant les couleurs d'une source et d'une destination
- Les effets obtenus dépendront de la manière de combiner les couleurs : plusieurs fonctions de blending possibles
- Utilisation du canal alpha
- Pas possible en mode color-index (il n'a pas d'info de composante alpha)
- Existe depuis OpenGL 1.2 ; nouvelles possibilités depuis OpenGL 1.4

Blending : principe

- Combinaison de la couleur du pixel traité (source) avec la valeur du pixel déjà présent dans le framebuffer (destination)
- Intervient juste avant l'affichage du framebuffer à l'écran

Example

Vue d'un objet à travers un verre vert, qui transmet 80% de la lumière (opacité = 20%) :
couleur à afficher
= 20% couleur du verre et 80% couleur de l'objet
= 20% nouvelle couleur + 80% ancienne couleur
= $\alpha_{new} RGBA_{new} + (1.0 - \alpha_{new}) RGBA_{old}$

Blending : en pratique

- En fait on peut faire plus général : couleur = $(a_1 R_{new} + a_2 R_{old}, b_1 G_{new} + b_2 G_{old}, c_1 B_{new} + c_2 B_{old}, d_1 \alpha_{new} + d_2 \alpha_{old})$
- 2 commandes possibles :
  - `glBlendFunc(srcFactor, destFactor)`
  - `glBlendFuncSeparate(srcRGB, destRGB, srcAlpha, destAlpha)`
- Possibilités pour les paramètres : `GL_ZERO`, `GL_ONE`, `GL_SRC_ALPHA`, `GL_ONE_MINUS_SRC_ALPHA`, `GL_CONSTANT_COLOR`, `GL_CONSTANT_ALPHA`, ...
- Dans les deux derniers cas, spécifier la constante avec `glBlendColor(cstRed,cstGreen,cstBlue,cstAlpha)`
- Ne pas oublier `glEnable(GL_BLEND)` !

Fonctions de blending

- On peut faire plus compliqué qu'une simple addition de coefficients :
  - `GL_FUNC_ADD` : $k_1 RGBA_{new} + k_2 RGBA_{old}$
  - `GL_FUNC_SUBTRACT` : $k_1 RGBA_{new} - k_2 RGBA_{old}$
  - `GL_FUNC_REVERSE_SUBTRACT` : $k_2 RGBA_{old} - k_1 RGBA_{new}$
  - `GL_MIN` : $\min(k_1 RGBA_{new}, k_2 RGBA_{old})$
  - `GL_MAX` : $\max(k_1 RGBA_{new}, k_2 RGBA_{old})$
  - `GL_LOGIC_OP` : $RGBA_{new}$ op $RGBA_{old}$
- A utiliser dans `glBlendEquation(mode)` ou `glBlendEquationSeparate(modeRGB, modeAlpha)`

Exemples d'utilisation du blending

Example (Mélange de deux images à proportions égales)

1. `glBlendFunc(GL_ONE,GL_ZERO);`
2. Dessiner la première image.
3. `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);`
4. Mettre α à 0.5 (par exemple avec `glColor4f()`) et dessiner la seconde image.

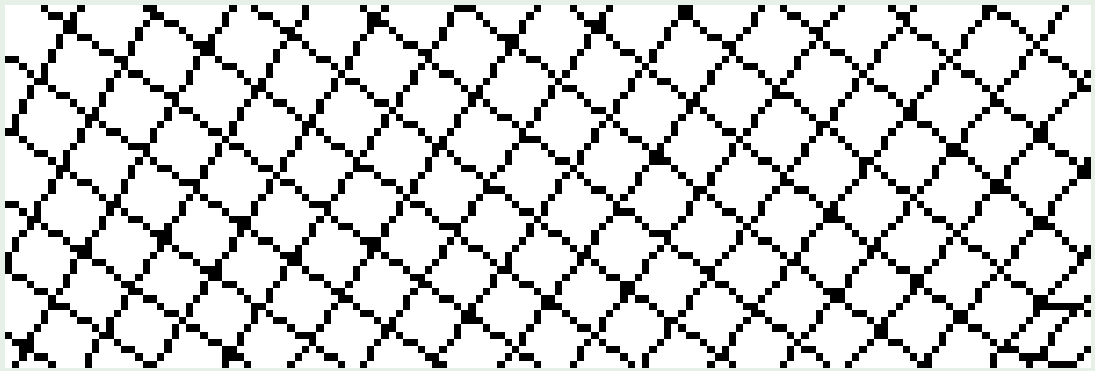
Example (Programme de dessin)

