OpenGL – Notions avancées

David Odin

Septembre 2007

OpenGL – Notions avancées – Plan

OpenGL – Notions avancées – Plan

Textures

OpenGL peut utiliser différents types de texture :

- Monodimensionnelles (GL_TEXTURE_1D), utilisé dans des cas très spécifiques, comme le cell shading.

Textures

OpenGL peut utiliser différents types de texture :

- Monodimensionnelles (GL_TEXTURE_1D), utilisé dans des cas très spécifiques, comme le cell shading.
- Bidimensionnelles, (GL_TEXTURE_2D) est le cas le plus courant, appelé plaquage d'image,

Textures

OpenGL peut utiliser différents types de texture :

- Monodimensionnelles (GL_TEXTURE_1D), utilisé dans des cas très spécifiques, comme le cell shading.
- Bidimensionnelles, (GL_TEXTURE_2D) est le cas le plus courant, appelé plaquage d'image,
- Tridimensionnelles, (GL_TEXTURE_3D), permet de simuler de la 3D volumique, mais très gourmand en mémoire.

COORDONNÉES DE TEXTURES

De la même façon que l'on peut donner une couleur en chaque point d'une primitive graphique, il est possible de donner (en général à la place) une coordonnée de texture.
On utilise pour cela les fonctions du genre : glTexCoord*(). Par exemple, pour une texture 2D, on pourra spécifier le coin supérieur gauche d'une image à l'aide de l'appel suivant :

- glTexCoord2f(0.0, 0.0);

Ces coordonnées peuvent éventuellement être modifiées par la matrice de texture courante que l'on sélectionnera avec l'appel suivant :

- glMatrixMode(GL_TEXTURE);

Par exemple, l'animation suivante est effectuée en ne faisant qu'une translation dans la matrice de texture : animation suivante

CHARGEMENT D'UNE TEXTURE

Avant de pouvoir utiliser une texture, il faut la charger en mémoire vidéo, et positionner pas mal de paramètres.
On commence par réserver un descripteur de texture :

- GLuint texName;
- glGenTextures (1, &texName);

puis, on sélectionne cette texture comme texture 2D courante :

- glBindTexture(GL_TEXTURE_2D, texName);

