

Les textures



par Jeff Molofee (NeHe) ([Autres Articles](#)) Traduit par Fiquet

Date de publication : 23/03/2007

Dernière mise à jour : 23/03/2007

Nous allons voir à quel point l'application de texture peut s'avérer avantageuse. Imaginons que vous vouliez faire voler un missile d'un bout à l'autre de l'écran. Avant ce tutoriel, nous aurions probablement créé le missile entièrement en polygones, avec des couleurs fantaisistes. Grâce à l'application de texture, vous pouvez prendre une vraie photo d'un missile, et faire "voler" l'image d'un côté à l'autre de l'écran. D'après vous lequel donnera le meilleur résultat ? Une photographie ou un objet formé de triangles et de carrés ? En utilisant l'application de texture, cela ne va pas seulement rendre mieux, mais votre programme tournera plus vite. La texture affichée du missile sera seulement un quadrilatère se déplaçant sur l'écran. Un missile formé de polygones pourrait être fait de centaines ou de milliers de polygones. Une seule texture affichée nécessitera alors beaucoup moins de puissance de calcul.

- 0 - Contributions
- 1 - Tutoriel
 - 1.1 - Inclusions et déclarations
 - 1.2 - La fonction LoadBMP
 - 1.3 - La fonction LoadGLTextures
 - 1.4 - La fonction InitGL
 - 1.5 - La fonction DrawGLScene
 - 1.6 - Conclusion
- 2 - Téléchargements
- 3 - Remerciements
- 4 - Liens

0 - Contributions

Remarque : la traduction des autres tutoriels est en cours mais votre aide serait appréciée ! Si vous voulez aider, n'hésitez pas à envoyer un **message privé** ou un mail vers **nehe arrobase redaction-developpez point com**.

1 - Tutoriel

1.1 - Inclusions et déclarations

Commençons par ajouter cinq nouvelles lignes de code en-haut de la leçon une. La première nouvelle ligne est `#include <stdio.h>`. L'ajout de cet en-tête nous permettra de travailler avec les fichiers. Nous devons inclure cette ligne afin d'utiliser la fonction `fopen()` plus tard dans le code. Ensuite nous ajoutons trois nouvelles variables flottantes : `xrot`, `yrot` et `zrot`. Ces variables seront utilisées pour faire tourner le cube sur l'axe des X, l'axe des Y et l'axe des Z. La dernière ligne `GLuint texture[1]` met de côté l'espace mémoire pour une texture. Si vous voulez charger plus d'une texture, vous pouvez changer l'index 1 selon le nombre de textures que vous souhaitez charger.

En-têtes et déclarations

```
#include <windows.h>           // En-tete pour Windows
#include <stdio.h>             // En-tete pour les entrees/sorties standards (NOUVEAU)
#include <gl\gl.h>             // En-tete pour la bibliotheque OpenGL32
#include <gl\glu.h>           // En-tete pour la bibliotheque GLu32
#include <gl\glaux.h>         // En-tete pour la bibliotheque GLaux

HDC      hdc=NULL;           // Contexte prive du systeme GDI
HGLRC    hRC=NULL;          // Contexte de rendu permanent
HWND     hWnd=NULL;         // Contient notre descripteur de fenetre
HINSTANCE hInstance;        // Sauve l'instance de l'application
bool     keys[256];         // Tableau utiliser pour les routines du clavier
bool     active=TRUE;       // Drapeau d'activation de la fenetre
bool     fullscreen=TRUE;   // Drapeau pour le mode plein ecran
GLfloat  xrot;              // Rotation en X (NOUVEAU)
GLfloat  yrot;              // Rotation en Y (NOUVEAU)
GLfloat  zrot;              // Rotation en Z (NOUVEAU)
GLuint   texture[1];        // Stockage d'une texture (NOUVEAU)
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); // Declaration de WndProc
```

