Institut Universitaire de Technologie
Aix*Marseille Université

Imagerie Numérique
Synthèse d'images

2. Texture

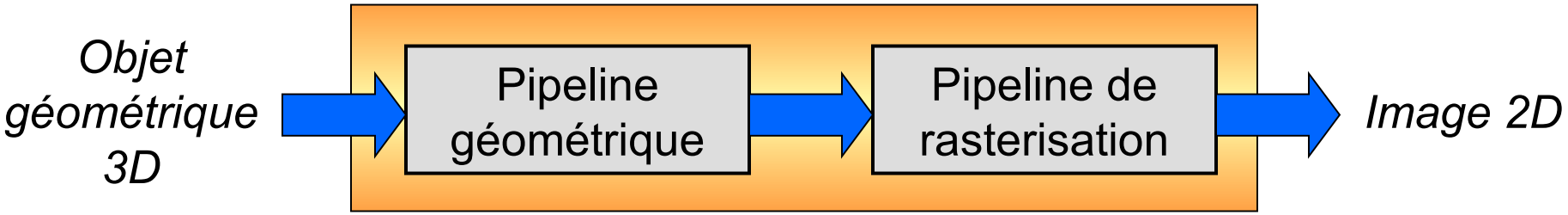
DUT INFO 2ème année 2013-2014

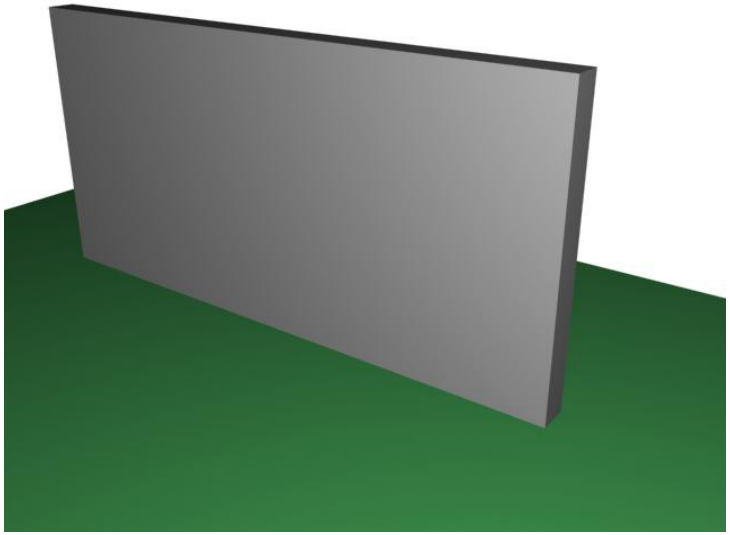
Sébastien THON

IUT d'Aix-Marseille Université, site d'Arles
Département Informatique

1. Problématique

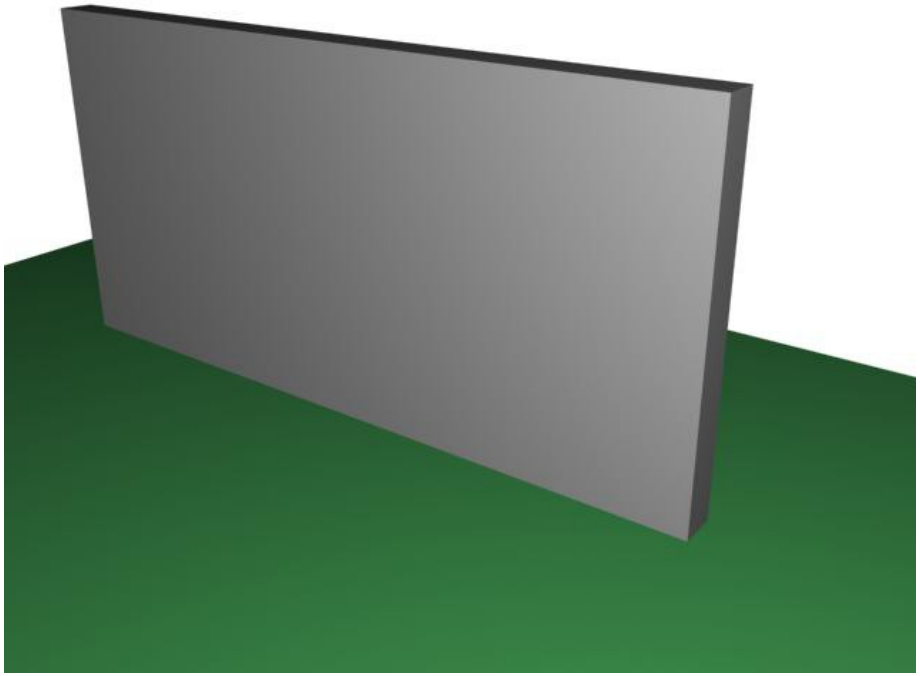
Modèle de Phong : prise en compte des interactions lumière/matière, mais ce n'est pas suffisant.

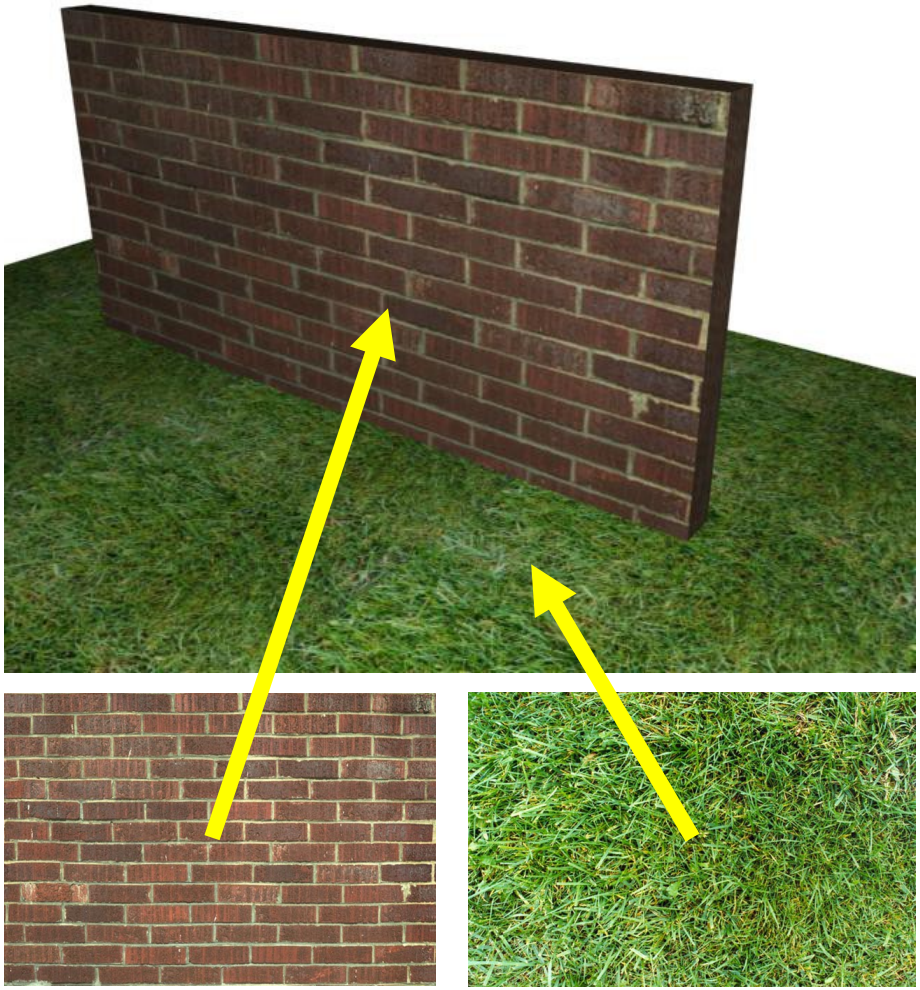




Manque de réalisme d'une scène n'utilisant que le modèle d'illumination de Phong.

Solution : utilisation de **textures**.

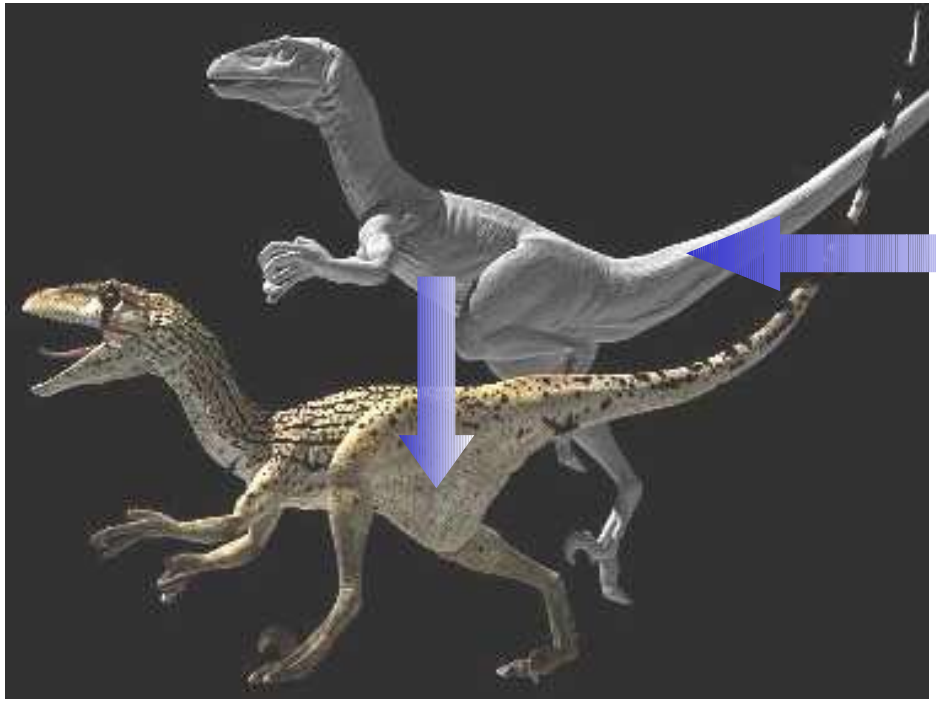




Plaquage de textures (« *texture mapping* »)
Plaquage d'images sur des primitives géométriques.

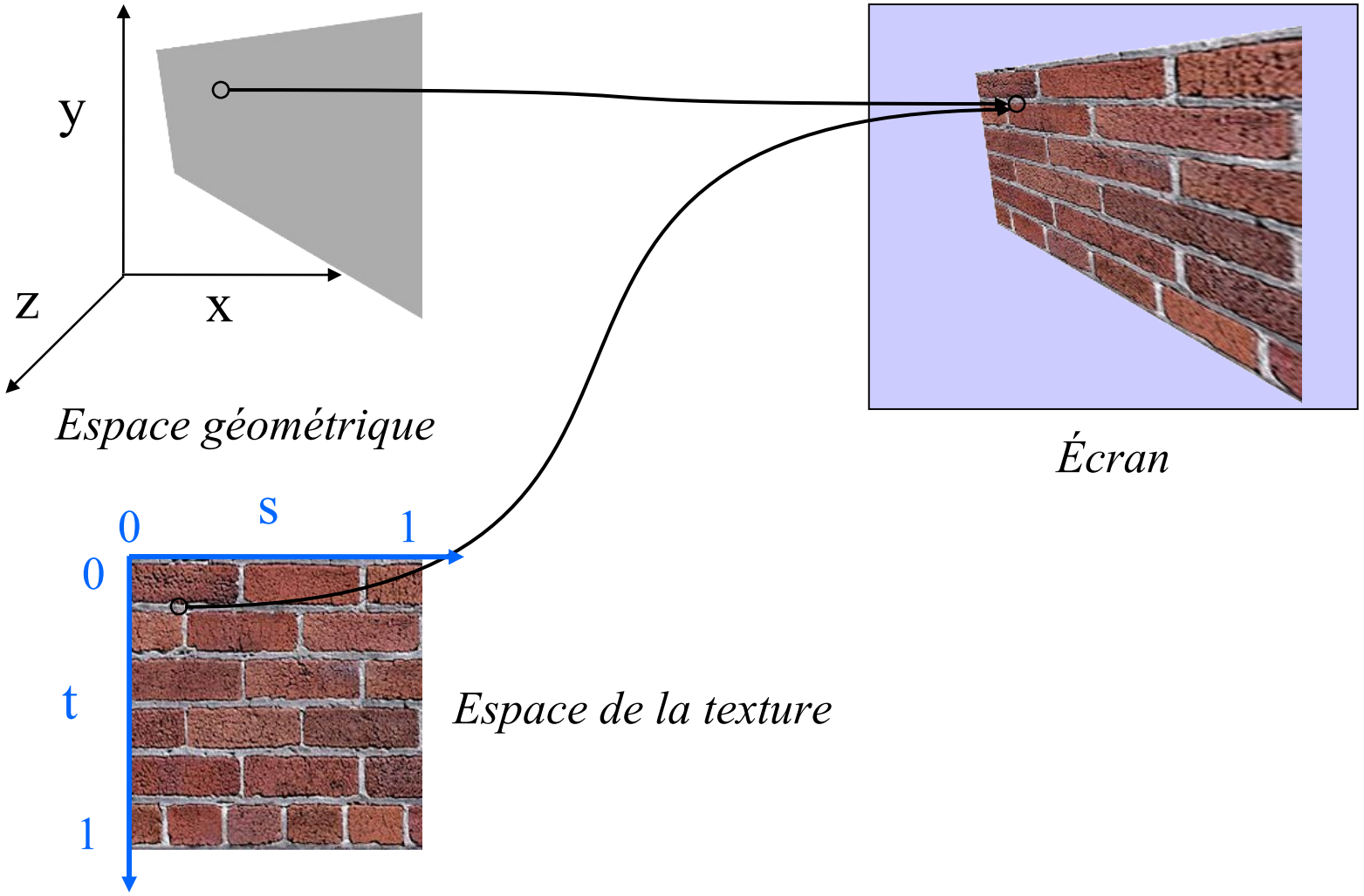
Objectifs

- Simuler des matériaux (pierre, bois, …)
- Réduire la complexité (nb de polygones) d'objets 3D
- Simulation de surfaces réfléchissantes
- …



2. Principe du plaquage de texture

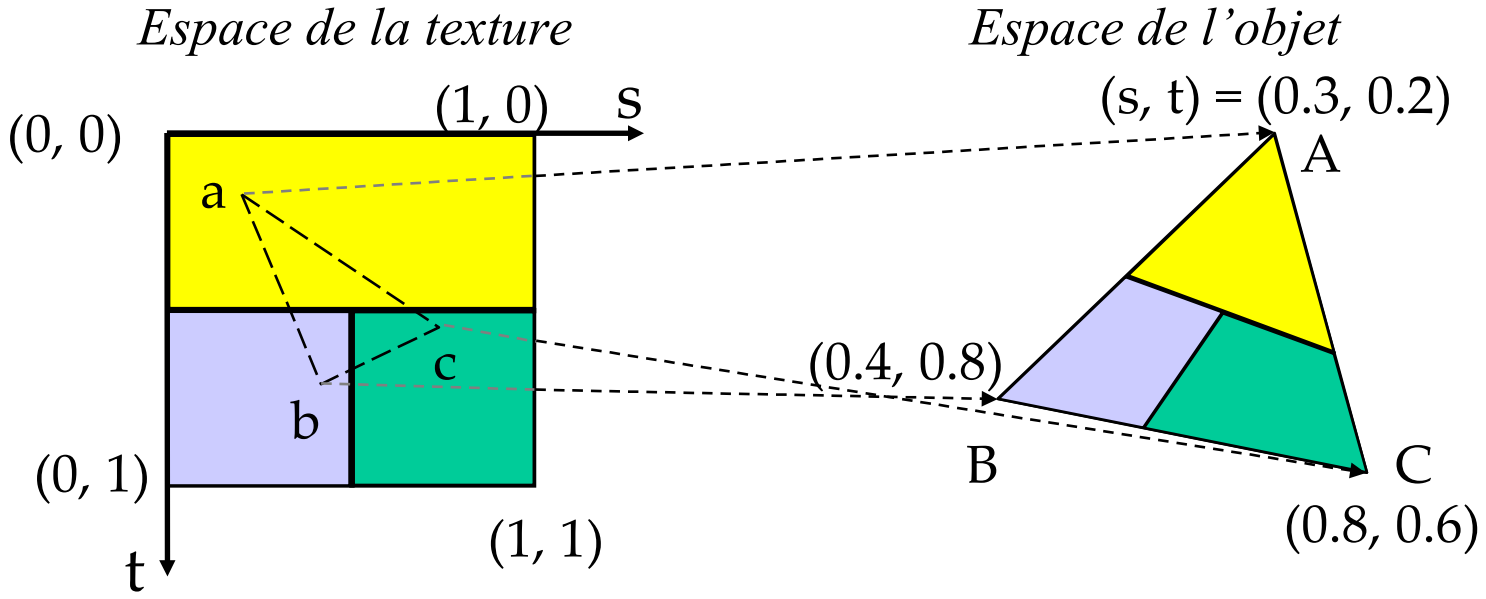
2.1 Coordonnées de texture



Les textures sont des images qui peuvent être en :

- 1D : coordonnée de texture (s)
- **2D : coordonnées de texture (s, t), cas le plus courant**
- 3D : coordonnées de texture (s, t, r)
- 4D : coordonnées de texture (s, t, r, q)

Ex: pour une texture 2D, un point dans l'image est donné par ses coordonnées (s,t), le coin supérieur gauche étant (0,0) et le coin inférieur droit étant (1,1).

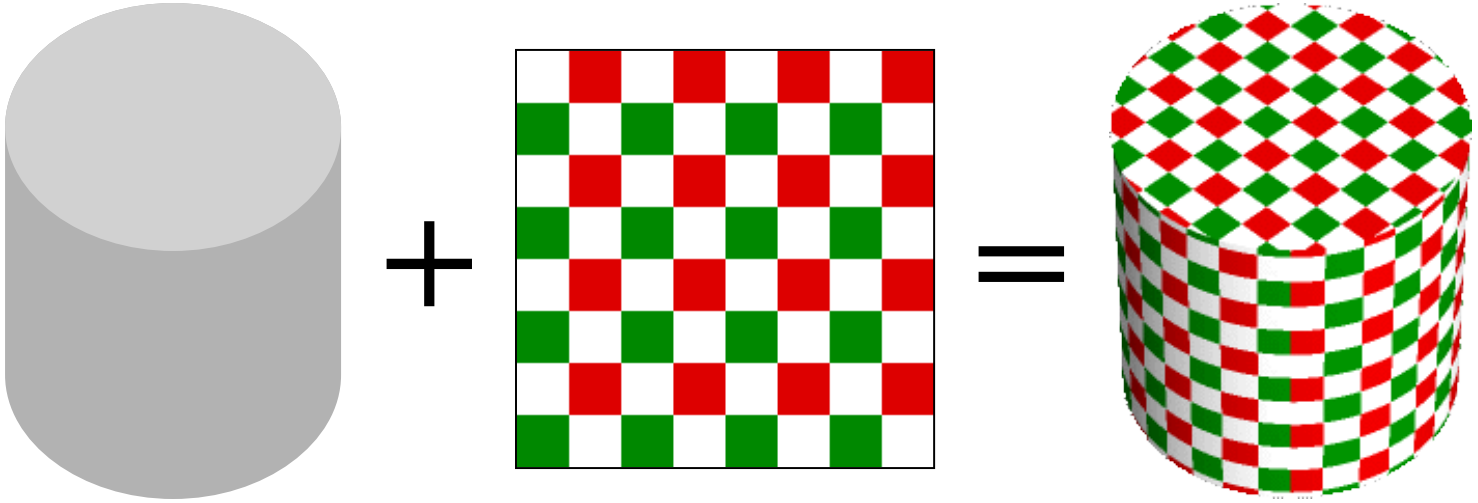


Notes :

- Un pixel d'une texture est appelé *texel*.
- Dans OpenGL, les dimensions de la texture doivent être une puissance de 2 (ce n'est pas le cas dans OpenGL 2.0).
- Les dimensions des textures sont limitées (dépend des cartes graphiques).