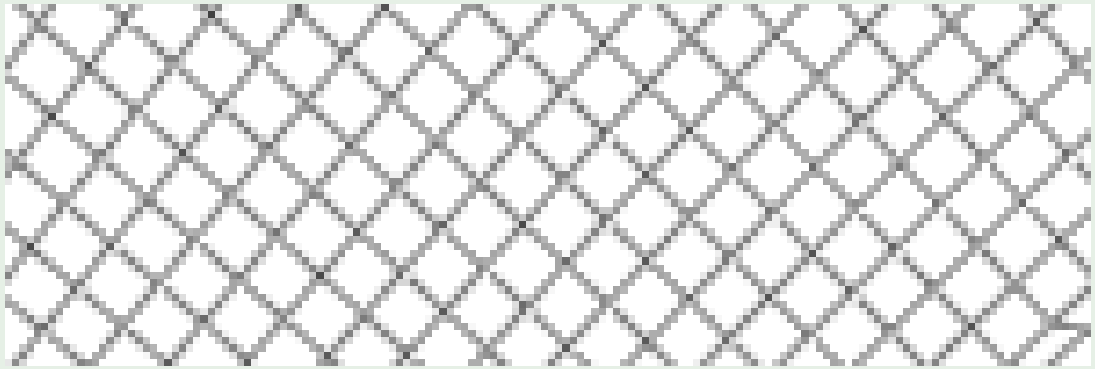
Pour que la brosse ajoute à chaque fois un peu de couleur à l'image :

1. `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);`
2. Mettre α à 0.1 par ex. et dessiner l'image de la brosse.

Antialiasing

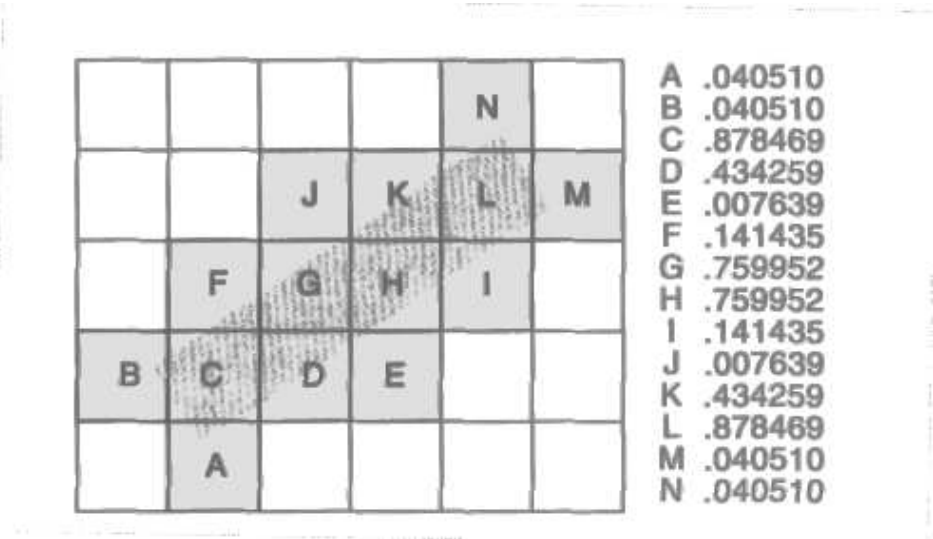
Example





Antialiasing : principe

- Calcul d'une valeur c de couverture pour chaque pixel traversé par la ligne
  $\rightsquigarrow$ $c =$ pourcentage de l'aire du pixel traversé par la ligne
- Multiplication de la valeur de la composante alpha par c
- Blend du pixel avec la couleur précédemment stockée dans le framebuffer