Et l'on ajuste certains paramètres :

```
glTexParameteri (GL_TEXTURE_2D,
                 GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri (GL_TEXTURE_2D,
                 GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

STOCKAGE DANS LA MÉMOIRE VIDÉO

Pour envoyer une image de texture dans la mémoire vidéo, on peut utiliser la fonction suivante :

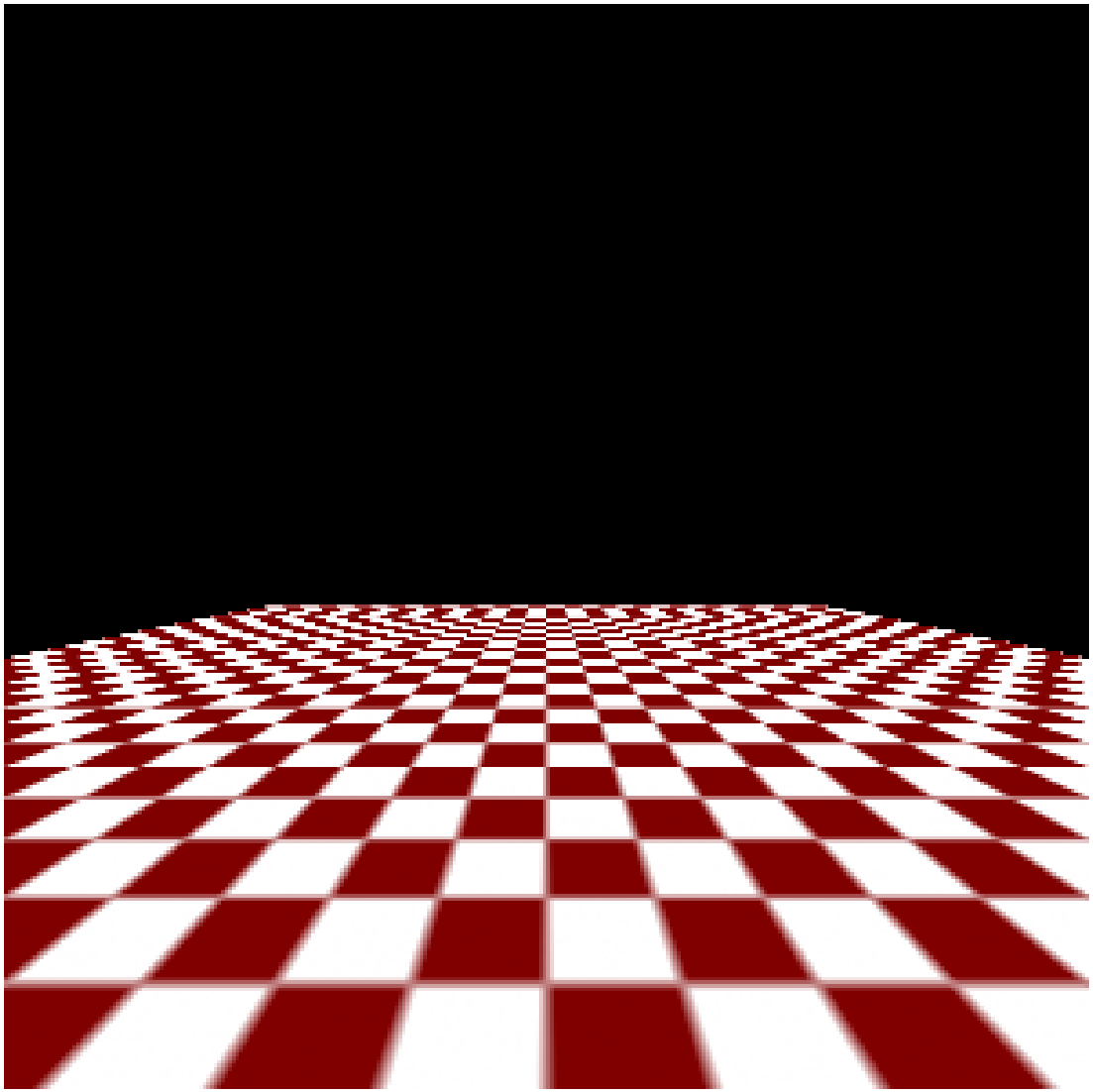
```
glTexImage2D (GL_TEXTURE_2D,     /* le type de texture      */