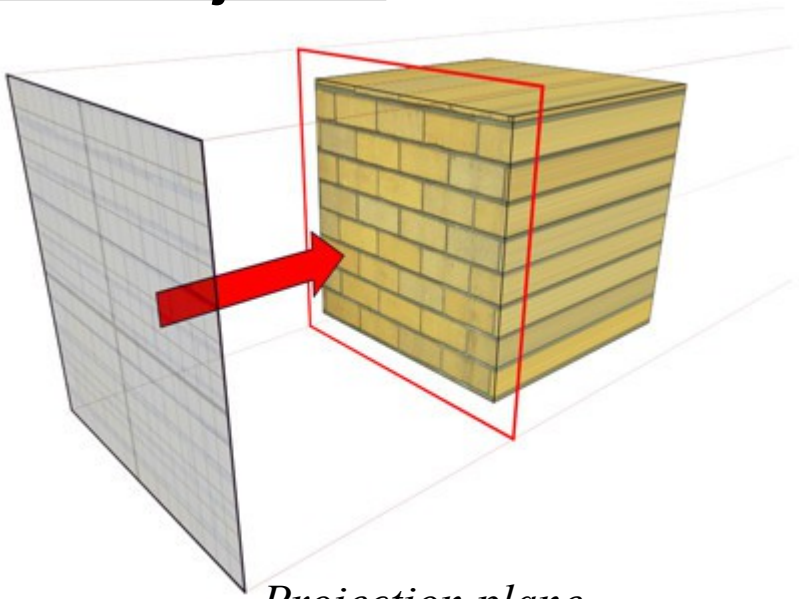
2.2 Paramétrisation

Question : Comment définir les coordonnées de texture des sommets de l'objet 3D à texturer ? « Comment définir les endroits de l'objet 3D où iront les couleurs de l'image ? »

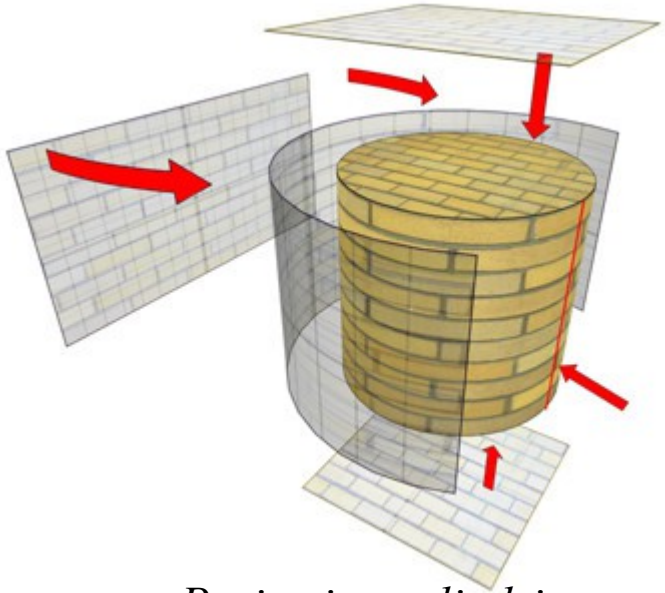


Objet 3D Texture Objet 3D texturé

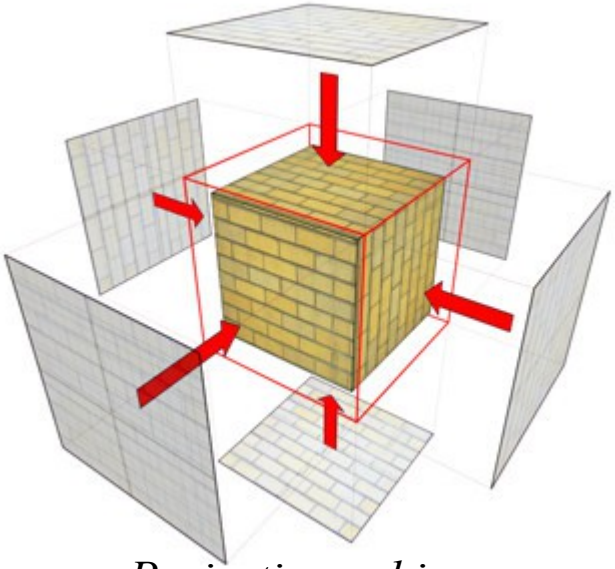
2.2.1 Projection



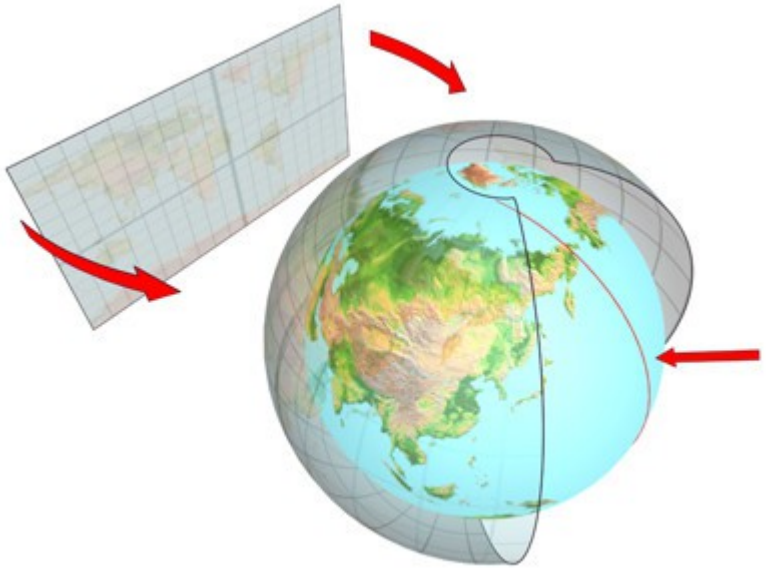
Projection plane



Projection cylindrique

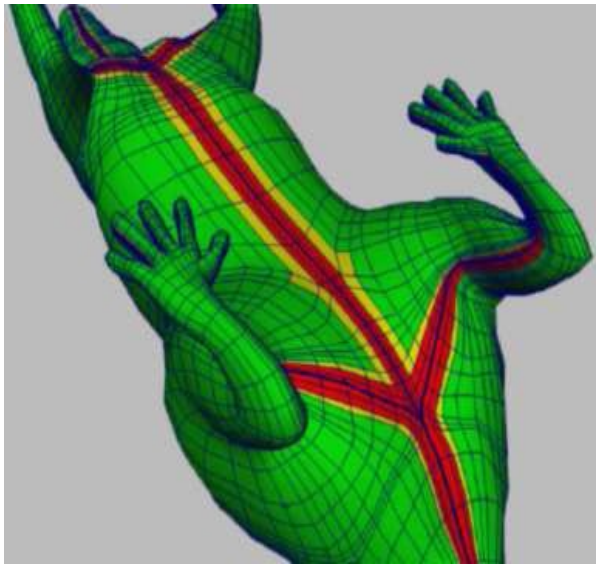


Projection cubique

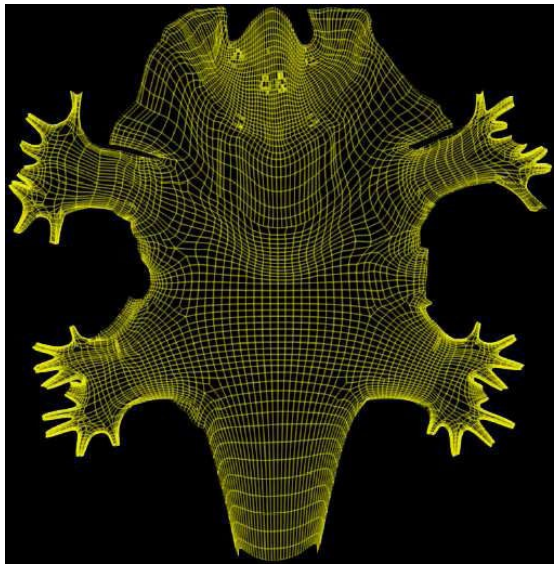


Projection sphérique

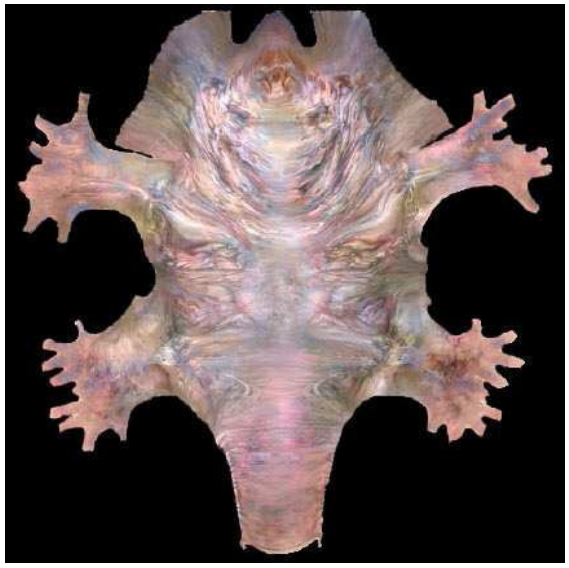
2.2.2 Dépliage de texture



Modèle 3D à texturer



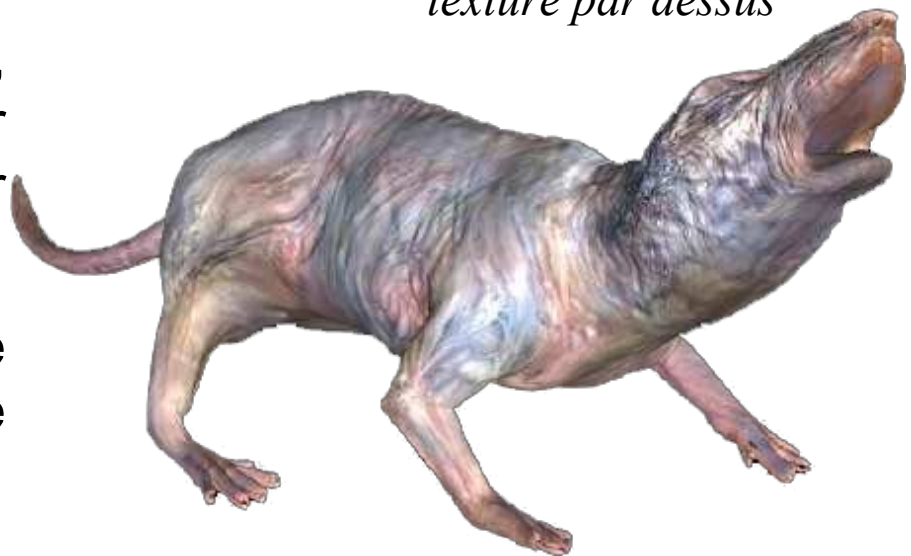
Le maillage 3D déplié en 2D



Un artiste dessine la texture par dessus

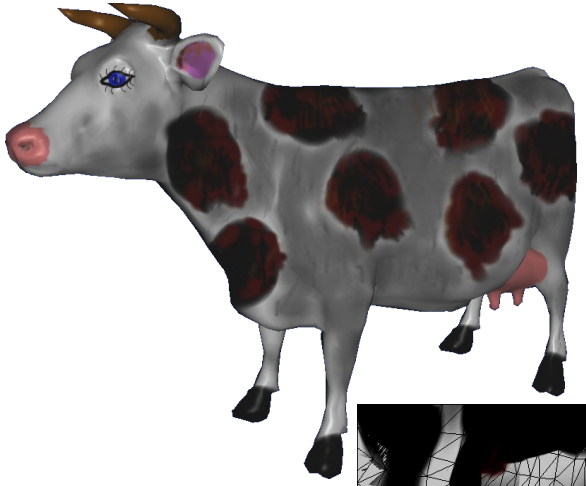
Le maillage du modèle 3D est déplié, tracé dans l'image 2D de la texture par un algorithme qui essaie de conserver les surfaces des triangles.

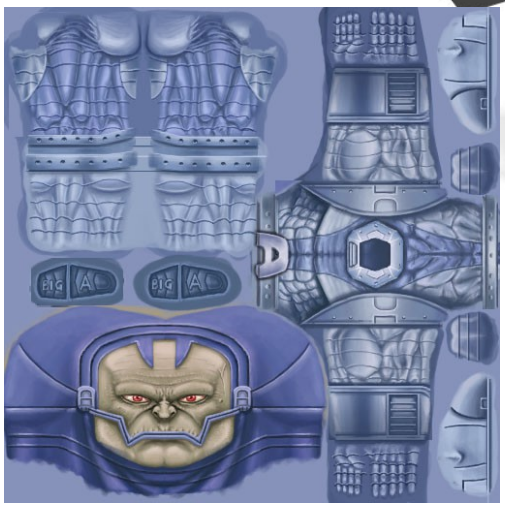
Un artiste dessine ensuite la texture par dessus en se servant du maillage comme guide.

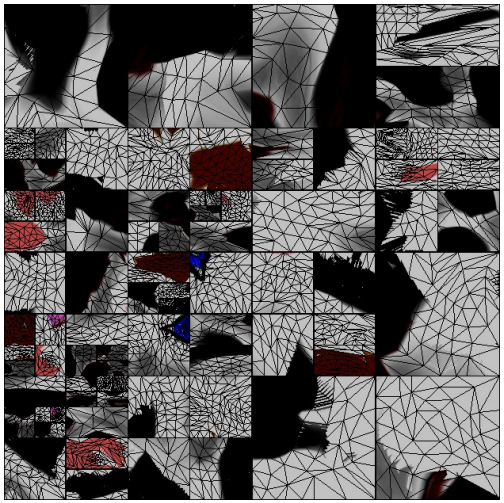


2.2.3 Atlas de textures

Dans l'espace de la texture, les différentes parties de la texture sont disjointes. Elles correspondent à des groupes de triangles du modèle 3D. Il faut répartir ces groupes de triangles de manière à optimiser l'utilisation de la surface de la texture.

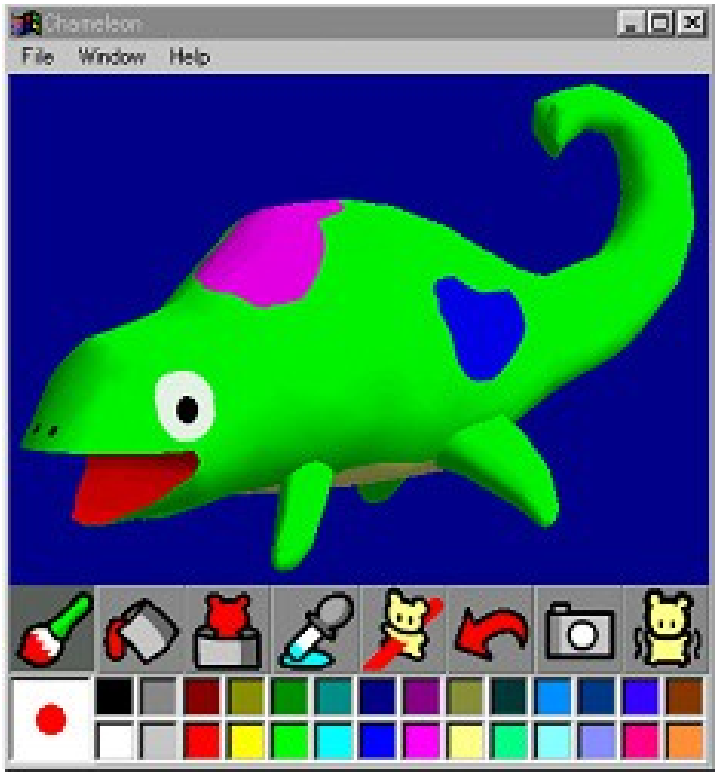


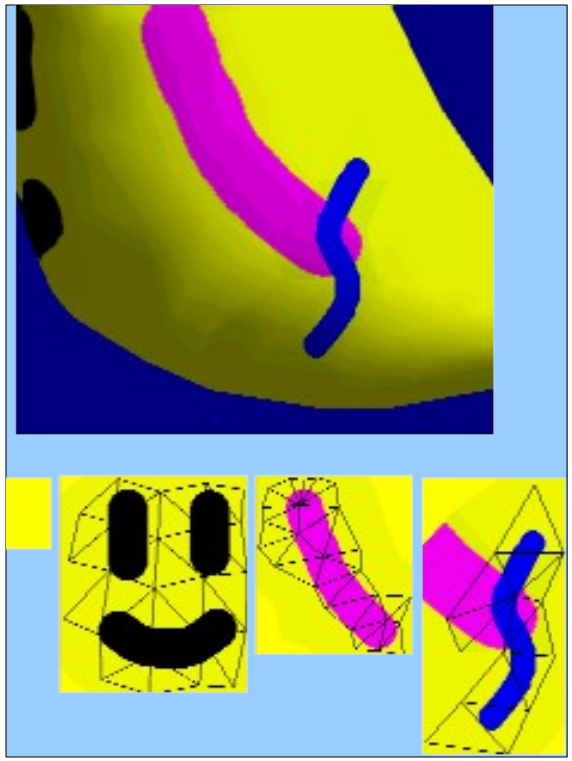




2.2.4 Peinture 3D

Des logiciels de création de texture permettent de dessiner directement avec des outils de dessin sur le modèle 3D à texturer. Ces logiciels créent au fur et à mesure un atlas de textures ou ont au préalable déplié la géométrie (Deep Paint 3D, Chameleon Paint System).