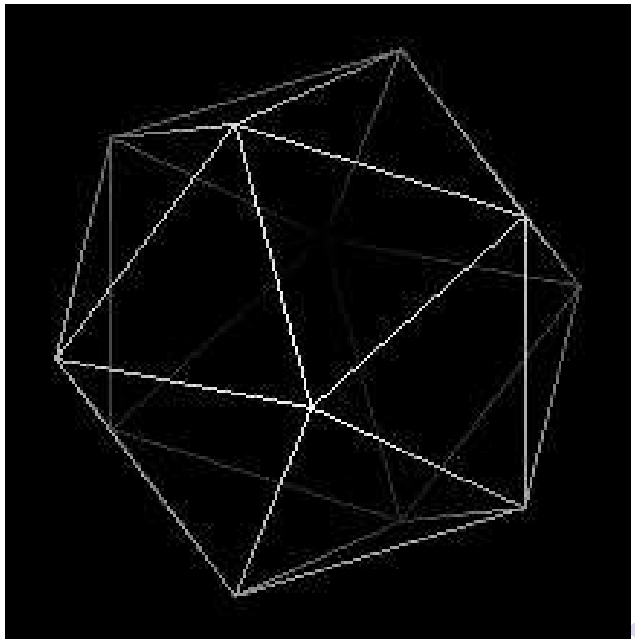


Antialiasing : en pratique

- Problème : le calcul des valeurs de couverture n'est pas simple
- Quelques solutions :
  1. utiliser `glEnable(GL_LINE_SMOOTH)` ou `glEnable(GL_POINT_SMOOTH)`
     ⇝ aucun contrôle sur la qualité du résultat
  2. ajouter `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)` (ne pas oublier `glEnable(GL_BLEND)`) et éventuellement jouer sur `glLineWidth(myWidth)` et `glPointSize(mySize)`
  3. faire du multisampling (décomposition des calculs pour un pixel en calculs plus simples pour des morceaux de pixels)
- D'autres existent (utilisation de textures, du buffer d'accumulation, . . .)

Brouillard (*fog*) : principe

- Plus un objet est loin de la caméra, plus sa couleur prend celle du brouillard
- Permet de simuler le brouillard, mais aussi la fumée, la pollution, la vapeur, la buée, ...
- Depth cu(e)ing (points et lignes) : réduction de l'intensité d'un objet en fonction de la distance à l'observateur
  $\rightsquigarrow$ même principe



Brouillard : réglages

- Mélange de la couleur du brouillard et de la couleur de l'objet selon un facteur f :
  $RGBA = fRGBA_{objet} + (1 - f)RGBA_{fog}$
- f dépend de la profondeur z de l'objet dans la scène et peut prendre trois formes :

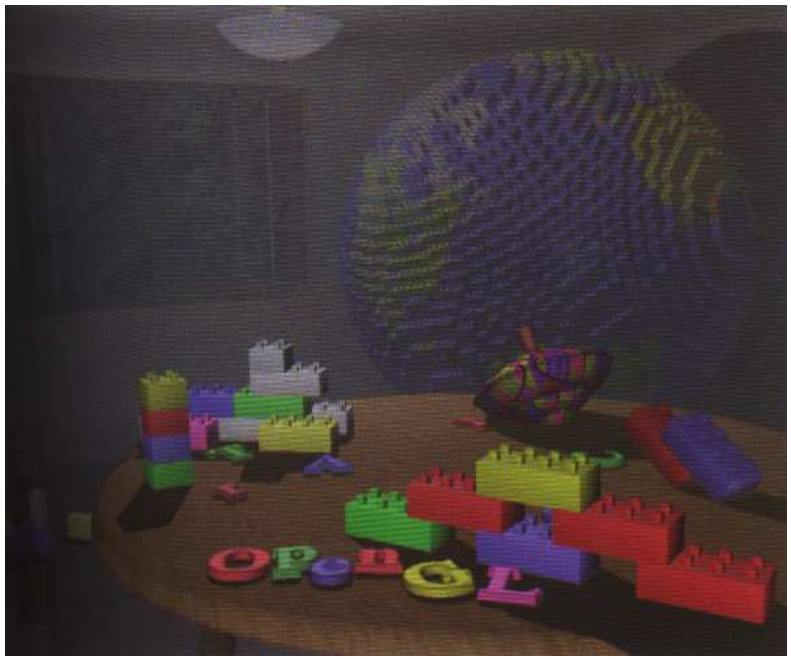
$f = e^{-density.z}$	`GL_EXP`
$f = e^{-(density.z)^2}$	`GL_EXP2`
$f = \frac{end-z}{end-start}$	`GL_LINEAR`

- *density*, *start* et *end* à fixer dans les variables `GL_FOG_DENSITY`, `GL_FOG_START` et `GL_FOG_END`
- Aussi possible en mode Color Index
- On peut aussi changer la direction de z avec `glFogCoord(...)`