              0,                 /* le niveau (mipmapping) */
              GL_RGBA,           /* format de stockage      */
              largeur,           /* largeur en pixels       */
              hauteur,           /* hauteur en pixels       */
              0,                 /* le bord de la texture   */
              GL_RGBA,           /* format des donnees      */
              GL_UNSIGNED_BYTE,  /* format d'un canal       */
              checkImage         /* adresse des donnees     */
             );
```

Une fois tout ceci fait, on peut utiliser cette texture 2D, en pensant bien à utiliser un : `glEnable (GL_TEXTURE_2D);` et un :

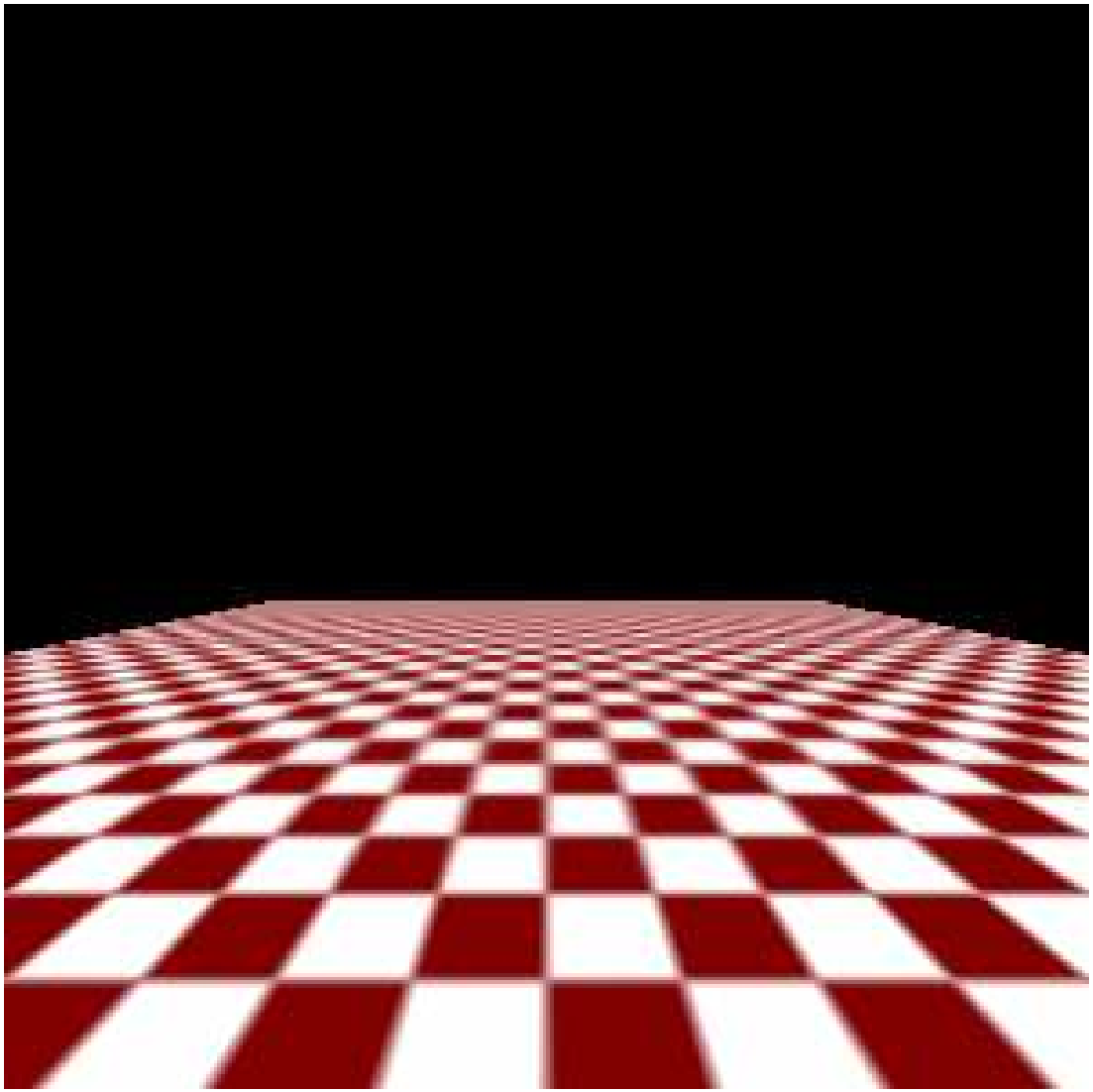
```
glTexEnv (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
                          GL_REPLACE);
```

ALIASING

Les perspectives peuvent être source de problèmes, comme l'aliasing :

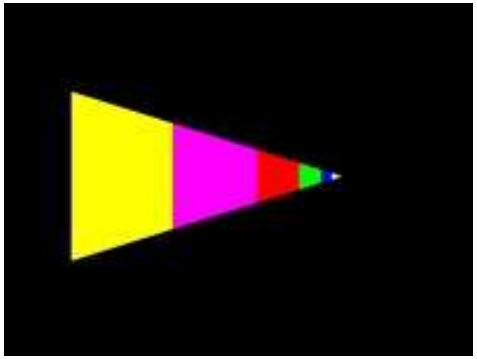


Une solution peut-être le mipmapping :

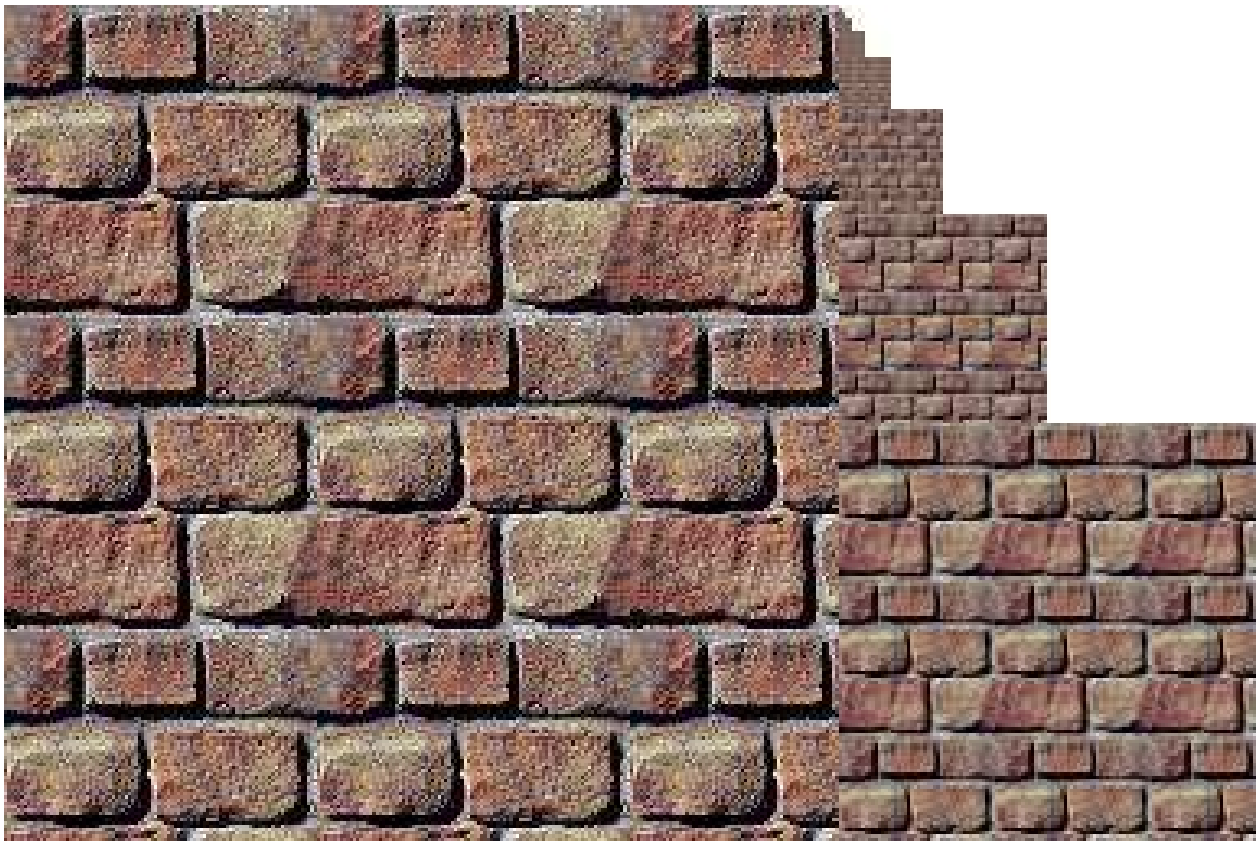