2.2.5 Textures procédurales

Les textures sont de deux types :

- **raster** = image composée de pixels (c'est le cas le plus courant)
- **procédural** = fonction mathématique

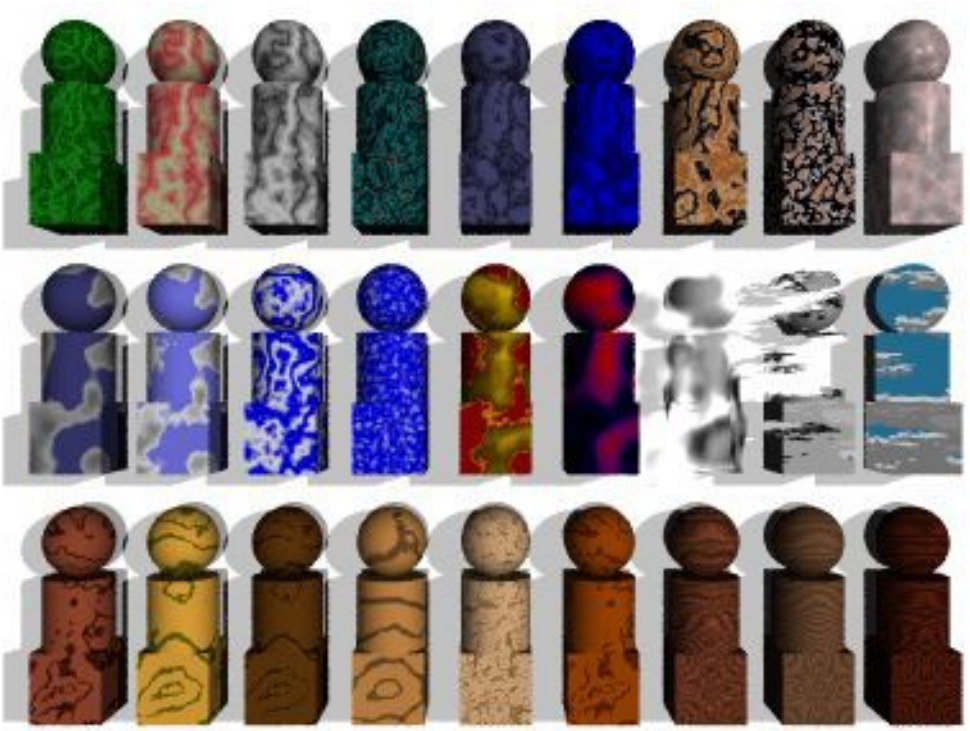
On calcule la couleur en un point de la surface de l'objet par une fonction 2D $f(x,y)$ ou en tout point du volume de l'objet par une fonction 3D $f(x,y,z)$

→ Textures procédurales 2D et 3D

On peut facilement écrire ces fonctions avec des ***shaders*** (voir chapitre 5).

Textures procédurales de bois, de marbre, de pierre, etc.

Les textures procédurales sont utilisées dans de nombreux logiciels de synthèse d'images (POV, 3D Studio, …)



Exemple: texture procédurale 2D de damier :

```
COULEUR texture_damier( int x, int y )
{
   if( (x+y)%2 == 0 )
      return COULEUR(0,0,0);
   else
      return COULEUR(255,255,255);
}
```

Avantages des textures procédurales (2D et 3D)

Ces fonctions nécessitent très peu de mémoire (à peine quelques octets, pour le code de ces fonctions).

Textures 3D: Permet d'éviter les problèmes de plaquage et de raccord des textures 2D.

Inconvénients des textures procédurales (2D et 3D)

Le calcul de la couleur en un point peut être long (dépend de la complexité de la fonction).

La détermination des paramètres de ces fonctions n'est pas facile (peu intuitif).

3. Utilisation de textures dans OpenGL

On procède en 3 étapes :

1) Spécifier la texture

- Lire ou générer une image
- En faire une texture
- Activer le plaquage de texture

2) Assigner les coordonnées de texture aux sommets de l'objet 3D

3) Spécifier les paramètres de textures

- Wraping, filtering, …

3.1 Spécifier la texture

3.1.1 Lire ou générer une image

```
BYTE    *img;
int     largeur, hauteur;
GLuint  texture;

glGenTextures(1, &texture);
img = load_tga( "image.tga", &largeur, &hauteur );
```

Explication:

Dans OpenGL, **chaque texture est référencée par un indice** (entier). Pour obtenir ces indices, on utilise la fonction :

```
glGenTextures(GLuint n, GLuint *tab_text);
```

Qui crée **`n`** indices de textures et les place dans le tableau d'entiers **`tab_text`**.

Ex: obtention d'un ensemble d'indices pour 10 textures (→ tableau de 10 indices) :

```
GLuint tab_text[10];
glGenTextures(10, tab_text);
```

On peut aussi utiliser cette fonction pour ne demander qu'un seul indice :

Ex: obtention d'un indice pour une seule texture (→ une seule variable entière) :

```
GLuint texture;
glGenTextures(1, &texture);
```

3.1.2 En faire une texture

Les textures doivent être stockées dans la RAM de la carte graphique.

- Lorsqu'on charge une image pour en faire une texture, il faut ensuite la transférer dans la RAM vidéo :

```
glBindTexture(GL_TEXTURE_2D, texture);

glTexImage2D( GL_TEXTURE_2D, 0, 3,
              largeur, hauteur,
              0, GL_RGB, GL_UNSIGNED_BYTE, img);
```

La fonction **`glBindTexture()`** permet de fixer l'indice de la **texture courante**. Cette texture sera utilisée pour toutes les opérations (plaquage, modification de paramètres, …) jusqu'au prochain appel de **`glBindTexture()`**.

Explication:

```
glTexImage2D( target, level, components, w, h,
              border, format, type, *texels );
```

target : **GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D**

level : **0** sans mip-mapping

components : nombre d'éléments par texel

w, h : dimensions de la texture (puissances de 2)

border : bordure supplémentaire autour de l'image

format : **GL_RGB, GL_RGBA, ...**

type : type des éléments des texels

texels : tableau de texels

3.1.3 Activer le plaquage de texture

On active la plaquage avec :

```
glEnable(GL_TEXTURE_2D);
```

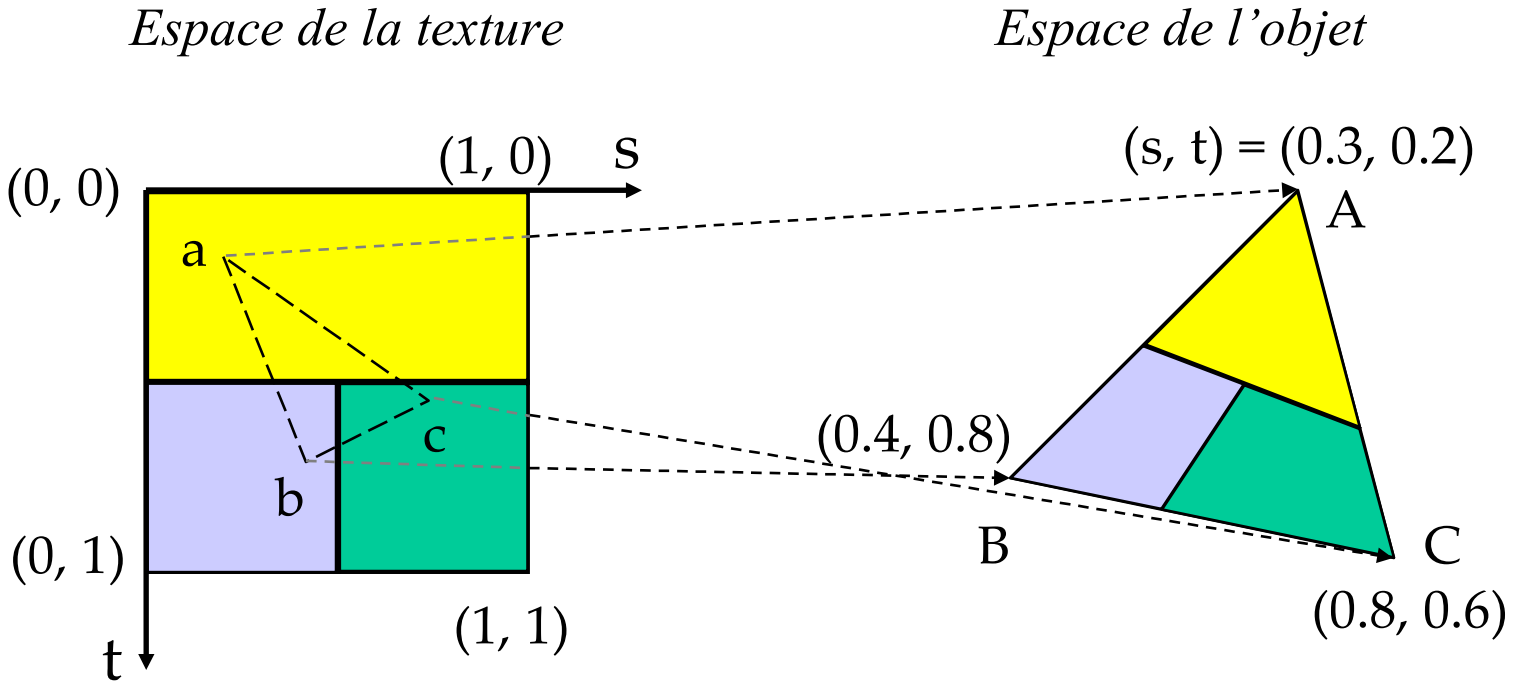
On peut aussi désactiver le plaquage :

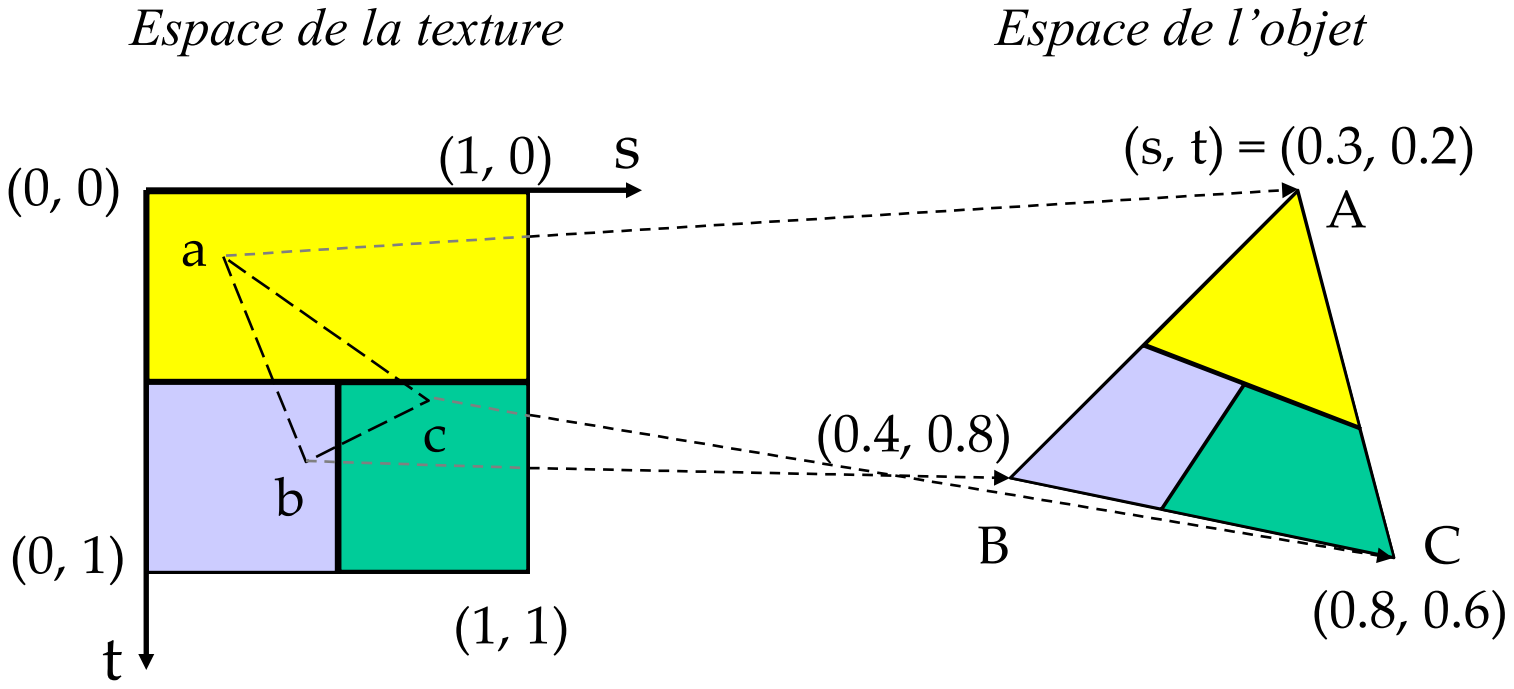
```
glDisable(GL_TEXTURE_2D);
```

On peut appeler ces deux fonctions à tout moment dans le programme, par exemple pour activer temporairement le plaquage afin d'afficher un objet texturé, puis en le désactivant pour afficher un objet qui ne comporte pas de texture.

3.2 Assigner les coordonnées de texture aux points de l'objet 3D

Pour plaquer une texture sur un objet géométrique, associer à chaque sommet 3D de l'objet les coordonnées 2D de texture (normalisés entre 0 et 1).






```
glBindTexture(GL_TEXTURE_2D, texture);

glBegin(GL_TRIANGLES);
  glTexCoord2f(0.3f,0.2f);
  glVertex3f(10.0f, 12.0f, 0.0f);
  glTexCoord2f(0.4f,0.8f);
  glVertex3f(4.0f, 6.0f, 0.0f);
  glTexCoord2f(0.8f,0.6f);
  glVertex3f(15.0f, 2.0f, 0.0f);
glEnd();
```

3.3 Paramètres de textures

Modes de filtrage

- Réduction, agrandissement
- Mip-mapping

Modes de bouclage

- Répéter, tronquer

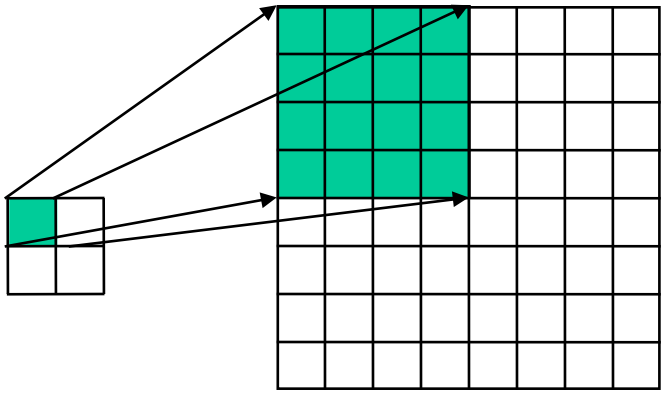
Fonctions de textures

- Comment mélanger la couleur d'un objet avec sa texture

3.3.1 Modes de filtrage

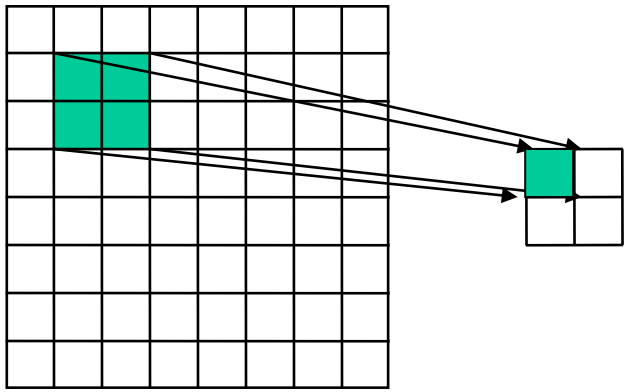
1) Réduction, agrandissement

Les textures et les objets texturés ont rarement la même taille (en pixels). OpenGL défini des filtres indiquant comment un texel doit être agrandit ou réduit pour correspondre à la taille d'un pixel.



Agrandissement

`GL_TEXTURE_MAG_FILTER`



Réduction

`GL_TEXTURE_MIN_FILTER`

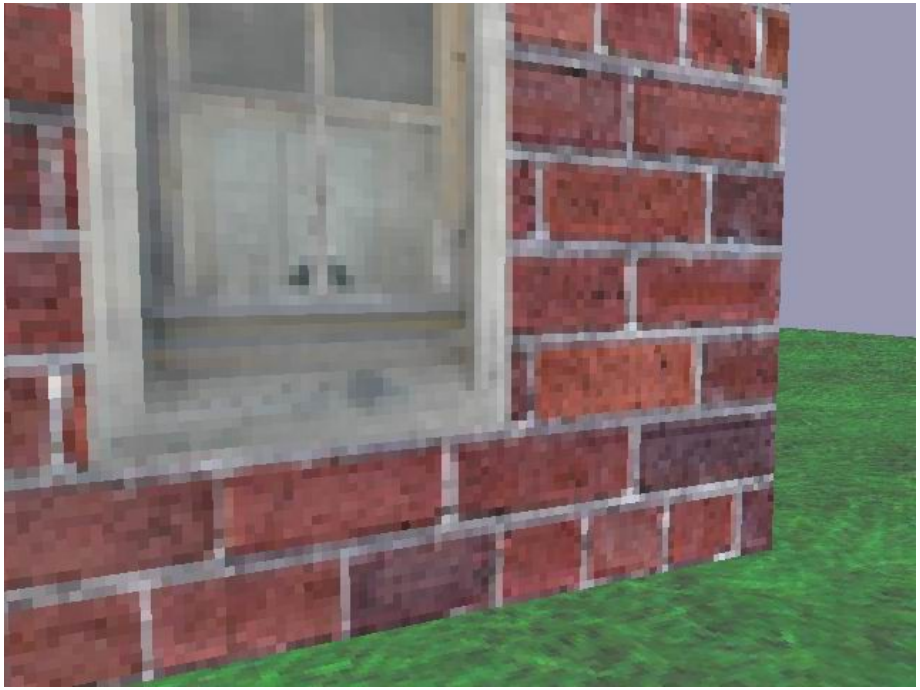
```
glTexParameteri(target, type, mode);
```

Avec :

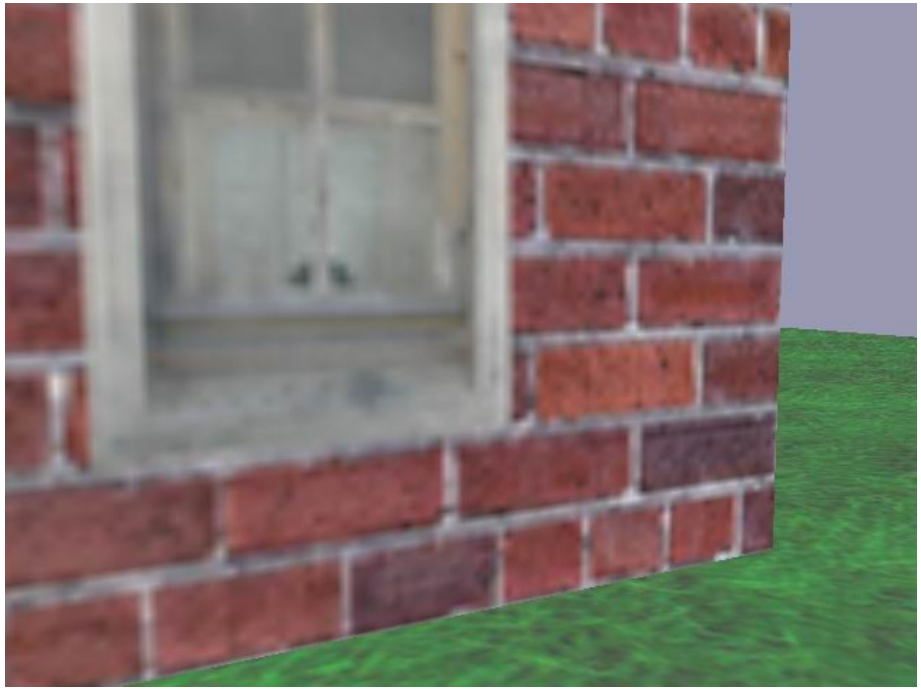
```
target : GL_TEXTURE_2D, ...
type   : GL_TEXTURE_MIN_FILTER,
         GL_TEXTURE_MAG_FILTER
mode   : GL_NEAREST, GL_LINEAR
```

GL_NEAREST : la couleur du pixel est donnée par celle du texel le plus proche.