1.2 - La fonction LoadBMP

Maintenant juste après le code ci-dessus, et avant le `ReSizeGLScene()`, nous allons ajouter la section de code suivante. Le but de ce code est de charger un fichier bitmap. Si le fichier n'existe pas, `NULL` est retourné signifiant que la texture n'a pas pu être ouverte. Avant de commencer à expliquer le code, il y a quelques petites choses importantes que vous devez savoir sur les images que vous prévoyez d'utiliser comme textures. La hauteur et la largeur de l'image doivent être une puissance de deux. La largeur et la hauteur doivent être d'au moins 64 pixels, et pour des raisons de compatibilité, ne devraient pas être plus grandes que 256 pixels. Si l'image que vous voulez utiliser n'a pas 64, 128 ou 256 pixels en hauteur et en largeur, redimensionnez-la avec un programme de retouche d'image. Il existe des moyens pour contourner cette limitation, mais pour le moment nous nous collerons seulement aux tailles de texture standards.

Voici l'image que nous allons charger :



L'image qui sera la texture

La première chose que nous faisons est de créer un descripteur de fichier. Un descripteur est une valeur utilisée pour identifier une ressource, de manière à ce que notre programme puisse y accéder. Pour commencer nous initialisons le descripteur à la valeur NULL.

Fonction de chargement d'un BMP

```
AUX_RGBImageRec *LoadBMP(char *Filename) // Charge une image bitmap
{
    FILE *File=NULL; // Descripteur de fichier
```

Nous vérifions ensuite qu'un nom de fichier a vraiment été passé en paramètre. La personne peut avoir utilisé LoadBMP() sans spécifier de fichier à charger, nous sommes donc obligés de le contrôler. Nous n'allons pas essayer de charger rien du tout :).

Test du nom de fichier

```
if (!Filename) // Verifie qu'un nom de fichier a ete donne
{
    return NULL; // Si non, retourne NULL
}
```

Si un nom de fichier a été donné, il faut maintenant vérifier qu'il existe bien. La ligne ci-dessous tente d'ouvrir le fichier.

Test d'existence du fichier

```
File=fopen(Filename,"r"); // Controle si le fichier existe
```

Si nous n'arrivons pas à ouvrir le fichier, c'est qu'il n'existe probablement pas. Nous fermons le fichier avec fclose(File) puis nous retournons les données de l'image. auxDIBImageLoad(Filename) lit les données.

Renvoi des données du bitmap

```
if (File) // Est-ce que le fichier existe ?
{
    fclose(File); // Ferme le descripteur
```

Renvoi des données du bitmap

```
return auxDIBImageLoad(Filename); // Ouvre le bitmap et renvoie un pointeur  
}
```

Si nous n'avons pas pu ouvrir le fichier, nous renvoyons NULL. Cela signifie que le fichier n'a pas pu être ouvert. Plus tard dans le programme nous vérifierons que le fichier a bien été chargé. S'il ne l'était pas, nous quitterions le programme avec un message d'erreur.

Retour en cas d'échec

```
return NULL; // Retourne NULL si le chargement a echoue  
}
```

1.3 - La fonction LoadGLTextures

Ceci est la partie du code qui permet d'ouvrir un bitmap (en appelant le code ci-dessus) et qui le convertit en texture.

Fonction de création de la texture

```
int LoadGLTextures() // Ouvre un bitmap et le convertit en texture  
{
```

Nous définissons une variable appelée Status. Nous utiliserons cette variable pour savoir si nous avons pu charger le bitmap et créer une texture. Nous définissons par défaut Status à FALSE, ce qui signifie que rien n'a été ouvert ou créé.

Variable de statut

```
int Status=FALSE; // Indicateur du statut
```

Maintenant nous créons une structure d'image pour y stocker notre bitmap. L'enregistrement va garder la taille, la largeur, et les données du bitmap.