Brouillard : en pratique

Example

```
glEnable(GL_FOG);
GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0 };
glFogi(GL_FOG_MODE, GL_EXP);
glFogfv(GL_FOG_COLOR,fogColor);
glFogf(GL_FOG_DENSITY,0.35);
```



Paramètres de points

- Objectif : afficher des sphères de différentes tailles (par ex. pour simuler des gouttes de pluie)
- `glPointSize()` est une première solution mais peu de contrôle
- `glPointParameterf(pname,param)` permet d'avoir plus de contrôle
- pname : `GL_POINT_DISTANCE_ATTENUATION`, `GL_POINT_SIZE_MIN`, `GL_POINT_SIZE_MAX`, ...
- Si `GL_POINT_DISTANCE_ATTENUATION`, $taille_{new} = \frac{taille_{old}}{\sqrt{a+b.z+c.z^2}}$, $(a, b, c) =$ param

OpenGL - Cours 1

Philosophie

- Display list = ensemble de commandes OpenGL stockées pour être exécutées plus tard
- Permet d'améliorer les performances : on stocke une seule fois des commandes qui peuvent être exécutées souvent (affichage, changements d'états)
  $\rightsquigarrow$ utilisation en réseau : display lists stockées sur le serveur
- Attention : certaines commandes (notamment celles modifiant ou dépendant de l'état du client) ne peuvent pas être stockées dans une display list et sont exécutées immédiatement

Création d'une display list

- Une liste est identifiée par un entier : son indice
- On génère des indices par `glGenLists(range)` qui renvoie le premier indice d'une suite de range indices libérés pour identifier des listes
- Une display list comprend toutes les routines OpenGL comprises entre un `glNewList(listIndex,GL_COMPILE)` et `glEndList()`
- `GL_COMPILE` peut être remplacé par `GL_COMPILE_AND_EXECUTE` si on veut que les routines soient exécutées en plus d'être stockées dans la display list

Gestion d'une display list

- Exécution avec `glCallList(listIndex)`
- Commandes exécutées dans l'ordre où elles ont été stockées
- Display lists hiérarchiques : appel d'une display lis dans la création d'une autre
- Pour exécuter *n* display lists successivement :
  `glCallLists(n,index_type,ptr_index)`
  $\rightsquigarrow$ appeler `glListBase(firstListIndex)` lors de la création des listes pour indiquer l'offset à ajouter à ptr_index

Display lists multiples : exemple

Example

```
void init() {
  GLuint base;
  base = glGenLists(128);
  glListBase(base);
  glNewList(base+'1',GL_COMPILE);
    drawLetter(data1);
  glEndList();
  ...
  glNewList(base+'Z',GL_COMPILE);
    drawLetter(dataZ);
  glEndList(); }

void printString(GLByte *s) {
  GLint len = strlen(s);
  glCallLists(len,GL_BYTE,s); }
```

OpenGL - Cours 1

Plaquage de textures (*texture mapping*)

Très général, beaucoup de possibilités :

- Texture = tableau 2D de données (couleur, luminance, . . .) appelées texels
- Plusieurs formats de texture possibles (RGBA, profondeur, luminance, intensité)
- Plusieurs modes de plaquages possibles (remplacer, mélanger, moduler)
- Rotations/translations avant plaquage possibles
- Sur ou sous-échantillonnage
- Une texture peut se répéter ou non sur l'objet
- . . .

Plaquage de textures : processus

1. Générer ou charger une texture depuis un fichier bitmap
2. Créer un texture object et lui spécifier une texture
3. Indiquer commer la texture doit être appliquée à chaque pixel :
   - Remplacer la couleur par celle de la texture ;
   - Mélanger la couleur avec celle de la texture ;
   - Moduler la couleur selon celle de la texture.
4. Activer le plaquage de textures :
   `glEnable(GL_TEXTURE_2D)` (ou `GL_TEXTURE_1D`, `GL_TEXTURE_3D` ou `GL_TEXTURE_CUBE_MAP`)
5. Dessiner la scène, en indiquant comment la texture va “s’attacher” à l’objet
   ⇝ coordonnées géométriques et coordonnées de texture