GL_LINEAR : la couleur du pixel est calculée par interpolation linéaire des texels les plus proches.



GL_NEAREST



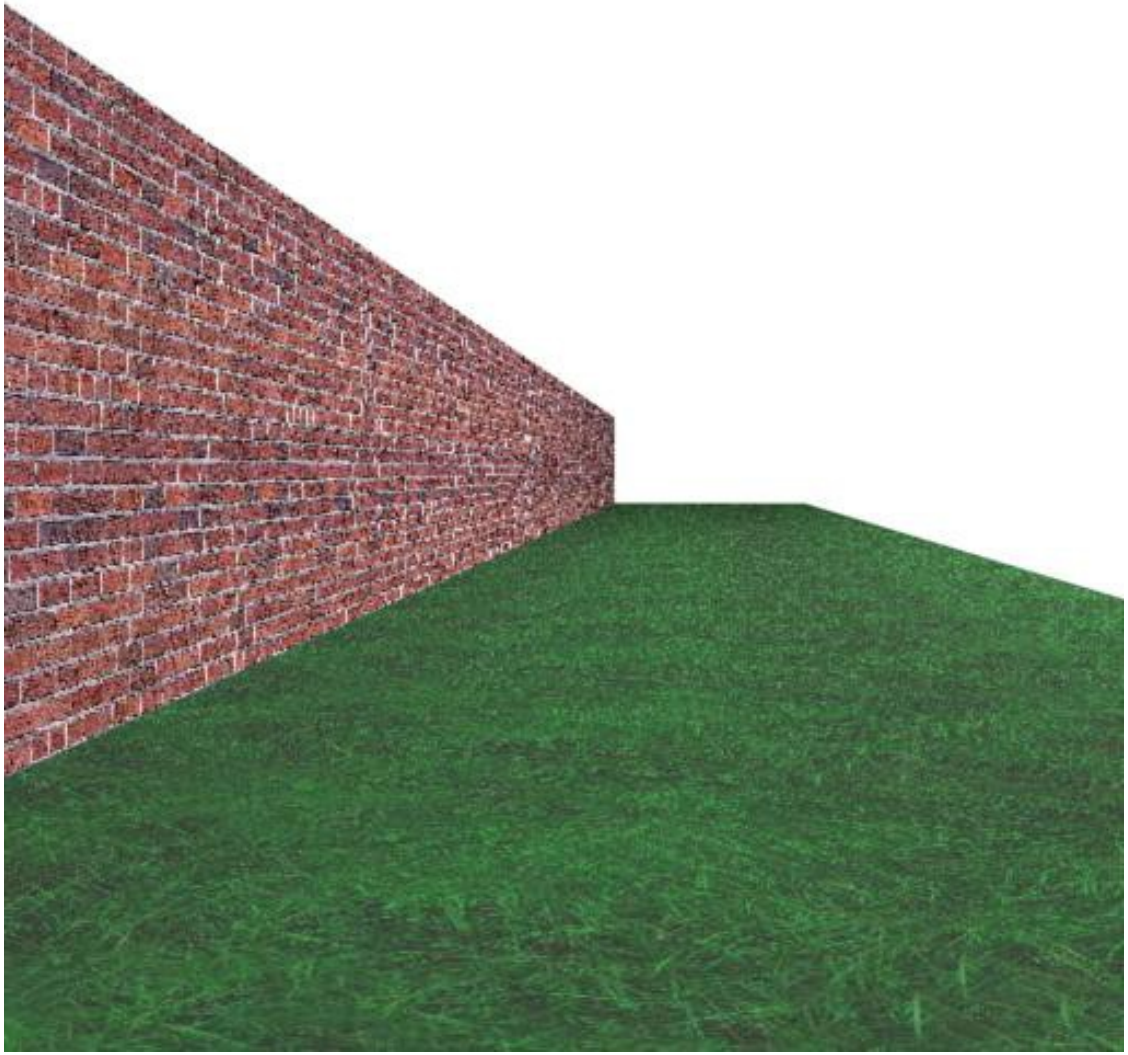
GL_LINEAR

Ex:

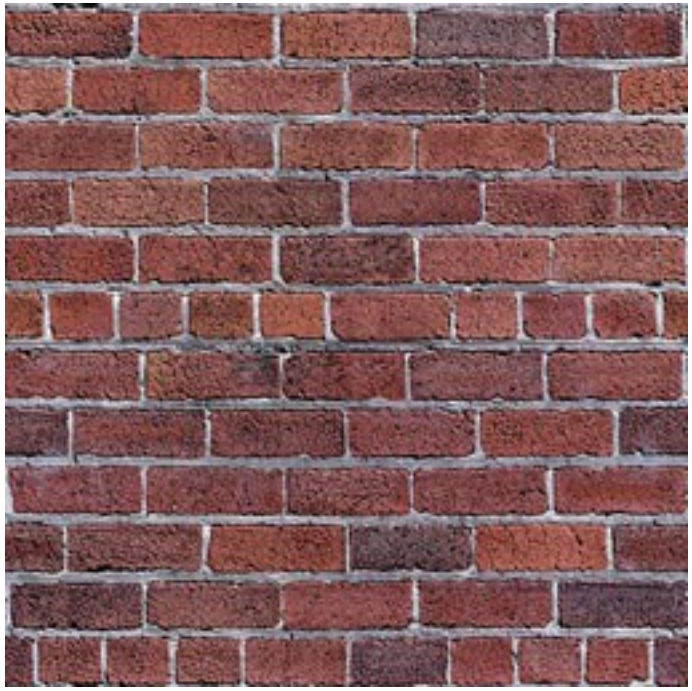
```
glBindTexture(GL_TEXTURE_2D, texture);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER,
                GL_LINEAR);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);
```

2) Mip-mapping

Problème d’aliassage :



Solution : La technique de **mip-mapping** consiste en un précalcul de plusieurs versions réduites d'une même texture.



256 x 256

128 x 128

64 x 64

32 x 32

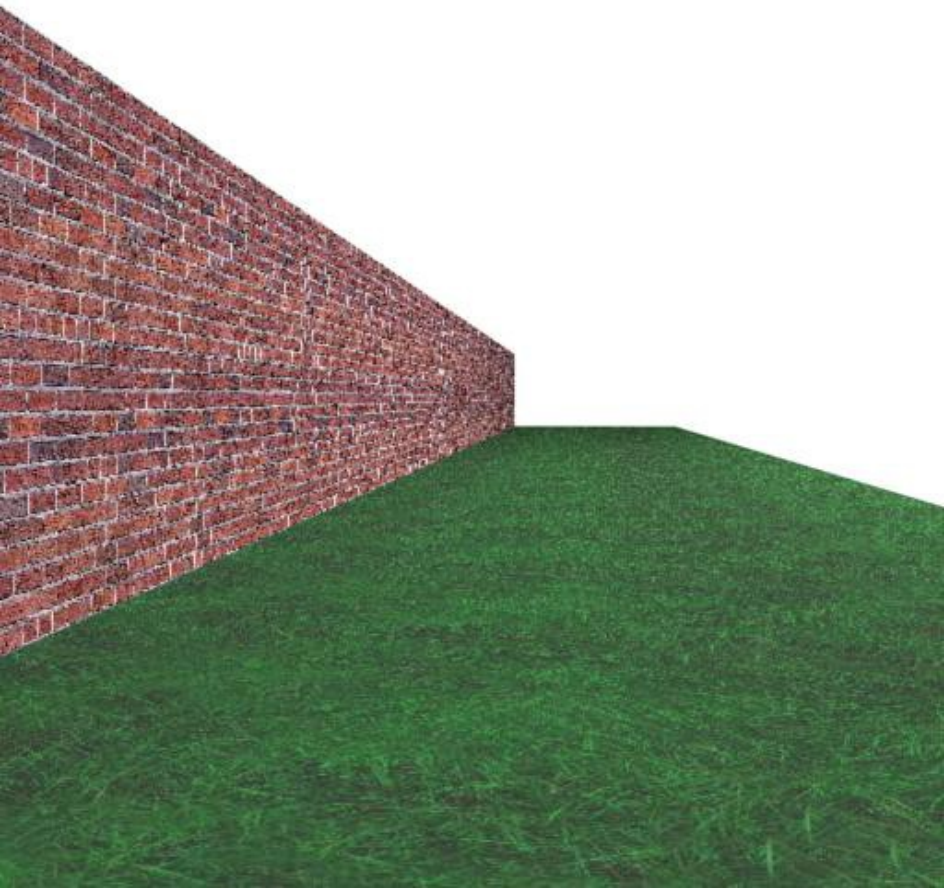
16 x 16

8 x 8

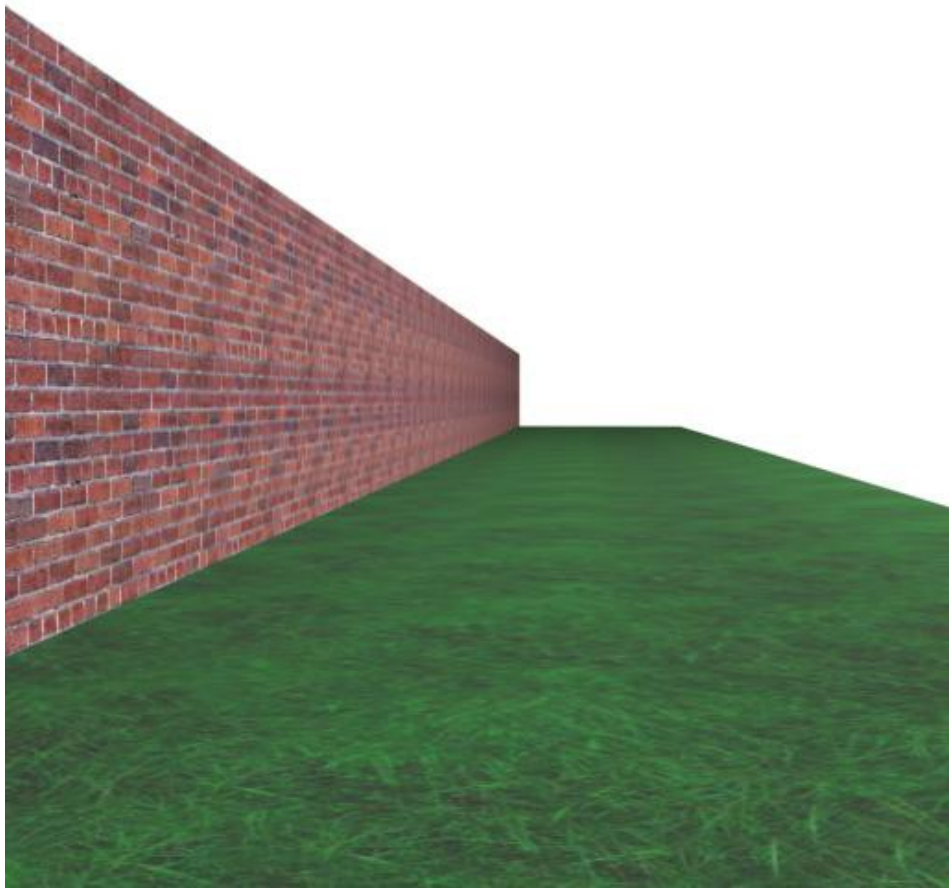
Dimensions : puissances de 2 (min : 1x1 texel)

→ Approche le mécanisme de vision humaine : adapte le niveau de détails en fonction de la distance.

→ Permet de réduire les problèmes d'aliassage.



Sans mip-mapping



Avec mip-mapping

Une fonction d'OpenGL permet de construire automatiquement les différentes textures de mip-map :

```
gluBuild2DMipmaps(GL_TEXTURE_2D, 3,
                  width, height,
                  GL_RGB, GL_UNSIGNED_BYTE, img);
```

Il faut ensuite indiquer le mode de filtrage de la texture en indiquant qu'on veut s'en servir en tant que mip-map :

```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);
```

3.3.2 Modes de bouclage (Wrap)

Ce mode indique ce qui doit se produire si une coordonnée de texture sort de l'intervalle [0,1].

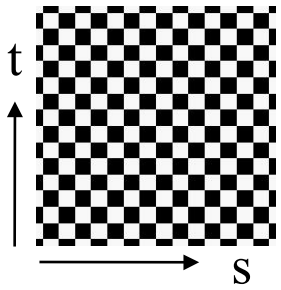
Deux possibilités :

- Répéter (« *Repeat* »)
- Tronquer (« *Clamp* »)

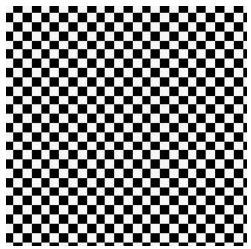
1) Répéter

Si le mode GL_REPEAT est utilisé, pour les coordonnées <0 ou >1, la partie entière est ignorée et seule la partie décimale est utilisée.

```
glTexParameteri( GL_TEXTURE_2D,
                 GL_TEXTURE_WRAP_S,
                 GL_REPEAT );
```



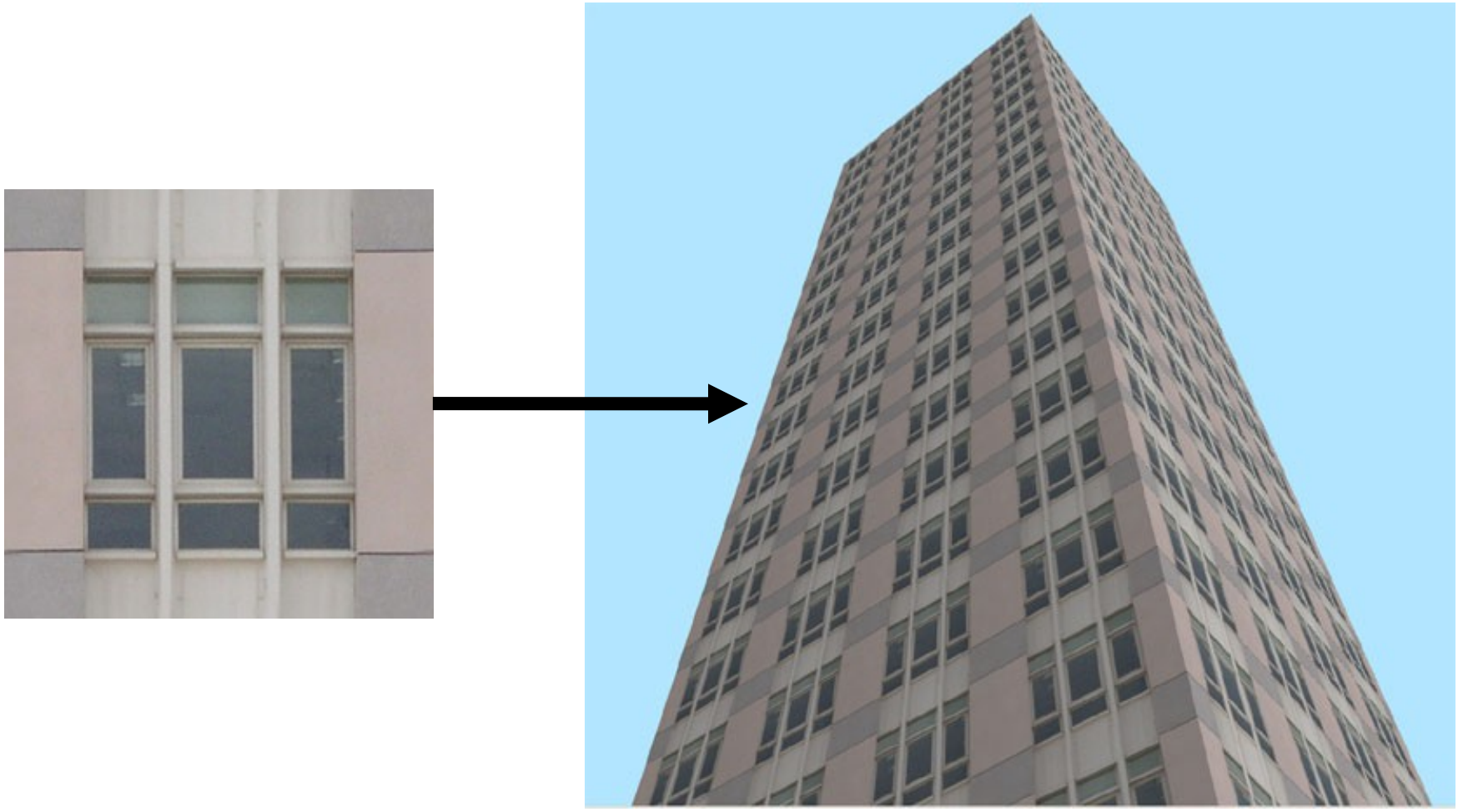
Texture



GL_REPEAT

Exemple : Répétition d'une texture en mode GL_REPEAT.

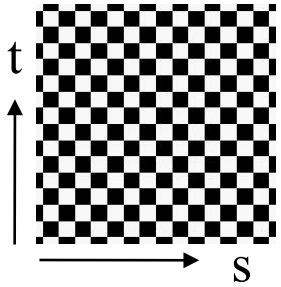
→ Il faut créer une texture de telle sorte qu'elle boucle naturellement (ici, selon les 2 axes), sans qu'on voit les bords.



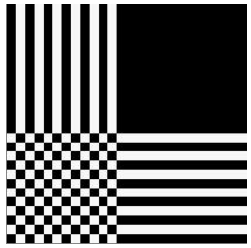
2) Tronquer

Si le mode GL_CLAMP est utilisé, la valeur de la texture aux extrêmes (0 ou 1) est utilisée.

```
glTexParameteri( GL_TEXTURE_2D,
                 GL_TEXTURE_WRAP_S,
                 GL_CLAMP );
```



Texture



GL_CLAMP

C’est le mode qu’on utilisera si la texture ne doit pas boucler.