Structureur d'image

```
AUX_RGBImageRec *TextureImage[1]; // Cree une structure d'image
```

Nous vidons cette structure juste pour être sûr qu'elle est vide

Initialisation par défaut

```
memset(TextureImage,0,sizeof(void *)*1); // Initialise le pointeur à NULL
```

Maintenant nous chargeons le bitmap et le convertissons en texture. TextureImage[0]=LoadBMP("Data/NeHe.bmp") va solliciter le code de LoadBMP(). Le fichier Nehe.bmp du répertoire Data sera chargé. Si tout va bien, les données de l'image seront stockées dans TextureImage[0], Status sera mis à TRUE, et nous pourrons commencer à construire notre texture.

Remplissage avec les données du BMP

```
// Charge le bitmpap, controle les erreurs et quitte si le bitmap n'a pas ete trouve  
if (TextureImage[0]=LoadBMP("Data/NeHe.bmp"))  
{  
    Status=TRUE; // Met la variable Status à TRUE
```

Maintenant que nous avons chargé les données de l'image dans TextureImage[0], nous pouvons créer une texture utilisant ces données. La première ligne glGenTextures(1, &texture[0]) dit à OpenGL que nous voulons générer un seul nom de texture (augmentez le nombre si vous voulez charger plus d'une texture). Rappelez-vous qu'au tout début de ce tutoriel nous avons créé la place pour une texture avec la ligne de code GLuint texture[1]. Bien que

vous pensiez que la première texture serait stockée dans `&texture[1]` au lieu de `&texture[0]`, ce n'est pas le cas. La première zone de stockage réelle est 0. Si nous voulions deux textures nous devrions utiliser `GLuint texture[2]` et la seconde texture serait stockée dans `texture[1]`.

La seconde ligne `glBindTexture(GL_TEXTURE_2D, texture[0])` dit à OpenGL d'assimiler la texture nommée `texture[0]` à une cible de texture. Les textures 2D ont une hauteur (sur l'axe des Y) et une largeur (sur l'axe des X). La fonction principale de `glBindTexture` est d'assigner un nom de texture aux données de cette texture. Dans ce cas nous disons à OpenGL que la mémoire est disponible dans `&texture[0]`. Quand nous créons la texture, elle sera stockée dans la mémoire référencée dans `&texture[0]`.

Génération de la texture

```
glGenTextures(1, &texture[0]); // Cree la texture
// Génération typique de texture utilisant les donnees du bitmap
glBindTexture(GL_TEXTURE_2D, texture[0]);
```

Après nous créons la texture réelle. La ligne suivante dit à OpenGL que la texture sera une texture 2D (`GL_TEXTURE_2D`). Zéro représente les images du niveau de détail, qui est d'habitude laissé à 0. Trois est le nombre de composantes des données. Puisque l'image se compose de couleur rouge, verte, et bleue, il y a donc trois composantes. `TextureImage[0]->sizeX` est la largeur de la texture. Si vous en savez la taille, vous pouvez la renseigner ici, mais il est plus simple de laisser l'ordinateur la trouver pour vous. `TextureImage[0]->sizeY` est la hauteur de la texture. Zéro est le bord. Il est normalement laissé à zéro. `GL_RGB` indique à OpenGL que les données de l'image que nous utilisons sont composées de données rouge, verte, et bleue dans cet ordre. `GL_UNSIGNED_BYTE` signifie que les données qui composent l'image sont des bytes non signés, et enfin... `TextureImage[0]->data` indique à OpenGL où aller chercher les données de la texture. Dans ce cas il pointe sur les données stockées dans l'enregistrement `TextureImage[0]`.