Chargement d'une texture

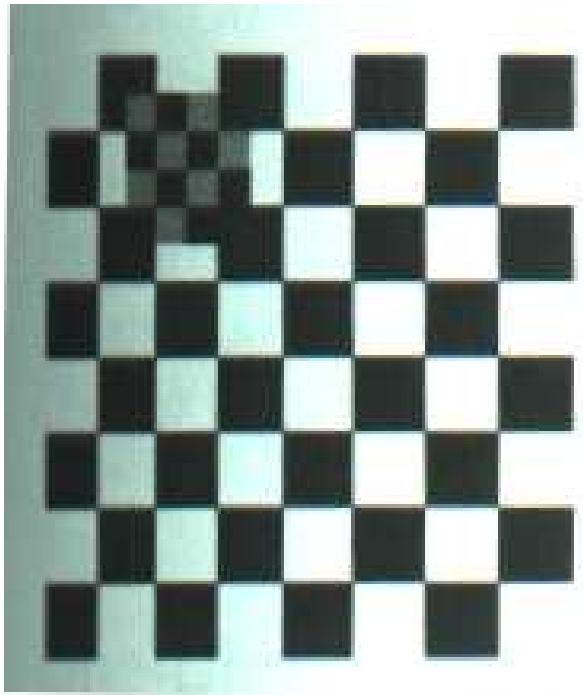
- `glTexImage2D(target,level,internalFormat, width,height,border,format,type,ptr_texels)` pour une texture 2D
- level : si on fait une texture multirésolution (mipmap)
- internalFormat : composants sélectionnés pour les texels (RGBA, profondeur, luminance, intensité)
- border : 1 ou 0 suivant si la texture a un bord ou pas
- On peut aussi lire la texture du framebuffer : `glCopyTexImage2D(target,level,internalFormat, x,y,width,height,border)` ((x, y) coin bas gauche du rectangle définissant la texture)
- Textures 1D et 3D : `glTexImage1D(...)`, `glTexImage3D(...)`

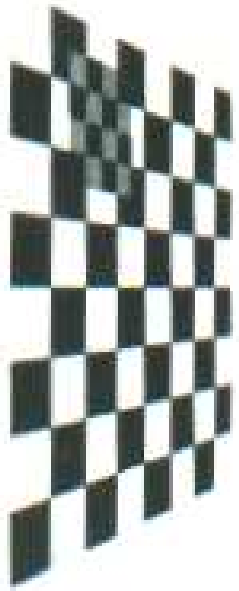
Remplacement d'une partie de la texture

On peut remplacer une partie de la texture par une autre texture :

`glTexSubImage2D(target,level,xOffset,yOffset, width,height,format,type,ptr_texels)`

$\rightsquigarrow$ évite de créer complètement une nouvelle texture





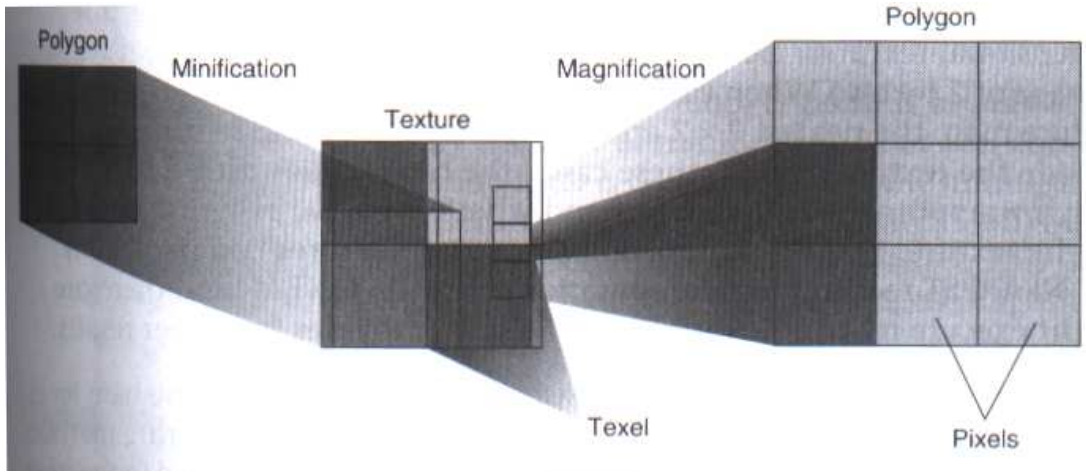
Création d'un *texture object*

- Un texture object évite de devoir recharger une image de texture plusieurs fois : il stocke les données d'une texture en mémoire
- Associer un nom (un entier) à *n* texture objects :
  `glGenTextures(n,ptr_texNames)`
- Créer (bind = attacher – à un nom) et utiliser un texture object :
  `glBindTexture(GL_TEXTURE_2D,texName)`
  $\rightsquigarrow$ ensuite, utiliser les procédures de chargement d'une texture vues auparavant
- Supprimer *n* textures objects :
  `glDeleteTextures(n,ptr_texNames)`