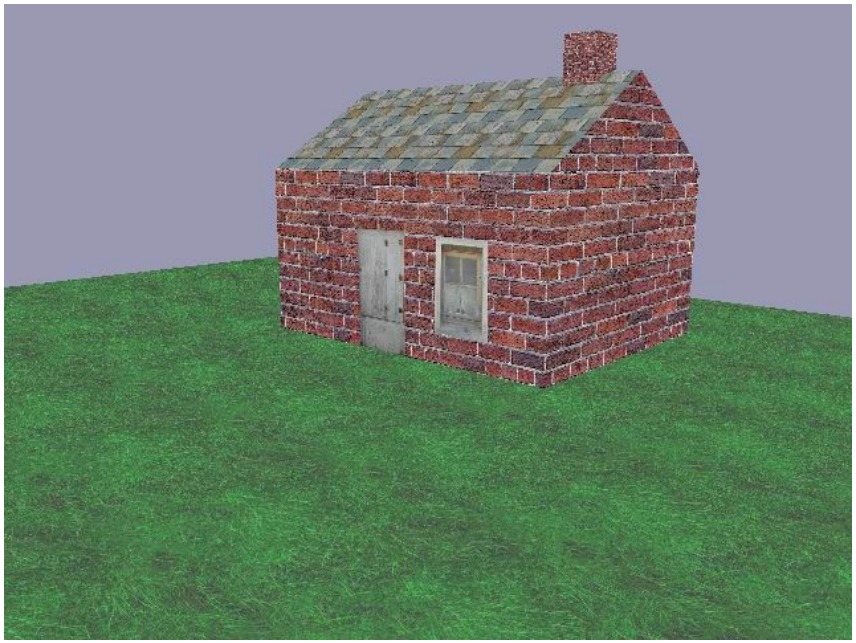
3.3.3 Fonctions de textures

Contrôle la manière selon laquelle la texture est mélangée à la couleur de l'objet.

```
glTexEnvf( GL_TEXTURE_ENV,
           GL_TEXTURE_ENV_MODE,
           param );
```

`param` peut prendre l'une des valeurs suivantes :

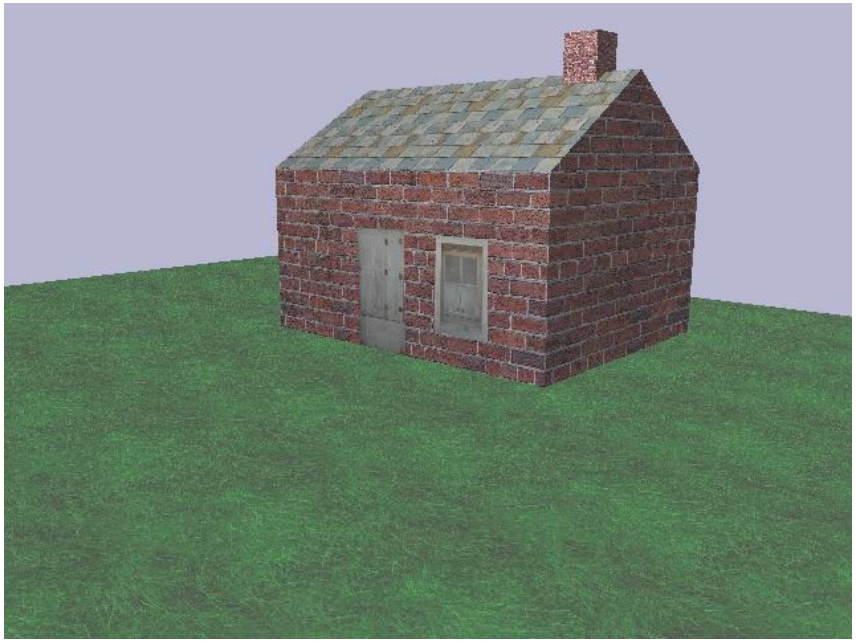
- **`GL_DECAL`** : remplace la couleur par le texel.
- **`GL_MODULATE`** : multiplie le texel par la couleur.



GL_DECAL

Remplace la couleur donnée par l'illumination de Phong par celle du texel.

```
glTexEnvf (GL_TEXTURE_ENV,

GL_TEXTURE_ENV_MODE,
            GL_DECAL);
```



GL_MODULATE

Multiplie la couleur donnée par l'illumination de Phong par celle du texel.

```
glTexEnvf (GL_TEXTURE_ENV,
            GL_TEXTURE_ENV_MODE,
            GL_MODULATE);
```

→ résultats plus réalistes

Exemple complet de lecture d'une texture RGB et de son application sur un triangle

```
// Déclarations de variables

BYTE   *img;
int    largeur, hauteur;
Gluint texture;

// Fin des déclarations de variables
```

```
// Création d'une texture
//    - lecture d'une image
//    - chargement en mémoire vidéo
//    - réglage des paramètres de la texture

glGenTextures(1, &texture);

img = load_tga( "image.tga", &largeur, &hauteur );
if( img != NULL )
{
   glBindTexture(GL_TEXTURE_2D, texture);
   glTexImage2D( GL_TEXTURE_2D, 0, 3,
                 largeur, hauteur,
                 0, GL_RGB, GL_UNSIGNED_BYTE, img);
   delete[] img;
}
```

```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER,
                GL_LINEAR);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_WRAP_S,
                GL_REPEAT);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_WRAP_T,
                GL_REPEAT);

glTexEnvf(GL_TEXTURE_ENV,
          GL_TEXTURE_ENV_MODE,
          GL_MODULATE);

// Fin de la création d'une texture
```

```
// Utilisation d'une texture

// Si le mode "texture" avait été désactivé,
// on l'active :
glEnable(GL_TEXTURE_2D);

glBegin(GL_TRIANGLES);
   glTexCoord2f(0.0f,0.0f);
   glVertex3f(4.0f, 5.0f, 0.0f);
   glTexCoord2f(1.0f,0.0f);
   glVertex3f(10.0f, 5.0f, 0.0f);
   glTexCoord2f(0.0f,1.0f);
   glVertex3f(4.0f, 12.0f, 0.0f);
glEnd();


// Fin de l'utilisation d'une texture
```

Exemple complet de lecture d'une texture RGB et de son application en mip-mapping sur un triangle

```
// Déclarations de variables

BYTE   *img;
int    largeur, hauteur;
GLuint texture;

// Fin des déclarations de variables
```

```
// Création d'une texture mip-map
//    - lecture d'une image
//    - chargement en mémoire vidéo en tant que mip-map
//    - réglage des paramètres de la texture

glGenTextures(1, &texture);

img = load_tga( "image.tga", &largeur, &hauteur );
glBindTexture(GL_TEXTURE_2D, texture);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3,
                  largeur, hauteur,
                  GL_RGB, GL_UNSIGNED_BYTE, img);
```

```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_WRAP_S,
                GL_REPEAT);

glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_WRAP_T,
                GL_REPEAT);

glTexEnvf(GL_TEXTURE_ENV,
          GL_TEXTURE_ENV_MODE,
          GL_MODULATE);

// Fin de la création d'une texture mip-map
```

```
// Utilisation d'une texture

// Si le mode "texture" avait été désactivé,
// on l'active :
glEnable(GL_TEXTURE_2D);

glBegin(GL_TRIANGLES);
   glTexCoord2f(0.0f,0.0f);
   glVertex3f(4.0f, 5.0f, 0.0f);
   glTexCoord2f(1.0f,0.0f);
   glVertex3f(10.0f, 5.0f, 0.0f);
   glTexCoord2f(0.0f,1.0f);
   glVertex3f(4.0f, 12.0f, 0.0f);
glEnd();


// Fin de l'utilisation d'une texture
```

4. Fonctions OpenGL avancées

- Modification d'une texture
- Gestion de textures en mémoire
- Alpha-Blending
- Multitexturing
- Environment mapping
- Bump mapping
- Displacement mapping
- Texture HDR
- Compression de texture
- Stencil buffer

4.1 Modification d'une texture

Il est possible de modifier tout ou partie d'une texture.
→ Évite d'avoir à recréer une nouvelle texture.



Exemple :

on veut modéliser une TV avec un cube, en plaquant sur une de ses faces une texture dont on modifie le contenu au cours du temps avec les images d'un film.

```
void glTexSubImage2D( GLenum target, GLint level,
                      GLint x, GLint y,
                      GLint w, GLint h,
                      GLenum format, GLenum type,
                      GLvoid *texels );
```

target : **GL_TEXTURE_1D**, **GL_TEXTURE_2D**, **GL_TEXTURE_3D**

level : **0** sans mip-mapping

x, y : coordonnées de la sous-région à modifier

w, h : dimensions de la sous-région de la texture

format : **GL_RGB**, **GL_RGBA**, ...

type : type des éléments des texels (**GL_UNSIGNED_BYTE**)

texels : tableau de texels source

4.2 Gestion de textures en mémoire

Les textures sont chargées en mémoire vidéo. Quand il n'y a plus de place, OpenGL supprime des textures.

4.2.1 Vérification de validité

On peut savoir si un entier correspond à un identificateur valide de texture :

```
GLboolean glIsTexture (GLuint texture);
```

Retourne **GL_TRUE** ou **GL_FALSE**.

4.2.2 Vérification de présence

On peut savoir si une texture est résidente en mémoire :

```
GLboolean glAreTexturesResident(
          GLsizei n,
          Gluint *texNums,
          GLboolean *residences);
```

n : nombre de textures.

texNums : tableau des identificateurs des textures.

Residences : tableau de booléens indiquant si les textures correspondantes de **texNums** sont résidentes ou non.

Si toutes les textures sont résidentes, la fonction retourne TRUE, sinon FALSE.

4.2.3 Ordre de priorité

On peut spécifier un ordre de priorité entre 0 et 1 pour chaque texture. Les textures d'ordre les plus faibles seront les premières supprimées.

```
glPrioritizeTextures(
            GLsizei n,
            GLuint *texNums,
            GLclampf *priorities);
```

n : nombre de textures.

texNums : tableau des identificateurs des textures.

Residences : tableau de réels indiquant l'ordre de priorité des textures correspondantes de texNums.

4.2.4 Suppression de textures

Lorsque des textures sont inutilisées, on peut les supprimer de la mémoire vidéo avec la fonction :

```
glDeleteTextures(Gluint n, Gluint *tab_text);
```

Où **tab_text** est un tableau contenant les **n** identificateurs des textures à supprimer.

4.3 Alpha blending

4.3.1 La composante Alpha

La composante alpha (4ème valeur, se rajoute à R,G,B) pour une couleur est une mesure de son opacité. Sa valeur va de 0.0 (complètement transparent) à 1.0 (complètement opaque).

→ Simulation d'objets translucides (eau, fumée, …), mélange d'images.

On peut utiliser comme textures semi-transparentes des images 32 bits RGBA (formats : TGA, BMP, …)

Il y a deux comportements possibles :

- **Transparence** (`GL_ALPHA_TEST`)
- **Translucidité** (`GL_BLEND`)

4.3.2 Transparence (`GL_ALPHA_TEST`)

Les pixels seront affichés ou non (→ comportement binaire) en fonction de la valeur de leur composante alpha et du mode spécifié par **`glAlphaFunc()`** :

```
glAlphaFunc(mode, valeur);
```

Teste la composante alpha de chaque pixel à afficher par rapport à **`valeur`** selon **`mode`**. Si le test échoue, le pixel ne sera pas affiché.

Valeurs possibles de **`mode`** :

`GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS`

La valeur par défaut est **`GL_ALWAYS`** : tous les pixels sont affichés.

Exemple :

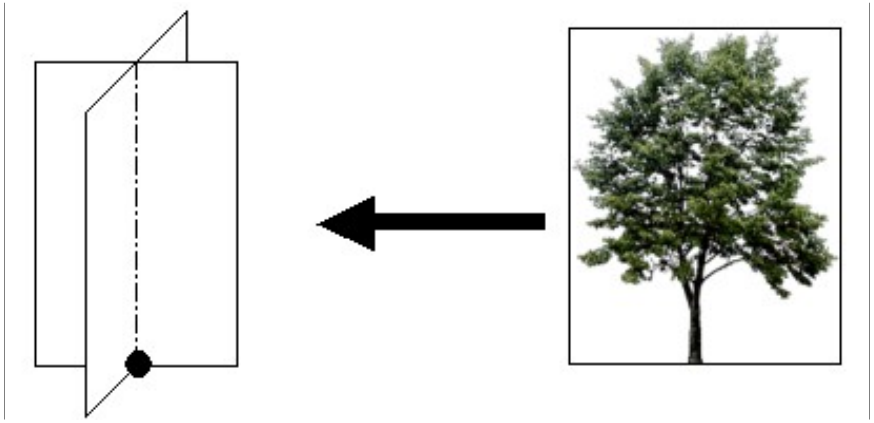
```
glAlphaFunc(GL_GREATER, 0);

glEnable(GL_ALPHA_TEST);

// Affichage des polygones texturés
// avec la texture RGBA

glDisable(GL_ALPHA_TEST);
```

Les polygones affichés auront des trous là où les texels de leur texture n'ont pas de composante alpha > 0.

Remarque :

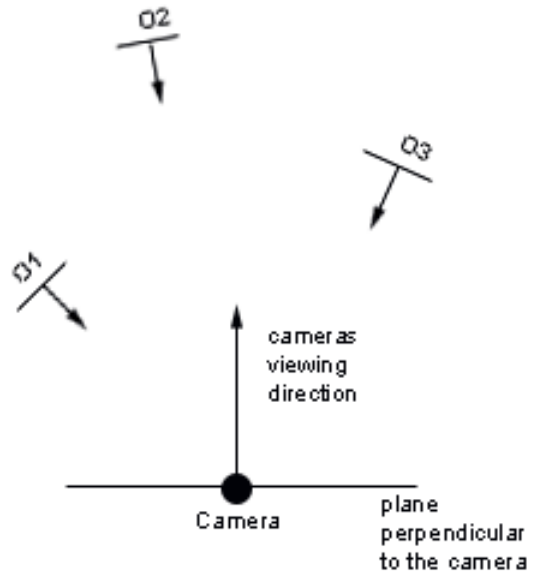
Technique très utilisée pour afficher des arbres dans des jeux ou des simulateurs :

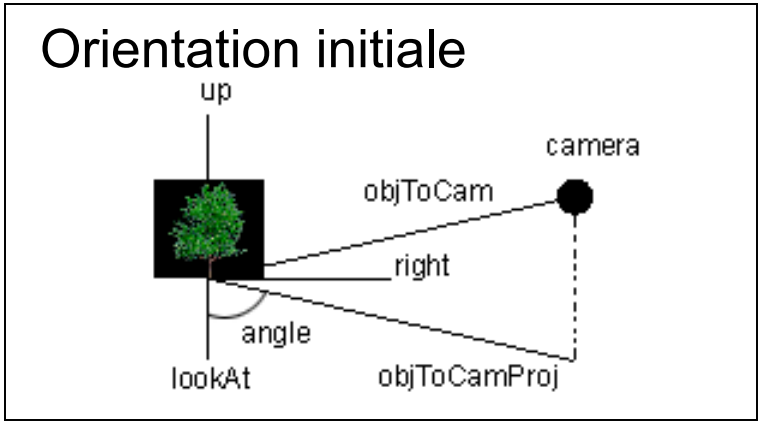
Plutôt que d'afficher un arbre modélisé en 3D avec des milliers de triangles (→ lent à afficher, surtout pour toute une forêt), on utilise un ou deux quadrilatères sur lesquels on plaque une texture d'arbre.

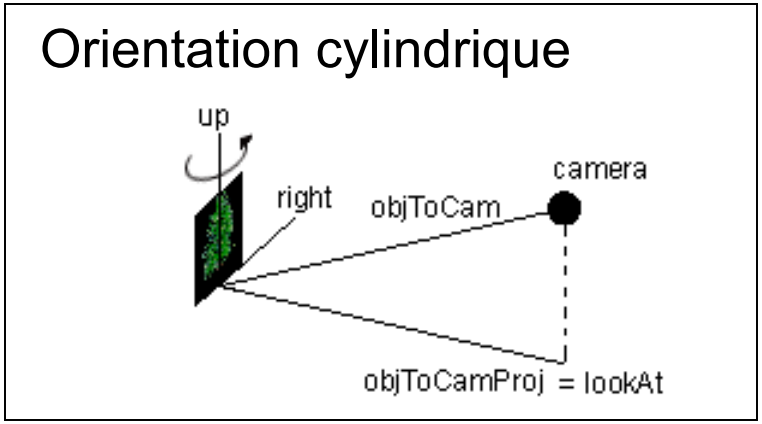


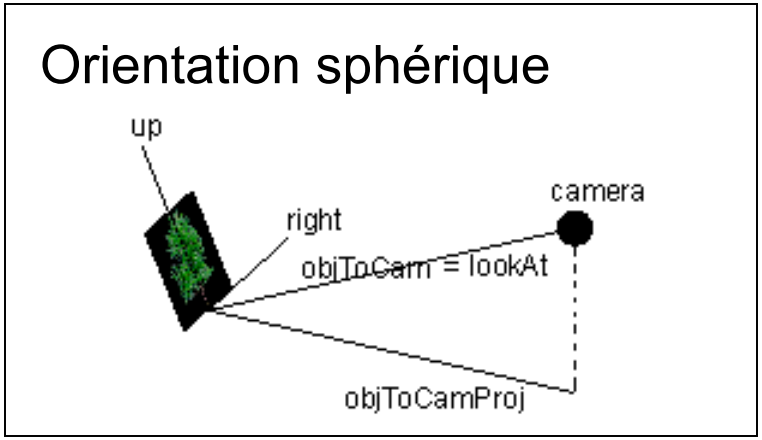
Billboarding

C'est un quadrilatère texturé (texture RGBA) qui est toujours orienté face à l'observateur.

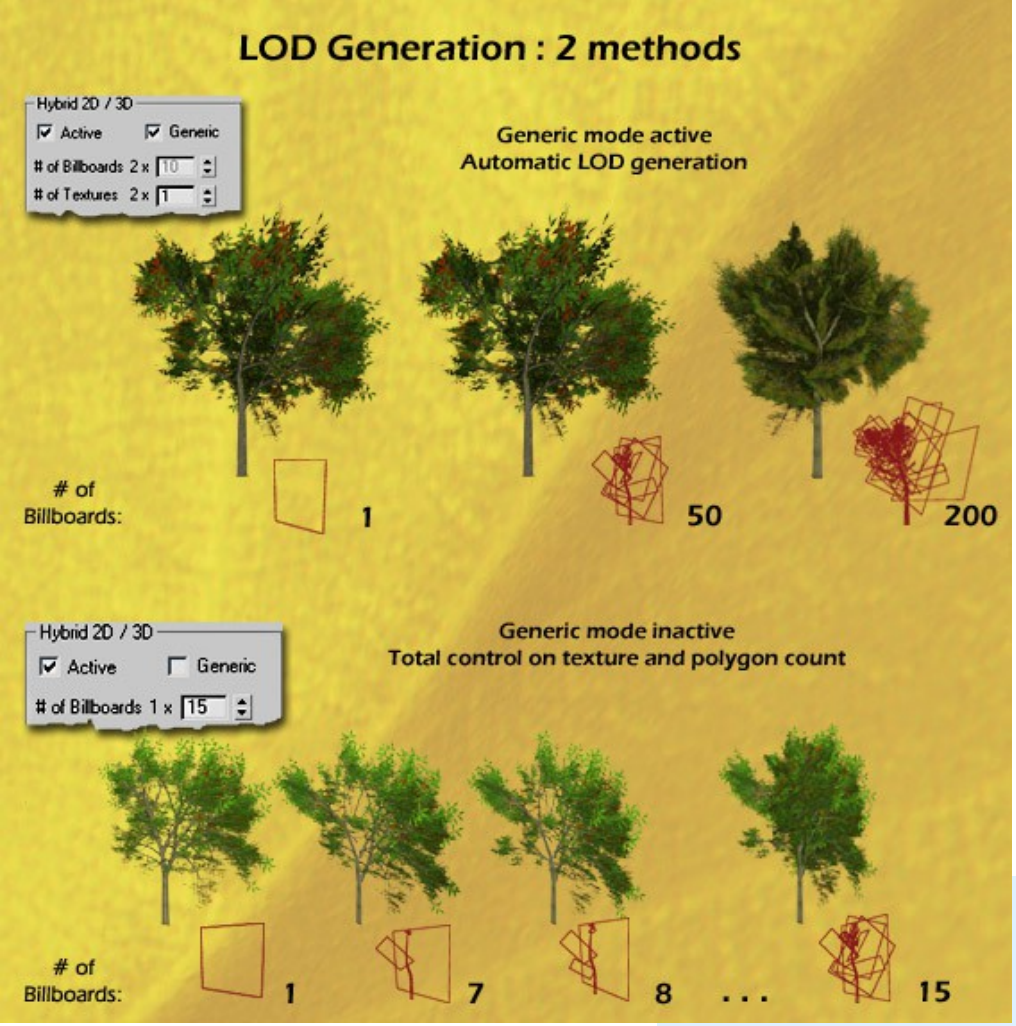








http://www.lighthouse3d.com/opengl/billboarding/



Pour représenter un arbre dans un jeu, on utilise généralement plusieurs billboards.