Crée une texture 2D

```
// Genere la texture
glTexImage2D(GL_TEXTURE_2D, 0, 3, TextureImage[0]->sizeX,
TextureImage[0]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage[0]->data);
```

Les deux lignes de code suivantes indiquent à OpenGL quel type de filtrage il faut utiliser, lorsque l'image est étirée sur l'écran ou est plus grande que la texture originale (`GL_TEXTURE_MAG_FILTER`), ou lorsqu'elle est plus petite (`GL_TEXTURE_MIN_FILTER`) sur l'écran que la texture réelle. J'utilise habituellement `GL_LINEAR` pour les deux cas. Cela rend une texture douce lorsqu'elle est éloignée ou proche de l'écran. Utiliser `GL_LINEAR` demande beaucoup de travail pour le GPU et la mémoire de la carte graphique, alors si votre système est lent, vous pouvez utiliser `GL_NEAREST`. Une texture filtrée avec `GL_NEAREST` apparaîtra pixellisée si elle est déformée. Vous pouvez aussi essayer une combinaison des deux. Filtrer lorsque l'image est proche, mais pas lorsqu'elle est éloignée.

Filtrage

```
// Filtrage lineaire
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
// Filtrage lineaire
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
}
```

Maintenant nous libérons toute la mémoire que nous avons utilisée pour stocker les données du bitmap. Nous vérifions que la structure du bitmap était bien stockée dans `TextureImage[0]`. Si elle l'était, nous contrôlons que le contenu même du bitmap était bien en mémoire. S'il l'était, nous le supprimons. Ensuite nous libérons la structure d'image en s'assurant que toute la mémoire utilisée soit bien libérée.

Libération de la mémoire

```
if (TextureImage[0]) // Si la texture existe
{
    if (TextureImage[0]->data) // Si l'image de la texture existe
```

Libération de la mémoire

```

        {
            free(TextureImage[0]->data); // Libere l'image de la texture
        }
        free(TextureImage[0]);          // Libere la structure de l'image
    }
    
```

Pour finir, nous retournons le statut. Si tout était OK, la variable Status sera mise à TRUE. Si quelque chose a échoué, Status sera mis à FALSE.

Renvoi

```

    return Status; // Renvoie le statut
}
    
```

1.4 - La fonction InitGL

J'ai ajouté quelques lignes de code dans InitGL. Je reposte l'entière partie du code, il sera donc facile de voir le code que j'y ai ajouté et où je l'ai mis. La première ligne `if (!LoadGLTextures())` lance notre fonction `LoadGLTexture` qui charge le bitmap et crée une texture depuis celui-ci. Si la fonction échoue pour une quelconque raison, la ligne suivante retournera FALSE. Si tout s'est passé correctement et que la texture a été créée, nous faisons l'application de texture (*texture mapping*). Si vous oubliez d'activer l'application de texture, votre objet apparaîtra normalement en blanc, ce qui n'est bien sûr pas ce que nous voulons.

Initialisation avec une texture

```

int InitGL(GLvoid) // Toute la configuration d'OpenGL se met ici
{
    if (!LoadGLTextures()) // Va dans notre routine de chargement de texture
        (nouveau)
        {
            // Si la la texture n'a pas ete chargée, retourne FALSE (nouveau)
            return FALSE;
        }
    glEnable(GL_TEXTURE_2D); // Active l'application de texture (nouveau)
    glShadeModel(GL_SMOOTH); // Active les ombres douces
    glClearColor(0.0f, 0.0f, 0.0f, 0.5f); // Fond d'écran noir
    glClearDepth(1.0f); // Configuration du tampon de profondeur
    glEnable(GL_DEPTH_TEST); // Active le test de profondeur
    glDepthFunc(GL_LEQUAL); // Le type de test de profondeur a faire

    // Pour avoir des jolis calculs de perspective
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    return TRUE; // L'initialisation etait OK
}
    
```

1.5 - La fonction DrawGLScene

Nous allons maintenant dessiner un cube texturé. Vous pouvez remplacer le code de `DrawGLScene` avec le code ci-dessous, ou vous pouvez ajouter le nouveau code dans celui de la leçon une. Cette partie sera fortement commentée afin d'assurer une bonne compréhension. Les deux premières lignes `glClear()` and `glLoadIdentity()` sont dans le code de la leçon une. `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)` videra l'écran avec la couleur que nous avons sélectionnée dans `InitGL()`. Dans notre cas, il sera mis en noir. Le tampon de profondeur sera aussi vidé. La vue sera ensuite mise à zéro avec `glLoadIdentity()`.