Implémentation du Mipmapping

OpenGL utilise alors une texture différente suivant la taille de ce qu'il doit représenter. L'exemple suivant montre un triangle en perspective rempli avec une texture "mipmap" dont chaque résolution à une couleur différente.



En pratique, on utilise plutôt une version réduite de l'image pour les différentes tailles :



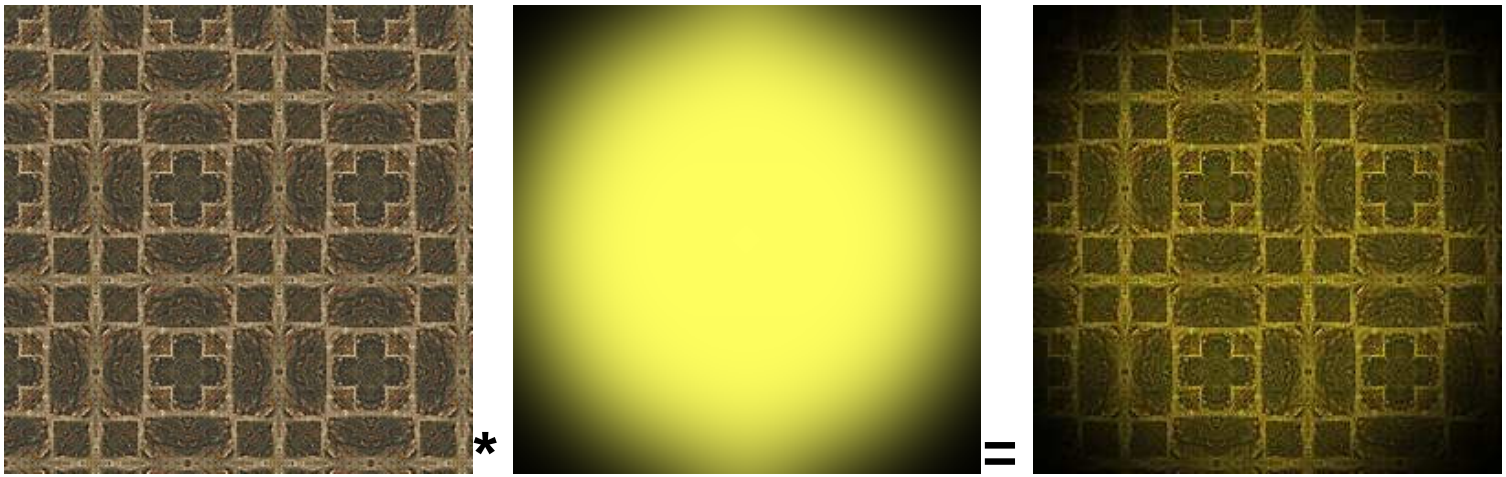
CRÉATION DE MIPMAPS

Pour faciliter la création des différentes textures, on pourra utiliser la fonction suivante :