Le nombre de billboards dépend de la distance à laquelle l'arbre est vu.

http://www.bionatics.fr

4.3.3 Translucidité (`GL_BLEND`)

La couleur de la texture se mélange à la couleur du reste de la scène dans des proportions données par la valeur alpha de chaque texel de la texture.

La manière de mélanger est définie par la fonction :

```
glBlendFunc(GL_SRC_ALPHA, mode);
```

Le **`mode`** le plus utilisé est **`GL_ONE_MINUS_SRC_ALPHA`** .

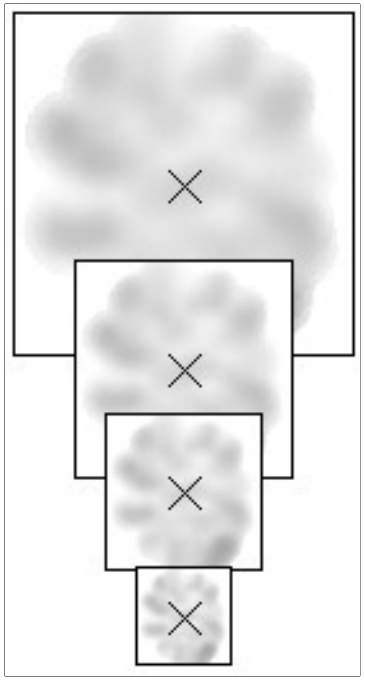
Exemple :

```
glBlendFunc(GL_SRC_ALPHA,
                GL_ONE_MINUS_SRC_ALPHA);

glEnable(GL_BLEND);

// Affichage des polygones
// texturés avec la texture RGBA

glDisable(GL_BLEND);
```

Les polygones affichés seront plus ou moins translucides.

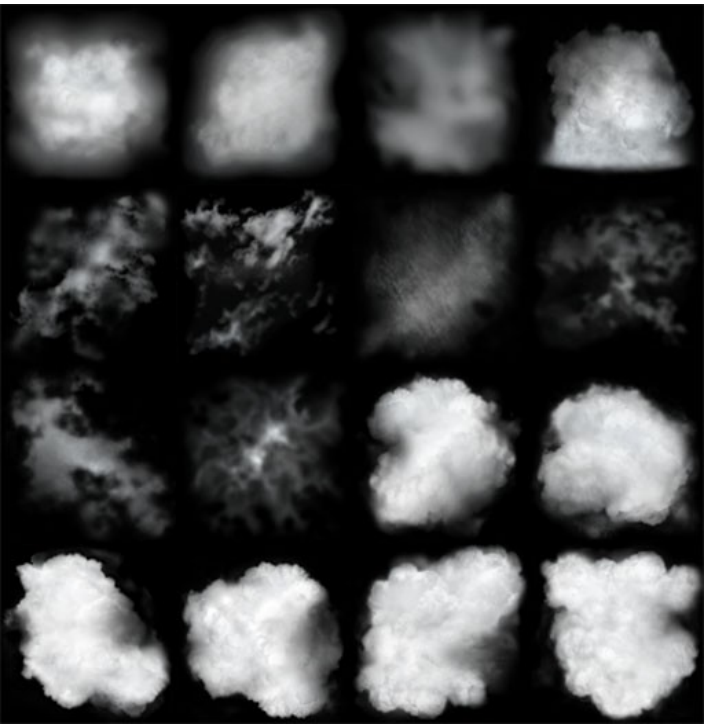
Remarque :

Technique très utilisée pour réaliser des effets de fumée, de flamme, d'explosion, de lumière, des nuages, etc. en plaquant une texture translucide sur un quadrilatère.




SKILLED
KÄRCHER
DecoRag
MORE THAN JUST A RUG STORE


L
485829
Y



Dans *Flight Simulator* (Microsoft), des nuages 3D sont obtenus grâce à des amas de centaines de billboards ayant des textures et des tailles différentes.

La couleur de ces textures est modulée : plus sombre pour les billboards à la base du nuage, plus claire pour ceux au sommet pour simuler l'atténuation de la lumière du soleil qui traverse le nuage.

4.4 Multi-texturing

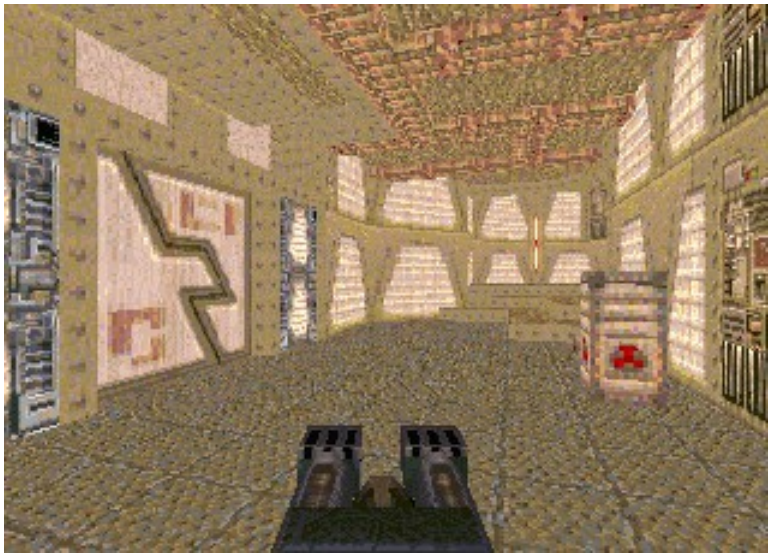
On a vu comment plaquer **une** texture sur un polygone.

Mais les cartes graphiques permettent l'application de **plusieurs** textures sur un même polygone, en combinant ces textures selon différentes opérations.

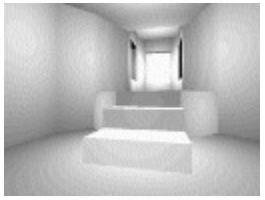
Ceci peut très facilement être fait au moyen de ***shaders*** (voir chapitre 6)

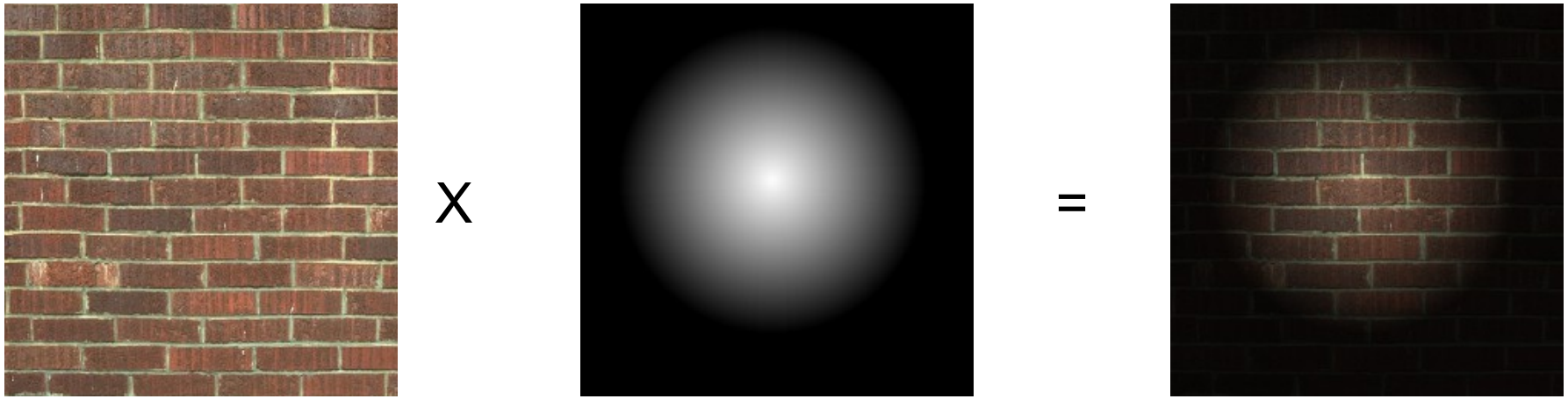
Application : « *lightmaps* »

Lors de l'affichage, mélange de la texture avec une texture stockant une illumination précalculée pour obtenir une texture éclairée. Permet de simuler plus de sources de lumière (OpenGL limité à 8 sources) ou des ombres portées.

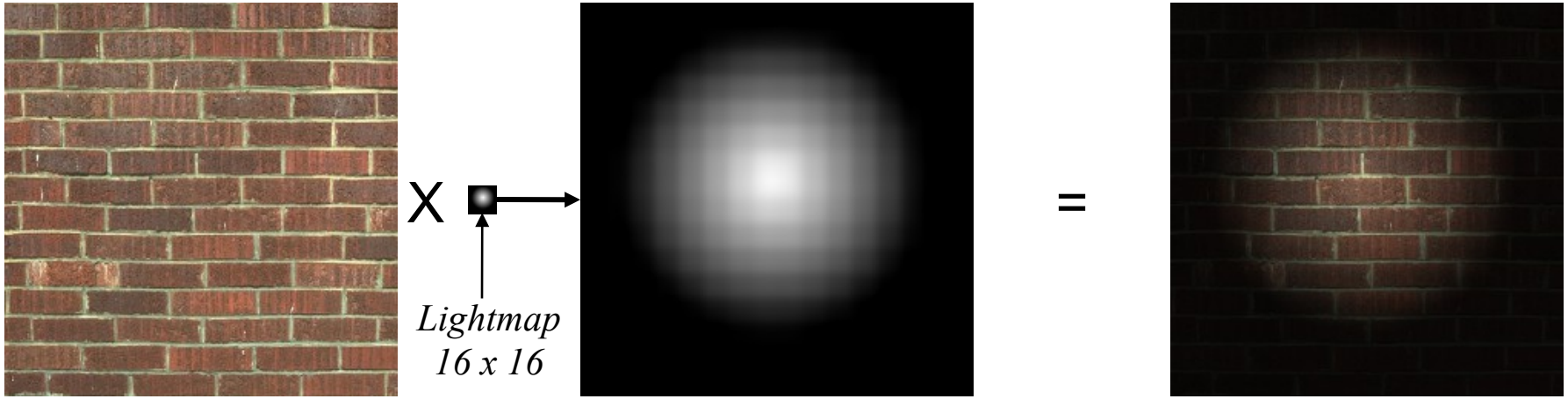


Technique très utilisée dans les jeux depuis Quake.



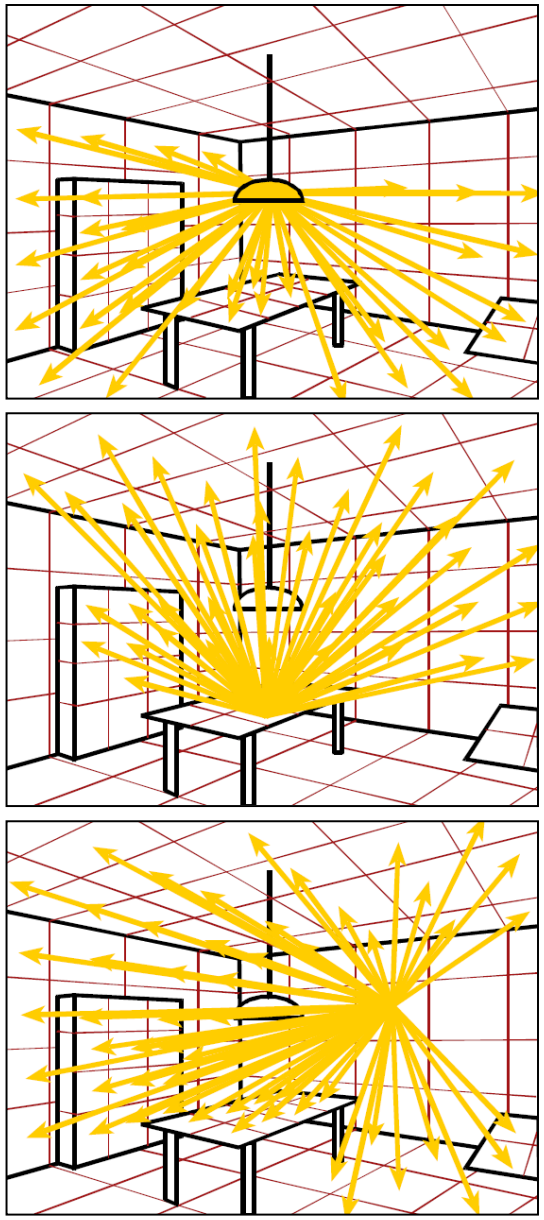


Texture de base *Lightmap* *Multitexturing*



Agrandissement

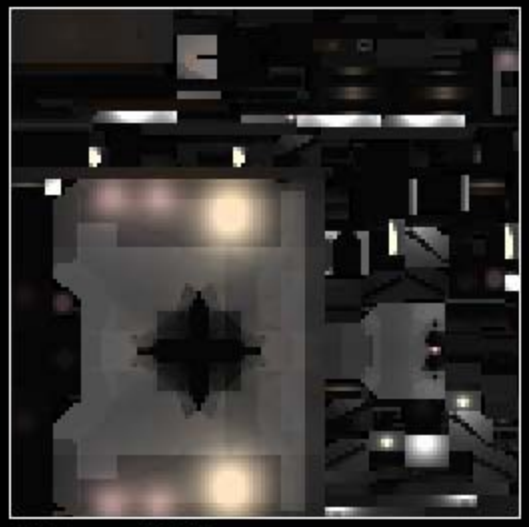
- Les lightmaps nécessitent de très faibles résolutions.
- Lightmap : seulement luminance (→ un seul octet par texel).
- Pas la peine d'avoir plusieurs versions éclairées différemment de la même texture de base.



Calcul des échanges de lumière entre les carreaux

Les lightmaps sont souvent précalculées avec une méthode de **radiosité** (illumination globale).

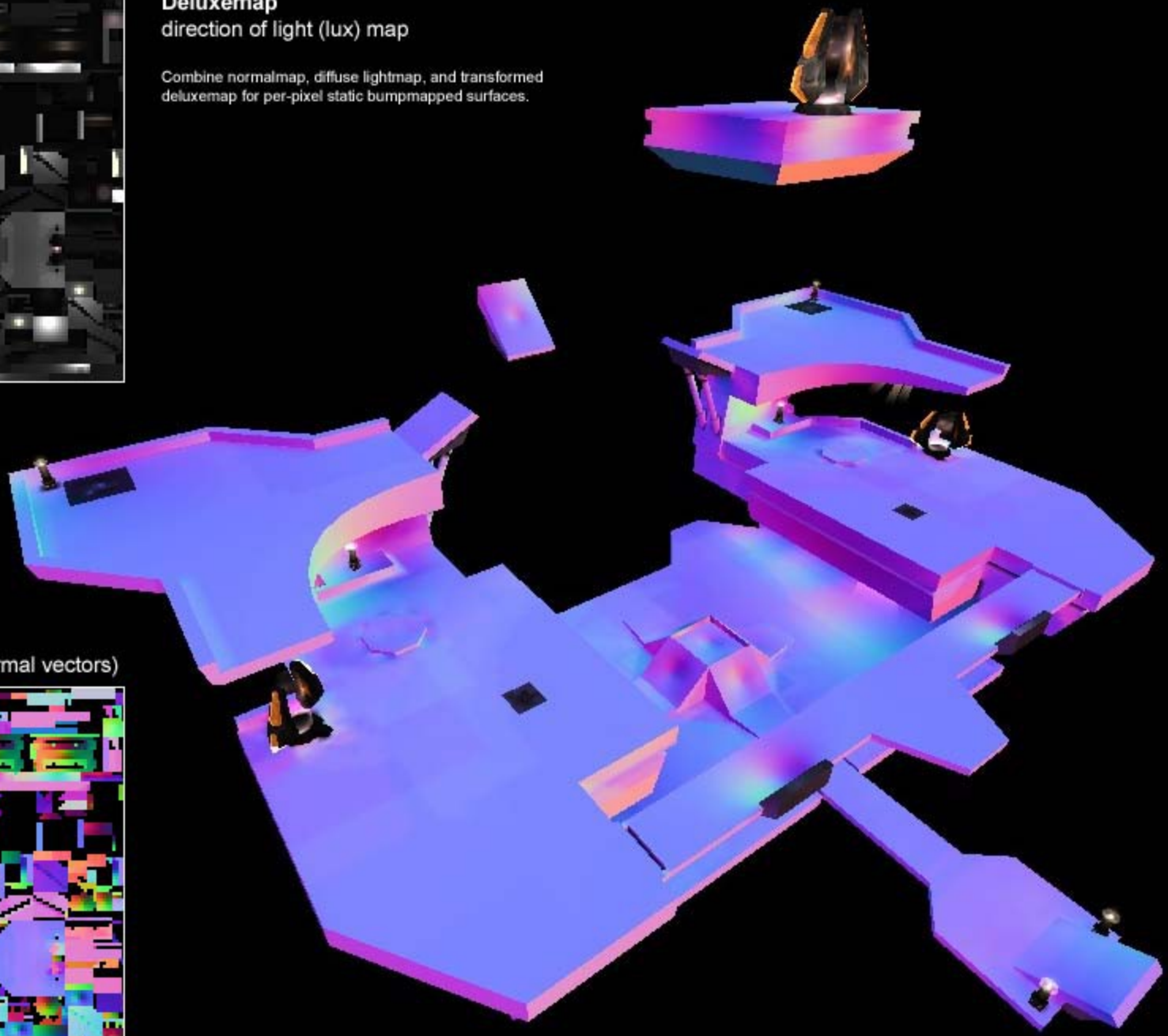
Son principe : décomposer les surfaces de la scène 3D en carreaux («*patchs*»). Certains carreaux correspondant aux sources de lumière ont une intensité lumineuse. On calcule ensuite les échanges lumineux entre les carreaux.