Filtrage de textures

Comment appliquer la texture à chaque pixel :

- Si un pixel correspond à une portion d'un texel, magnification (sur-échantillonnage)
  ⇒ `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_NEAREST)`
- Si un pixel correspond à plusieurs texels, minification (sous-échantillonnage)
  ⇒ `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_LINEAR)`



Génération de texture : exemple

Example

```
GLubyte Texture[16] =
{
0,0,0,0, 0xFF,0xFF,0xFF,0xFF,                                   //Image (2x2)
0xFF,0xFF,0xFF,0xFF, 0,0,0,0
};
GLuint Nom;

void InitGL()
{
   glClearColor(.5,.5,.5,0);                                    //Change la couleur du fond
   glEnable(GL_DEPTH_TEST);                                     //Active le depth test
   glEnable(GL_TEXTURE_2D);                                     //Active le texturing
   glGenTextures(1,&Nom);                                       //Génère un n° de texture
   glBindTexture(GL_TEXTURE_2D,Nom);                            //Sélectionne ce n°
   glTexImage2D (
           GL_TEXTURE_2D,                                       //Type : texture 2D
           0,                                                   //Mipmap : aucun
           4,                                                   //Couleurs : 4
           2,                                                   //Largeur : 2
           2,                                                   //Hauteur : 2
           0,                                                   //Largeur du bord : 0
           GL_RGBA,                                             //Format : RGBA
           GL_UNSIGNED_BYTE,                                    //Type des couleurs
           Texture                                              //Addresse de l'image
   );
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
}
```

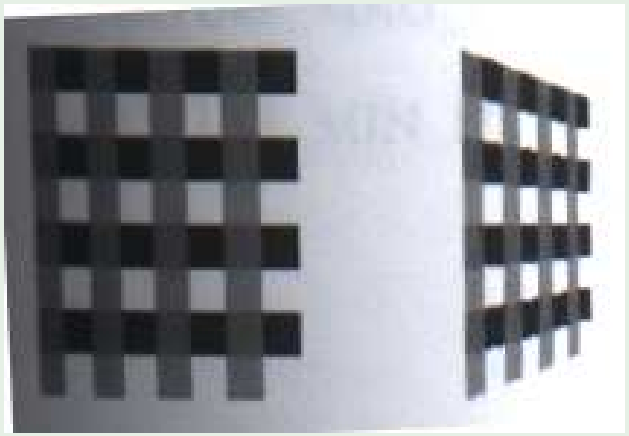
Fonctions de texture

- Ça c'était pour remplacer la couleur d'un pixel par celle de la texture
- On peut aussi utiliser la texture pour moduler la couleur, ou mélanger la couleur et celle de la texture
- Fonction de texture :
  `glTexEnv(target,pname,param)`
- Général et compliqué ; cf. une doc pour les détails
- Une utilisation simple : le mélange
  - `glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,param)`
  - param = `GL_DECAL`, `GL_REPLACE`, `GL_MODULATE`, `GL_BLEND`, `GL_ADD` ou `GL_COMBINE`

Cordonnées de texture

- Pour chaque sommet de la scène, préciser ses coordonnées (s, t) de texture
  $\rightsquigarrow$ texture 2D vue comme un carré de largeur et hauteur 1
- `glTexCoord2f(coords)`
- On peut répéter une texture dans une direction avec
  `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,GL_REPEAT)`