```
gluBuild2DMipMaps (GL_TEXTURE_2D,    /* le type de texture     */
                   GL_RGBA,          /* format de stockage     */
                   largeur,          /* largeur en pixels      */
                   hauteur,          /* hauteur en pixels      */
                   GL_RGBA,          /* format des donnees     */
                   GL_UNSIGNED_BYTE, /* format d'un canal      */
                   checkImage        /* adresse des donnees    */
                  );
```

Cette fonction s'utilise comme glTexImage2D() mais prend en charge la création des différentes images du mipmap.

Exemple d'utilisation du blending : le multitexturage :



- Question : Quelle combinaison de f_s et f_d ai-je utilisée ?

Liste d'affichage

Les listes d'affichage (display lists) permettent de stocker des commandes OpenGL pour pouvoir les utiliser plusieurs fois. Ces listes sont référencées par leur numéro. On crée un (ou plusieurs) numéro de liste avec la fonction suivante :

- GLuint glGenLists (GLsizei range);

Ensuite on peut créer les listes proprement dites à l'aide de la fonction :

- **void** glNewList (GLuint list , GLenum mode);

`list` est le numéro de la liste que l'on crée et `mode` peut prendre l'une des deux valeurs suivantes :

- GL_COMPILE
- GL_COMPILE_AND_EXECUTE

On place ensuite les appels OpenGL que l'on désire placer dans la liste, puis on termine la liste en appelant la fonction :

- **void** glEndList (**void**);

Exécution d'une liste

on pourra appeler une liste (ce qui exécutera l'ensemble des appels de fonctions stockées dans cette liste) avec la fonction :

- **void** glCallList (GLuint list);

Le paramètre `list` est évidemment le numéro de la liste que l'on veut appeler.
Il est impossible de modifier une liste, mais on peut la détruire (par exemple pour la recréer ensuite) avec la fonction suivante :

- **void** glDeleteLists (GLuint list , GLsizei range);

Note : on ne peut pas créer une liste à l'intérieur d'une autre, mais on peut appeler une liste à l'intérieur d'une autre.

Tableaux de sommets

OpenGL “sait” gérer 6 types de tableaux :

- Coordonnées de Sommets,
- Couleurs,
- Coordonnées de Normales,
- Coordonnées de Texture,
- Coordonnées d’index de couleurs,
- Indicateurs de contour de polygônes

ACTIVATION DES TABLEAUX

On demande à OpenGL de gérer un tableau à l'aide de la fonction suivante :

- **void** glEnableClientState (GLenum type);

Le paramètre `type` indique quel tableau on désire activer, il peut prendre les valeurs suivantes :

- GL_VERTEX_ARRAY
- GL_COLOR_ARRAY
- GL_NORMAL_ARRAY
- GL_TEXTURE_COORD_ARRAY
- GL_INDEX_ARRAY
- GL_EDGE_FLAG_ARRAY

CONTENU DES TABLEAUX

Suivant le type de tableau, on indiquera où se trouve le contenu de ces tableaux à l'aide de différentes fonctions :

- **void** glVertexPointer (GLint taille , GLenum type, GLsizei decalage, **void** ∗debut);
- **void** glColorPointer (GLint taille , GLenum type, GLsizei decalage, **void** ∗debut);
- **void** glNormalPointer (GLenum type, GLsizei stride, **void** ∗debut);
- **void** glTexCoordPointer (GLint taille , GLenum type, GLsizei decalage, **void** ∗debut);
- **void** glIndexPointer (GLint taille , GLenum type, GLsizei decalage, **void** ∗debut);
- **void** glEdgeFlagPointer (GLsizei decalage, **void** ∗debut);

Pour chacune de ces fonctions, "debut" est un pointeur sur le premier élément du tableau, "taille" est le nombre de coordonnées (2 pour 2D, 3 pour 3D, 3 pour RGB, etc.), "type" est le type des données du tableau (GL_FLOAT, GL_INT, etc.) et "decalage" est le nombre d'octets entre deux éléments, s'il n'y a pas de "trou" dans le tableau, on utilisera 0 pour "decalage".

UTILISATION DES TABLEAUX

Maintenant que OpenGL sait quels tableaux il doit utiliser, on va pouvoir simplifier les appels entre glBegin() et glEnd(). Par exemple, si les tableaux des sommets, des couleurs et des textures sont activés, l'appel glArrayElement(i); permettra de remplacer un appel à glColor*, un appel à glNormal* et un appel à glVertex*.
On peut aussi se passer du couple glBegin/glEnd si on dispose d'un nouveau tableau contenant la liste des indices des éléments que l'on veut à l'aide de la fonction suivante :

- glDrawElements(GLenum mode, GLSizei nombre, GLenum type, **void** *tableau_indices

Mais si les éléments que l'on veut dessiner sont consécutifs, on utilisera la fonction suivante :

- glDrawArrays(GLenum mode, GLint premier, GLsizei nombre);