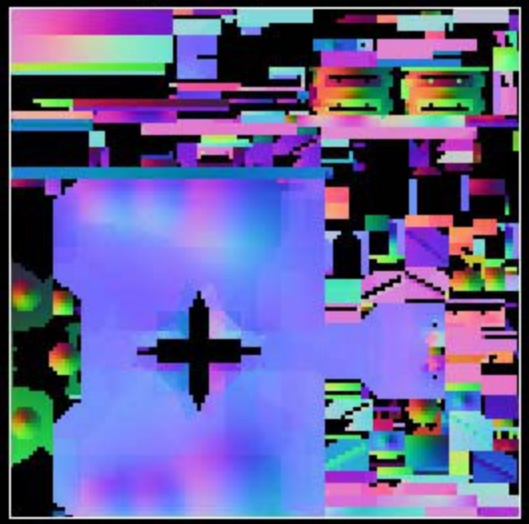
Lightmap (RGB)

Deluxemap
direction of light (lux) map

Combine normalmap, diffuse lightmap, and transformed deluxemap for per-pixel static bumpmapped surfaces.



Deluxemap (world-space normal vectors)



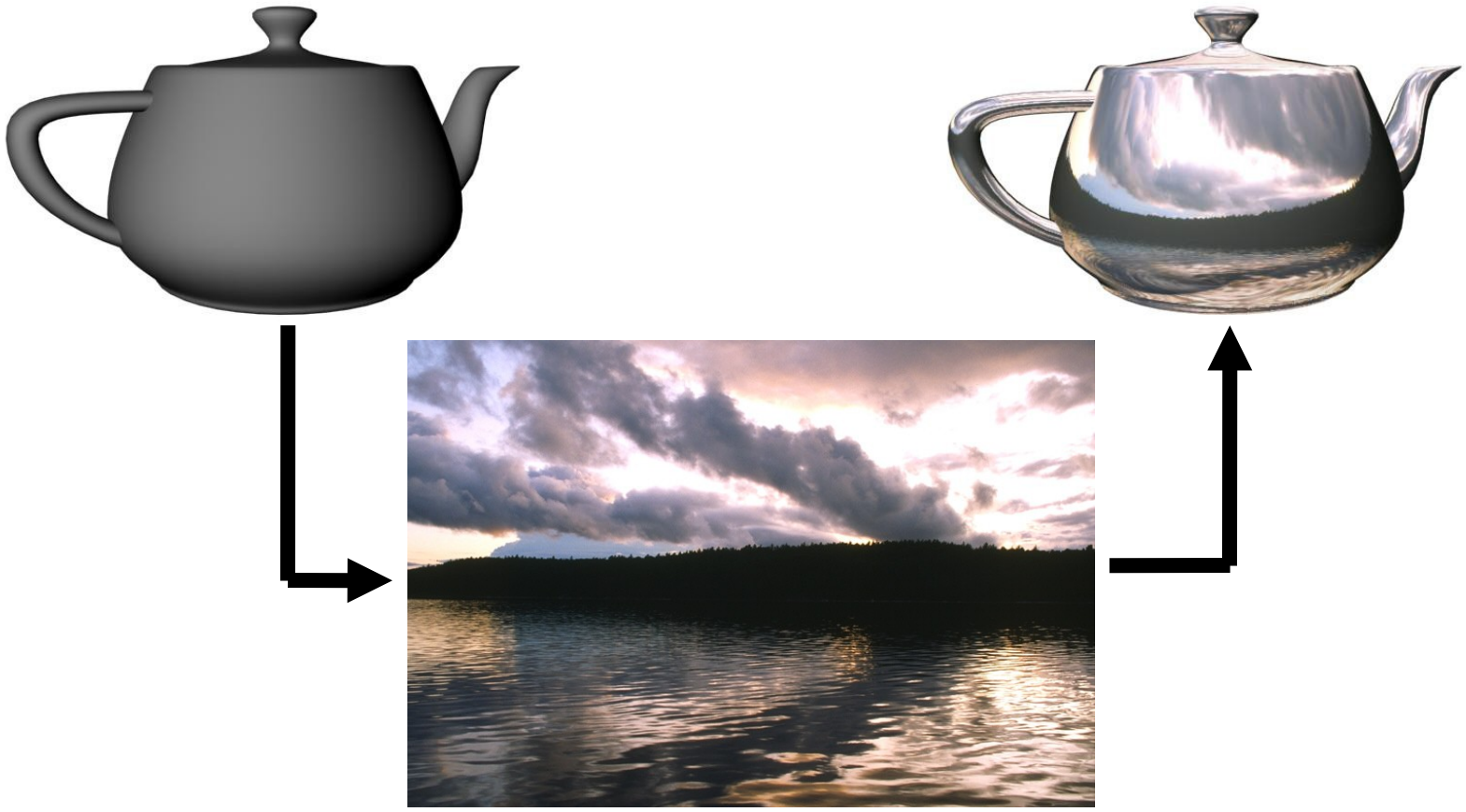
Artefact : effet d’escalier

L’agrandissement exagéré des textures de lightmap rend très apparents les texels, surtout sur les bords des ombres

→ effets de marches d’escalier.

4.5 Environment mapping

Permet de simuler des surfaces réfléchissantes (chrome, métal, …) au moyen d'un plaquage de texture.



Principe

Les coordonnées de texture de l'objet réfléchissant sont calculées dans une texture représentant l'environnement selon la position de l'observateur.

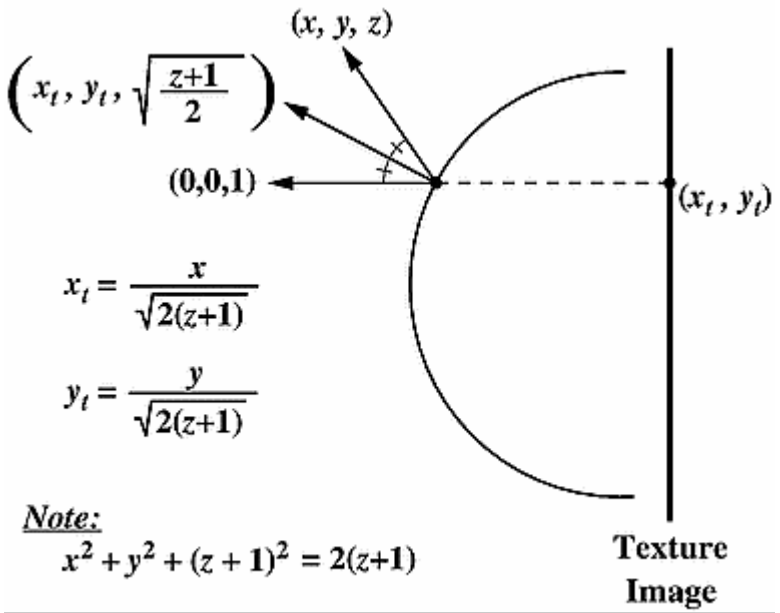
Plusieurs méthodes intégrées dans les cartes graphiques :

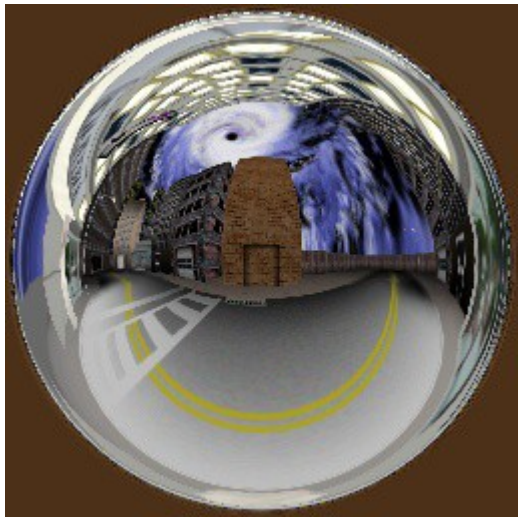
- Sphere mapping
- Cube mapping
- …

Sphere mapping

Utilisation d'une texture représentant une sphère parfaitement réfléchissante.

Pour chaque sommet de l'objet 3D à texturer, on calcule ses coordonnées de texture (x_t,y_t) à partir de la normale (x,y,z) en ce sommet.





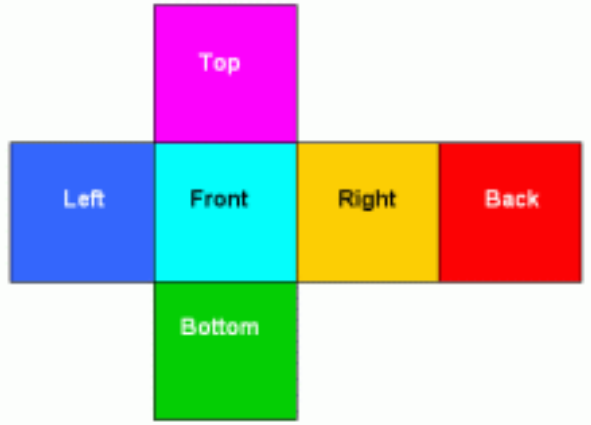
Texture d'environnement

Exemple de sphere mapping

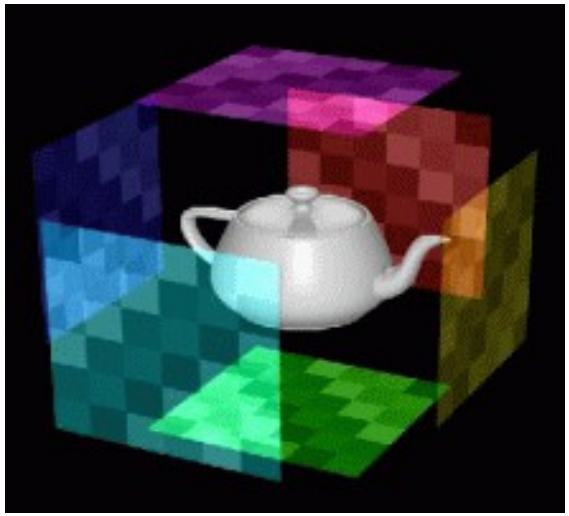
```
// A faire une seule fois (initialisation)
glTexGenfv(GL_S, GL_SPHERE_MAP, 0);
glTexGenfv(GL_T, GL_SPHERE_MAP, 0);



// Activer la génération de coordonnées de texture
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);

// Afficher l'objet recevant l'environment mapping
// ...

// Désactiver la génération de coordonnées de texture
glDisable(GL_TEXTURE_GEN_S);
glDisable(GL_TEXTURE_GEN_T);
```

Cube mapping

L'objet à texturer se trouve dans un cube dont les 6 faces ont des textures correspondant au monde environnant l'objet.



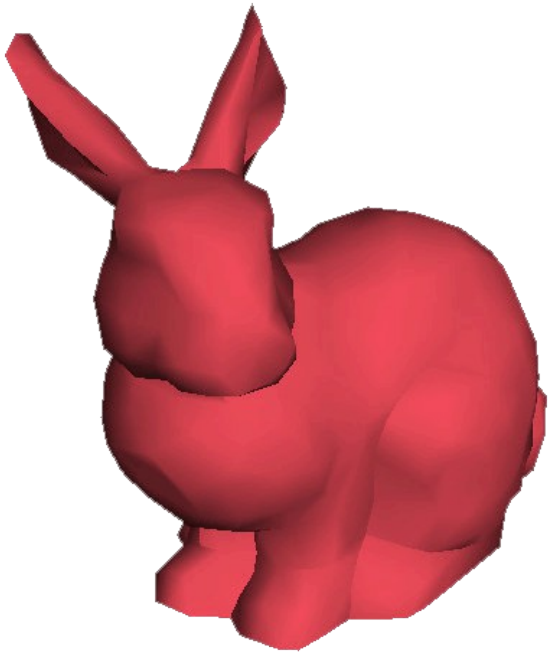
Donne de meilleurs résultats que le sphere mapping, mais est plus coûteux (6 images au lieu d'une seule).



4.6 Bump mapping

Lissage de Gouraud : au lieu de calculer l'illumination d'un triangle avec une seule normale, on illumine les sommets avec une **normale moyennée** et on interpole les couleurs sur toute la face.
→ On obtient l'illusion d'une surface douce, avec des normales qui varient doucement.

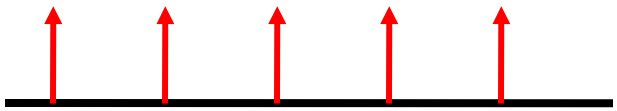
Flat shading (une normale par triangle)



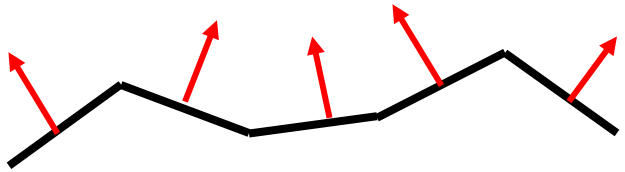
Lissage de Gouraud (une normale moyennée par sommet)

Le **Bump mapping** repose sur cette idée de modification des normales pour modifier l'illumination de la surface

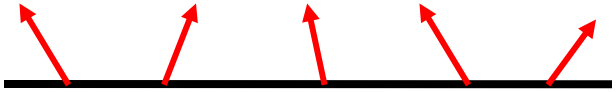
Il permet de simuler une géométrie complexe à partir d'une simple surface dont on modifie l'orientation de la normale, selon les orientations des normales de la surface à simuler.



Surface simple

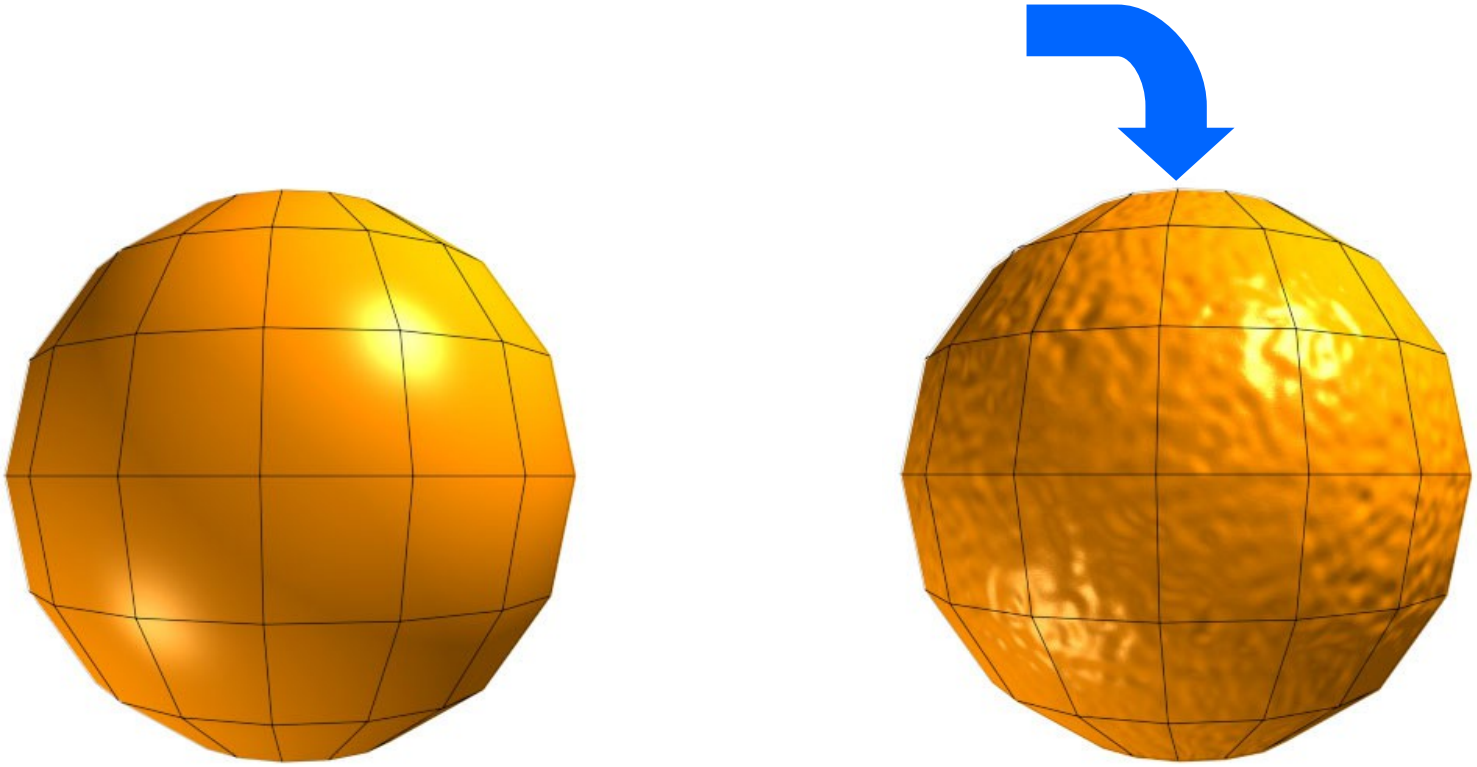


Véritable géométrie à simuler



Surface simple avec bump-mapping

Perturbation des normales
avec une texture de bump-map



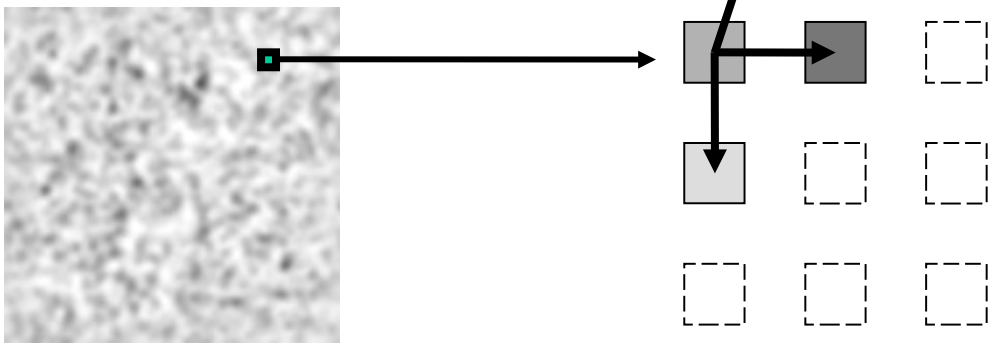
Sans bump-mapping

Avec bump-mapping

Ces deux objets ont **exactement** la même géométrie, le même nombre de polygones, seule leur illumination est différente.

Pour chaque pixel à afficher, on veut modifier la normale N utilisée (le calcul de l'illumination se fait pour chaque pixel). On utilise une image en niveaux de gris pour calculer une normale N_b qui sera rajoutée à la normale N pour la perturber.