Préparation

```

int DrawGLScene(GLvoid) // C'est ici que nous faisons tout les rendus
{
    
```

Préparation

```
// Vide l'ecran et le tampon de profondeur
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity(); // Remet a zero la matrice courante
glTranslatef(0.0f,0.0f,-5.0f); // Deplace dans l'ecran de 5 unites
```

Les trois lignes suivantes feront tourner le cube sur l'axe des X, puis l'axe des Y, et enfin l'axe des Z. L'angle de rotation dépendra de la valeur des variables xrot, yrot et zrot.

Rotation

```
glRotatef(xrot,1.0f,0.0f,0.0f); // Rotation sur l'axe des X
glRotatef(yrot,0.0f,1.0f,0.0f); // Rotation sur l'axe des Y
glRotatef(zrot,0.0f,0.0f,1.0f); // Rotation sur l'axe des Z
```

La ligne suivante sélectionne quelle texture nous voulons utiliser. Si vous voulez utiliser plus d'une texture dans votre scène, vous pouvez choisir la texture en utilisant `glBindTexture(GL_TEXTURE_2D, texture[numéro de la texture])`. Si vous voulez changer de texture, vous pouvez activer la nouvelle texture avec `glBindTexture()`. Une chose à savoir est que l'on ne peut pas utiliser cette fonction à l'intérieur de `glBegin()` et `glEnd()`, vous devez le faire avant ou après. Observez comment nous employons `glBindTexture` pour spécifier quelle texture créer et choisir une texture spécifique.

Sélection de la texture

```
glBindTexture(GL_TEXTURE_2D, texture[0]); // Selectionne notre texture
```

Pour appliquer proprement une texture sur un quadrilatère, vous devez vous assurer que la partie supérieure droite de la texture soit appliquée sur la partie supérieure droite du quadrilatère, la partie supérieure gauche de la texture sur la partie supérieure gauche du quadrilatère, etc. Si les coins de la texture ne correspondent pas avec ceux du quadrilatère, l'image pourrait apparaître à l'envers, décalée, ou ne pas être affichée du tout.

Le premier paramètre de `glTexCoord2f` est la coordonnée X. 0.0f est le côté gauche de la texture, 0.5f est le milieu et 1.0f est le côté droit. Le second paramètre de `glTexCoord2f` est la coordonnée Y. 0.0f est le bas de la texture, 0.5f le milieu et 1.0f le haut de la texture.

Nous savons maintenant que la coordonnée en-haut à gauche de la texture est 0.0f en X et 1.0f en Y, et que le sommet en haut à gauche d'un quadrilatère est -1.0f en X et 1.0f en Y. Maintenant tout ce que vous avez à faire est de faire correspondre les trois autres coordonnées de texture avec les trois coins restants du quadrilatère.

Essayez de jouer avec les valeurs de X et Y de `glTextCoord2f`. Changer le 1.0f en 0.5f dessinera seulement la moitié gauche d'une texture de 0.0f (gauche) à 0.5f (milieu de la texture). Changer 0.0f en 0.5f dessinera seulement la moitié droite d'une texture de 0.0f (milieu) à 1.0f (droite).

Tracé du cube texturé

```
glBegin(GL_QUADS);
// Face de devant
// En-bas a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
// En-bas a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
// En-haut a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
// En-haut a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

// Face arriere
// En-bas a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
// En-haut a droite de la texture et du quadrilatere
```

Tracé du cube texturé

```
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
// En-haut a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
// En-bas a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);

// Face superieure
// En-haut a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
// En-bas a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
// En-bas a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
// En-haut a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);

// Face inferieure
// En-haut a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
// En-haut a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
// En-bas a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
// En-bas a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);

// Face de droite
// En-bas a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
// En-haut a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
// En-haut a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
// En-bas a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);

// Face de gauche
// En-bas a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
// En-bas a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
// En-haut a droite de la texture et du quadrilatere
glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
// En-haut a gauche de la texture et du quadrilatere
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

glEnd();
```

Maintenant nous incrémentons la valeur de xrot, yrot et zrot. Essayez de changer la valeur d'incrément de chaque variable pour faire tourner le cube plus ou moins vite, ou essayez de changer le signe de l'incrément pour le faire tourner dans l'autre sens.