Example

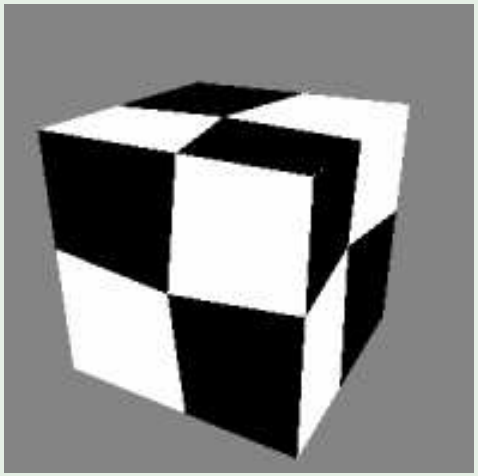


glHint()

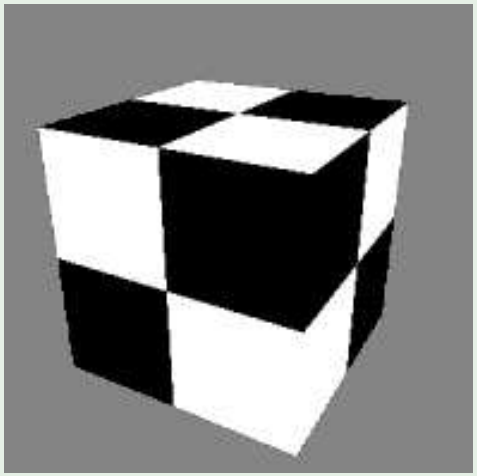
- `glHint(target,hint)` contrôle certains comportements d'OpenGL
- hint = `GL_FASTEST`, `GL_NICEST` ou `GL_DONT_CARE`
- target = `GL_LINE_SMOOTH_HINT` (antialiasing), `GL_FOG_HINT`, `GL_TEXTURE_COMPRESSION_HINT`, ...

Example

```
glHint(GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST)
```



Sans



Avec

OpenGL - Cours 1

Les buffers OpenGL

Quatre buffers forment le framebuffer :

- Color buffers
  ⇝ de 1 à 4 suivant que le système est double bufferisé et qu'il supporte les images stéréoscopiques
  + éventuellement des buffers auxiliaires
  ⇝ `glGetBoolean(GL_DOUBLEBUFFER);`
  `glGetBoolean(GL_STEREO);`
- Depth/Z buffer
  ⇒ suppression des parties cachées
- Stencil buffer
  ⇝ pour ne dessiner qu'une partie de l'image
  ⇒ simulateur de conduite
- Accumulation buffer
  ⇝ pour "accumuler" des images
  ⇒ antialiasing, motion blur

Motion blur

Utilisation d'un buffer

Initialisation :

- `glClearColor(red,green,blue,alpha)` (RGBA) ou `glClearIndex(red,green,blue,alpha)` (color index)
- `glClearDepth(depth)`
- `glClearStencil(s)` avec s un entier
- `glClearAccum(red,green,blue,alpha)`

+ ensuite `glClear(mask)` avec mask = `GL_COLOR_BUFFER_BIT`, `GL_DEPTH_BUFFER_BIT`, `GL_STENCIL_BUFFER_BIT` ou `GL_ACCUM_BUFFER_BIT`

Sélection d'un color buffer :

- `glDrawBuffer(mode)` ou `glDrawBuffers(n,ptr_buffers)`
- `glReadBuffer(mode)`
- mode = `GL_FRONT`, `GL_BACK`, `GL_LEFT`, `GL_FRONT_LEFT`, ...

Opérations sur les pixels avec les buffers OpenGL

- Accepter un pixel selon sa valeur alpha :
  `glAlphaFunc(func,ref)`
- Accepter un pixel selon sa profondeur :
  `glDepthFunc(func)`
- Définir une portion rectangulaire pour le stencil :
  `glScissor(x,y,width,height)`
- Accepter un pixel selon sa valeur par rapport au stencil :
  `glStencilFunc(func,ref,mask)`
- Contrôler l'accumulation buffer :
  `glAccum(op,value)`
  avec op = `GL_ACCUM`, `GL_LOAD`, `GL_RETURN`, `GL_ADD` ou `GL_MULT`

OpenGL - Cours 1

Sélection d'objets

- Une scène 3D, c'est encore mieux quand c'est interactif
- Deux modes :
  - Sélection : l'utilisateur choisit un ou plusieurs objets affichés à l'écran
  - Feedback : pour récupérer les infos d'affichage (par exemple pour l'impression ou l'export vers un fichier)
- Dans ces deux modes, les infos d'affichage ne seront pas envoyées au framebuffer : l'affichage est gelé tant que l'on est dans un de ces modes