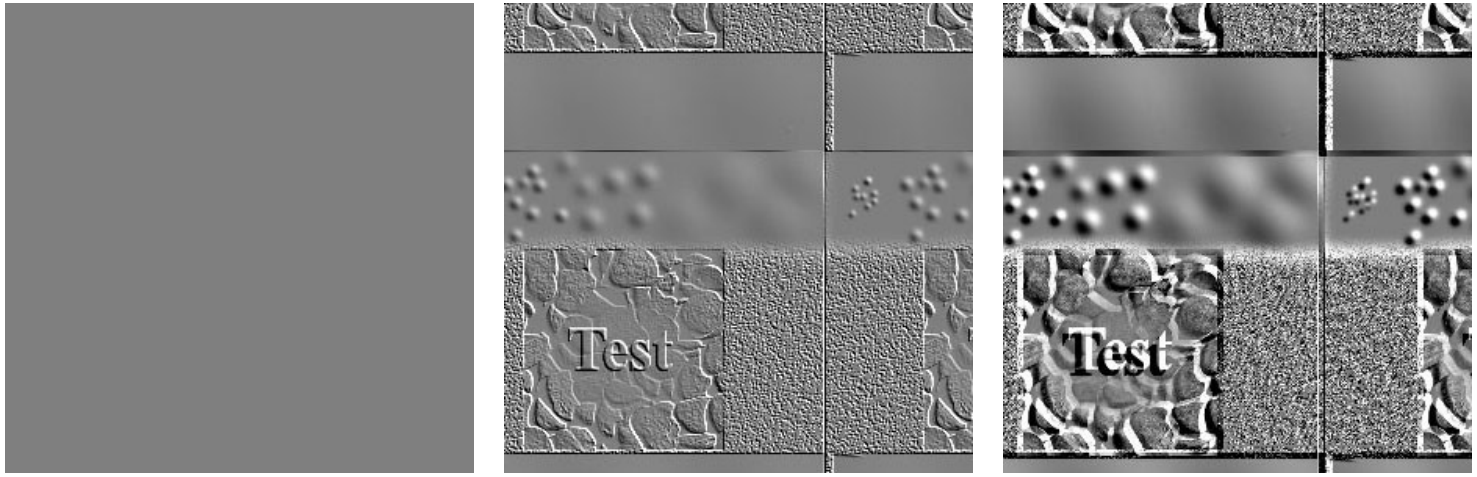
L'image en niveaux de gris est une carte de hauteurs. A partir de 3 pixels, on détermine la direction du vecteur N_b (produit vectoriel).



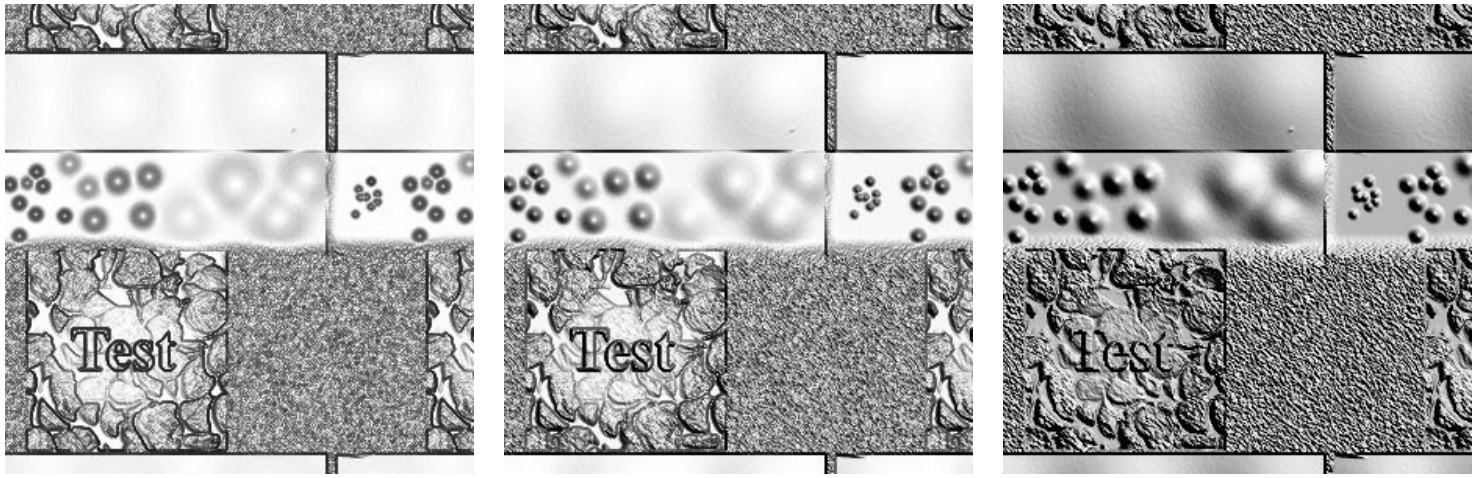
Carte de hauteurs utilisée pour faire du bump mapping

Plusieurs techniques de bump mapping sont apparues dans les cartes graphiques, plus où moins rigoureuses (et donc plus ou moins coûteuses en temps de clacul) :

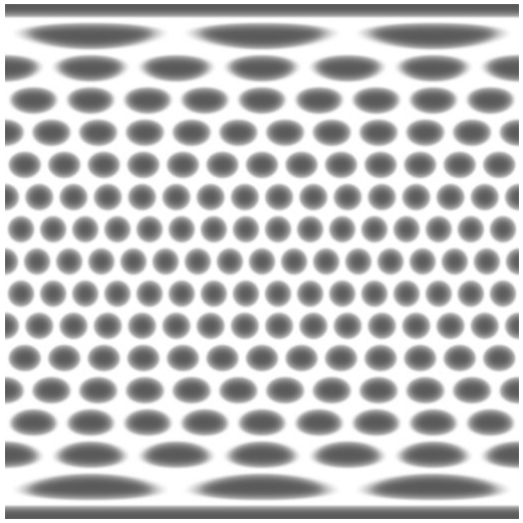
Emboss bump mapping



Dot product bump mapping



Maintenant, on calcule le bump mapping grâce aux shaders.

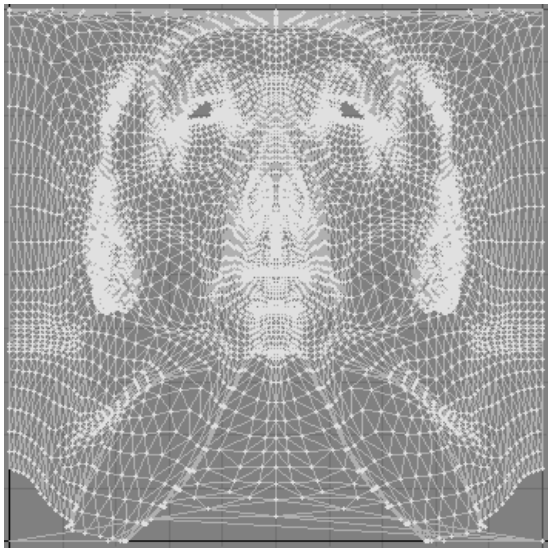


4.7 Normal maps

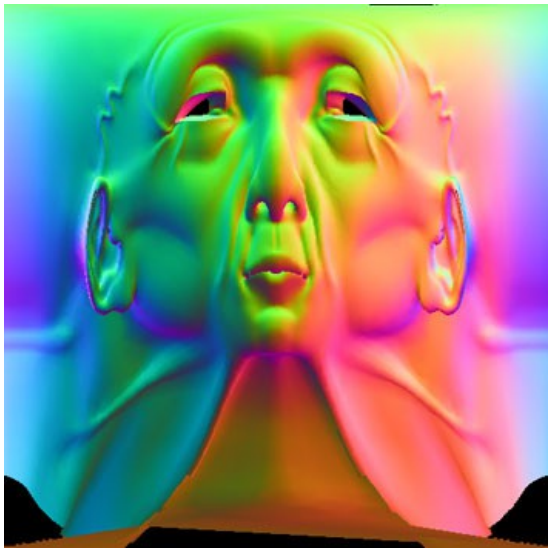
Technique de stockage des normales d'une surface dans une texture, utilisée selon un principe de bump-mapping.

La différence par rapport au bump mapping est que ce sont des images en couleur 24 bits qui sont utilisées (→ 3x plus de place) : les composantes (R,V,B) des pixels servent à coder les composantes (x,y,z) des normales.

Géométrie 3D



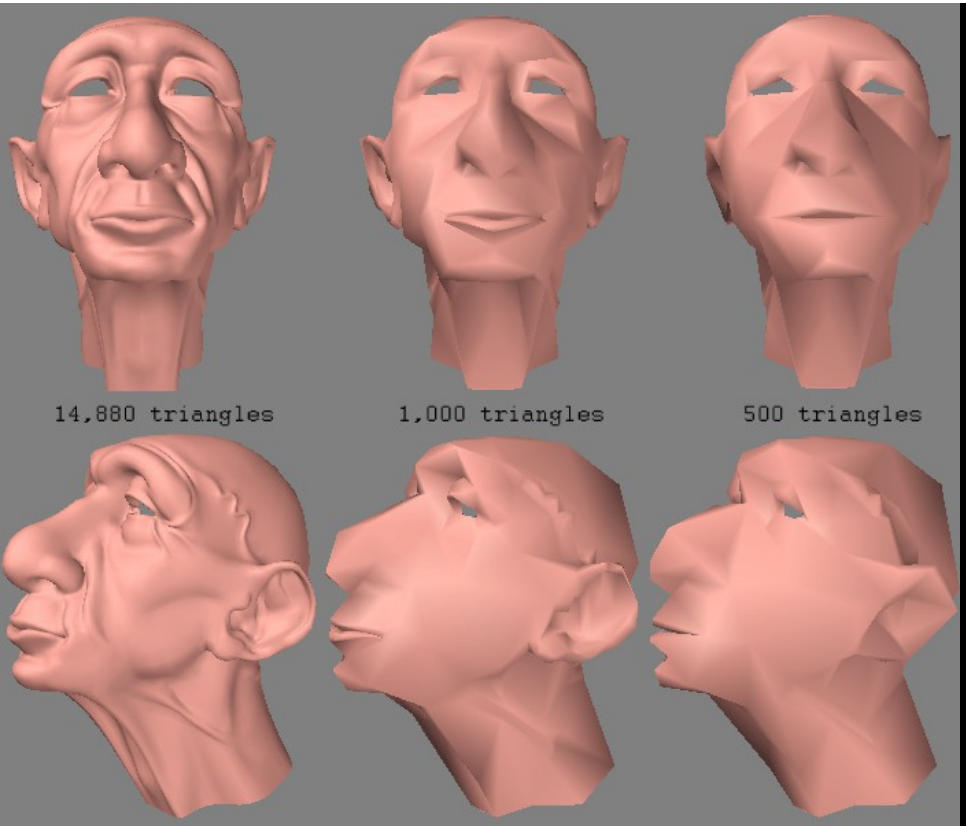
Projection de la géométrie

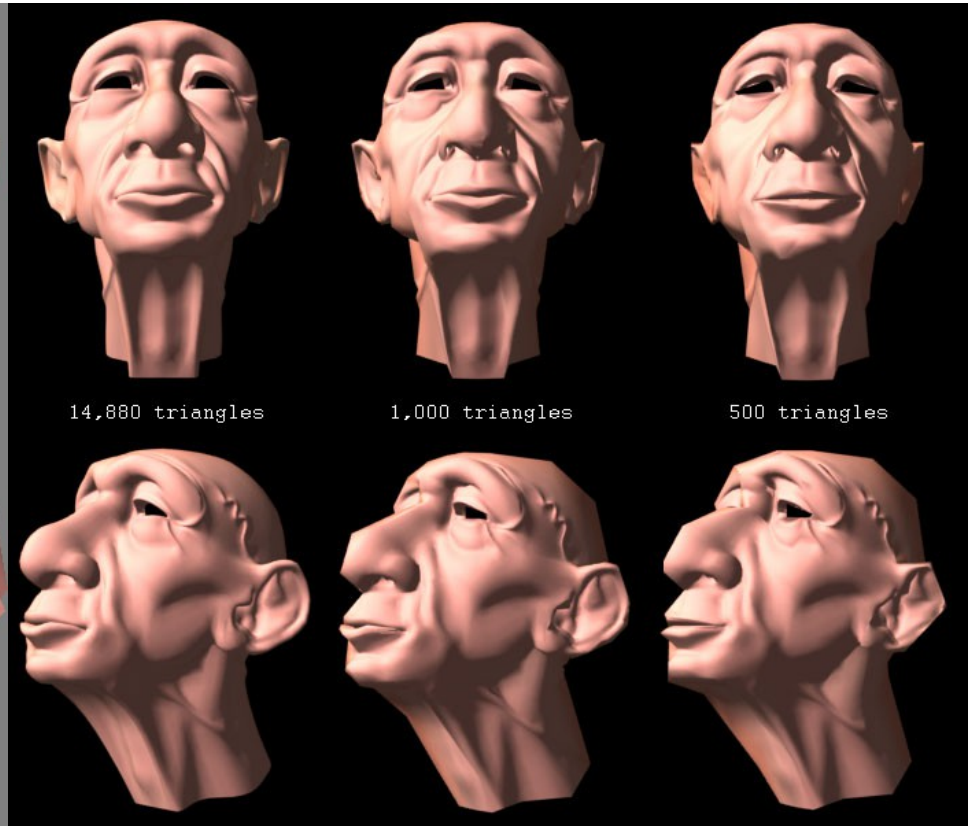


Normal map

Exemple d'application

Partir d'un modèle 3D très détaillé, en calculer les normales et les encoder dans une texture de normal map.
Utiliser ensuite cette normal map sur une version plus grossière du modèle 3D.
(Astuce utilisée dans Doom 3, Half-Life 2, …)

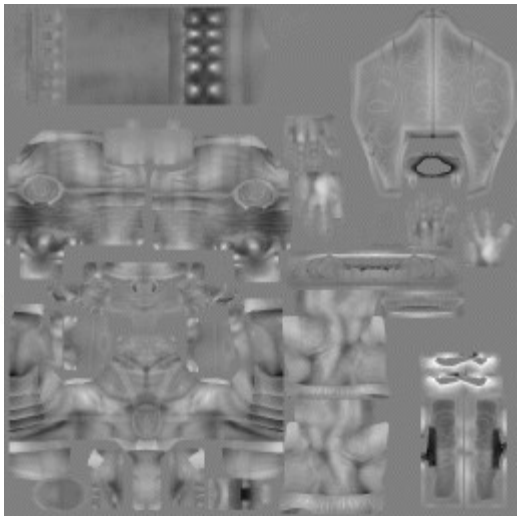




4.8 Displacement mapping

Augmente la complexité (niveau de détails) d'une surface en générant des polygones à partir de textures. Contrairement au bump mapping, la surface est réellement modifiée.

Fonctionnalité supportée par les cartes graphiques.



Displacement map

4.9 Texture HDR

Du fait des fortes variations d'intensité lumineuse rencontrées dans les scènes naturelles, l'œil possède une forte capacité d'**adaptation**.

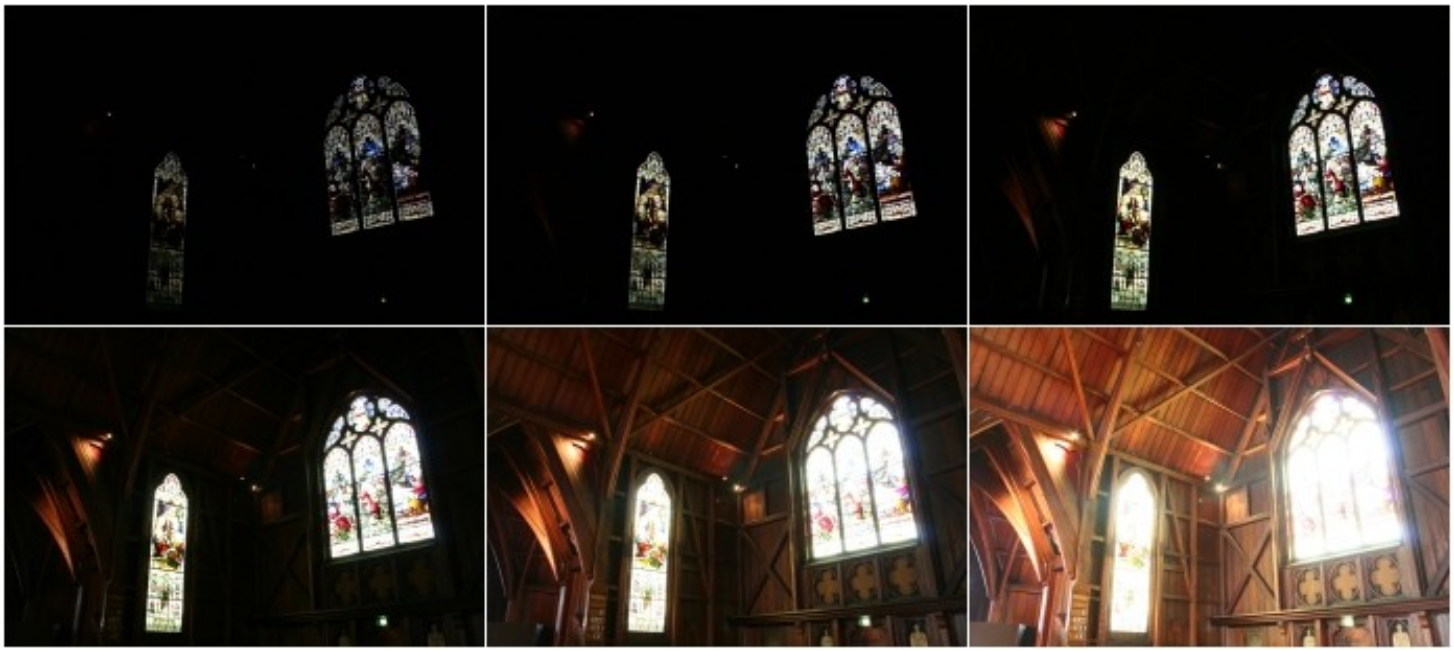
Exemples d'intensités lumineuses :

Soleil à 50° au dessus de l'horizon par ciel clair : 10^5 Lux

Éclairage artificiel intense : 10^3 à 10^4 Lux

Éclairage de bureau : 500 Lux

Pleine lune : 0,2 Lux



8 bits/composantes RVB ne permet pas de coder les écarts très importants de luminosité de la réalité.

→Les textures HDR codent plus d'information (sur 16 ou 32 bits)
HDR = High Dynamic Range

On peut écrire un shader qui calculera avec cette texture HDR la bonne valeur à donner aux pixels des polygones texturés, par exemple de manière à simuler le phénomène d'adaptation de l'œil.

4.10 Compression de texture

4.10.1 Problématique

Les applications 3D (ex: jeux, …) nécessitent des textures de plus en plus fines.

- Consommateur d'espace mémoire.
  Ex: texture RGB de 1024x1024 = 3 Mo

- Gestion de **textures compressées** par les cartes graphiques :

  - **S3TC** (S3)
  - **DXTC** (Microsoft), basé sur S3TC

4.10.2 Le format DXTC

DXTC = 5 formats de compression avec perte.

Nom	Type	Taux de compression
DXT1	RGB	8:1
DXT2	RGBA	4:1
DXT3	RGBA	4:1
DXT4	RGBA	4:1
DXT5	RGBA	4:1

La texture est décomposée en blocs de 16 pixels (4 pixels sur 4). Chaque bloc est compressé et décompressé indépendamment des autres.

4.10.3 Utilisation

Deux possibilités :

- Fournir une image non compressée à la carte graphique, qui se chargera de la compresser.
- Utiliser une image déjà compressée et la fournir à la carte graphique.

Outils de compression disponibles sur le site de NVIDIA :

http://developer.nvidia.com/object/nv_texture_tools.html

+ exemples d'utilisation avec OpenGL.