Rotation

```
xrot+=0.3f; // Rotation sur l'axe des X
yrot+=0.2f; // Rotation sur l'axe des Y
zrot+=0.4f; // Rotation sur l'axe des Z
return true; // Tout va bien
```

1.6 - Conclusion

Vous devriez maintenant avoir une meilleure compréhension de l'application de texture. Vous devriez être capable de texturer la surface dans n'importe quel quadrilatère avec une image de votre choix. Lorsque vous vous sentirez en confiance avec l'application de texture 2D, essayez d'ajouter six textures différentes au cube.

L'application de texture n'est pas très difficile à comprendre une fois que vous aurez compris les coordonnées de texture. Alors amusez-vous bien à créer vos propres scènes texturées :).

Enfin, voici une petite image de ce que vous devez voir :



Image de l'application

Jeff Molofee (NeHe)

2 - Téléchargements

Compte tenu du nombre de versions de codes sources pour les tutoriels nehe, nous les laissons en anglais. En principe, si vous avez compris le code présenté dans ce tutoriel (et les tutoriels antérieurs), vous n'aurez pas de mal à le comprendre :

- **Borland C++ Builder 6 (Conversion par Christian Kindahl)**
- **C# (Conversion par Sabine Felsing)**
- **Code Warrior 5.3 (Conversion par Scott Lupton)**
- **CygWin (Conversion par Stephan Ferraro)**
- **D (Conversion par Familia Pineda Garcia)**
- **Delphi (Conversion par Michal Tucek)**
- **Dev C++ (Conversion par Dan)**
- **Euphoria (Conversion par Evan Marshall)**
- **Game GLUT (Conversion par Milikas Anastasios)**
- **GLUT (Conversion par Kyle Gancarz)**
- **Irix (Conversion par Lakmal Gunasekara)**
- **Java (Conversion par Jeff Kirby)**
- **Jedi-SDL (Conversion par Dominique Louis)**
- **JOGL (Conversion par Kevin J. Duling)**
- **LCC Win32 (Conversion par Robert Wishlaw)**
- **Linux (Conversion par Richard Campbell)**
- **Linux GLX (Conversion par Mihael Vrbanec)**
- **Linux SDL (Conversion par Ti Leggett)**
- **LWJGL (Conversion par Mark Bernard)**
- **Mac OS (Conversion par Anthony Parker)**
- **Mac OS X/Cocoa (Conversion par Bryan Blackburn)**
- **MASM (Conversion par Nico (Scalp))**
- **VC++/OpenIL (Conversion par Denton Woods)**
- **Pelles C (Conversion par Pelle Orinius)**
- **Power Basic (Conversion par Angus Law)**
- **Python (Conversion par John Ferguson)**
- **REALBasic (Conversion par Thomas J. Cunningham)**
- **Scheme (Conversion par Brendan Burns)**
- **Solaris (Conversion par Lakmal Gunasekara)**
- **VB.NET CsGL (Conversion par X)**
- **Visual Basic (Conversion par Ross Dawson)**
- **Visual Basic (Conversion par Peter de Tagyos)**
- **Visual Fortran (Conversion par Jean-Philippe Perois)**
- **Visual Studio**
- **Visual Studio NET (Conversion par Grant James)**

3 - Remerciements

Merci à **khayyam90**, **qi130** et à **jc_cornic** pour leur relecture.

4 - Liens

Sommaire

Tutoriel précédent : Première pyramide et premier cube

Tutoriel prochain : Les filtres et l'éclairage