Pile des noms

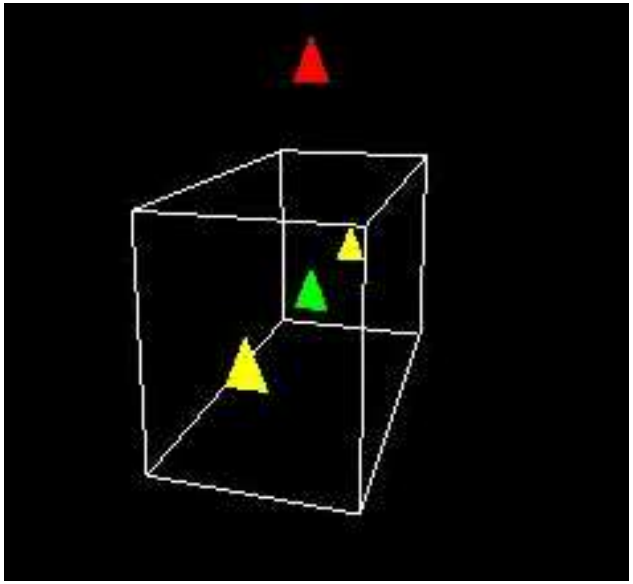
- A chaque primitive sélectionnable est associé un nom, dont la valeur est un entier
- Ces entiers sont gérés dans une pile
- Initialisation de la pile des noms avec `glInitNames()`
- Association d'un nom à un objet sélectionnable : empiler le nom avec `glPushName(i)`, dessiner l'objet, puis dépiler le nom avec `glPopName(i)`
- Autre possibilité : suite de `glLoadName(i)` qui remplace l'élément en haut de la pile des noms par i

Sélection d'objets : processus

1. Spécifier le buffer de sélection :
   `glSelectBuffer(bufMaxSize,bufName)`
2. Dessiner la scène
3. Passer en mode sélection : `glRenderMode(GL_SELECT)`
4. Initialiser la pile des noms : `glInitNames()`
5. Définir le volume de visualisation pour la sélection (ne contient pas forcément toute la scène), avec notamment `glPushMatrix()`
6. Redessiner la scène et associer son nom à chaque objet sélectionnable
7. Sortir du mode sélection et renvoyer les données de sélection :
   `glPopMatrix()`
   `hits = glRenderMode(GL_RENDER)`

Hits

- Un hit lorsqu'une primitive à laquelle est associée un nom intersecte le volume de visualisation
- Hit stocké dans le buffer de sélection
- Un hit contient notamment tous les noms présents dans la pile au moment de la sélection
  ⇝ on peut sélectionner plusieurs objets à la fois



Feedback

- Idem sélection : affichage gelé, données envoyées à un buffer
- Le buffer de feedback contient des infos sur les primitives dessinées : type (point, ligne, polygone, ...), sommets, couleur, ...
- Ces infos sont stockées sous forme de flottants
- Spécification du buffer avec
  `glFeedBackBuffer(bufSize,bufType,bufname)`
  ⇝ bufType dépend du type de primitives affichées :
  `GL_2D`, `GL_3D`, `GL_3D_COLOR`, ...
- Passage en mode feedback avec
  `glRenderMode(GL_FEEDBACK)`
- La taille du buffer est donnée lorsqu'on sort du mode feedback, avec `size = glRenderMode(GL_RENDER)`

Pour en savoir plus

OpenGL :

- Le red book : http://www.opengl-redbook.com/, http://fly.cc.fer.hr/~unreal/theredbook/
- http://www.opengl.org/

Autour d'OpenGL :

- GLUT : http://www.opengl.org/resources/libraries/glut.html
- Qt : http://www.trolltech.com/
- libQGLViewer : http://artis.imag.fr/Members/Gilles.Debunne/QGLViewer/

Tutoriaux en ligne

- Antoine Bouthors :
http://evasion.imag.fr/Membres/Antoine.Bouthors/teaching/opengl/
- Martin Beaudet :
http://eraquila.iquebec.com/site/delphi/opengl/glmenu.htm
- Nehe :
http://nehe.gamedev.net/
- Nate Robins :
http://www.xmission.com/~nate/tutors.html
- SGI :
http://www.sgi.com/products/software/opengl/examples/index.html

Fin

C'est tout pour aujourd'hui !