gamagora

UNIVERSITÉ LUMIÈRE LYON 2
UNIVERSITÉ DE LYON

Synthèse d'images

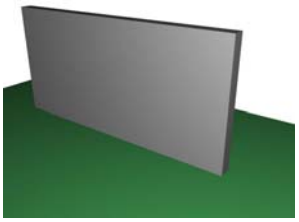
# Texture

Gilles Gesquière

Cours s'appuie sur
- les supports de cours de S. Thon (LSIS, Université Aix Marseille)
- les supports de thèse de F. Graglia (Université Aix-Marseille)

## 1. Problématique

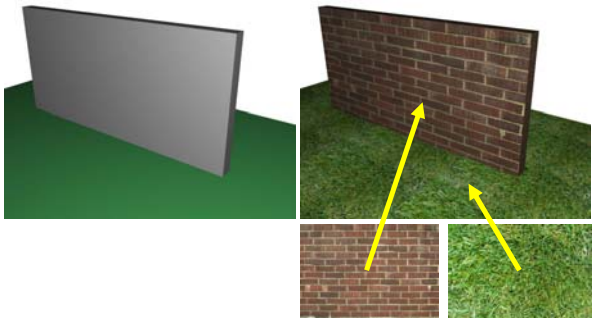
Modèle de Phong : prise en compte des interactions lumière/matière, mais ce n'est pas suffisant.



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

2

Solution : utilisation de **textures**.



3

**Plaquage de textures (« *texture mapping* »)**

→Plaquage d'images 1D, 2D ou 3D sur des primitives géométriques.

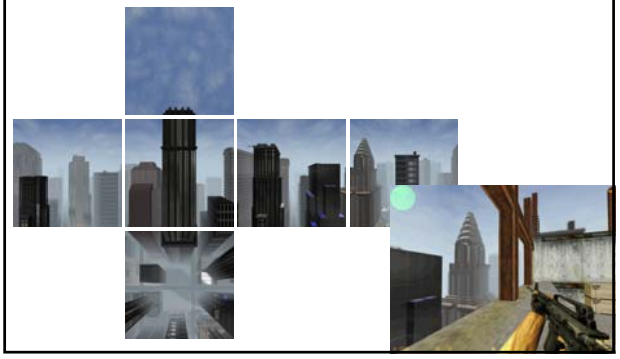
### Objectifs

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

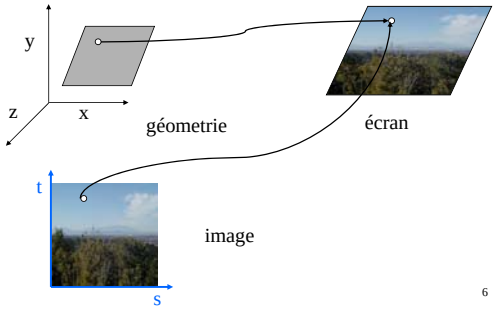
4

**Exemple : « Skybox »**

Simule un ciel (sky) avec un cube (box) sur les 6 faces internes duquel on plaque des textures. L'observateur reste toujours au centre du cube. Très utilisé dans les jeux (Quake, Half-Life, Call of Duty, …)



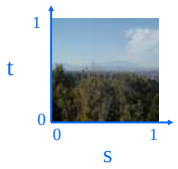
## 2. Principe du plaquage de texture




6

Les textures sont des images qui peuvent être en 1D, 2D, 3D ou 4D. Les coordonnées de l'image sont notées s, t, r, q.

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




7

**Notes :**

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

8

## 3. Utilisation des textures dans OpenGL

**3 étapes :**

1. Spécifier la texture
   - 1.1 Lire ou générer une image
   - 1.2 En faire une texture
   - 1.3 Activer le plaquage de texture
2. Assigner les coordonnées de texture aux points de l'objet 3D
3. Spécifier les paramètres de textures
   - Wraping, filtering, …

9

## 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 );
```

10

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**.

11

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);
```

12

### 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);
```

13

```
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

14

La fonction **`glBindTexture()`** permet de définir l'indice de la **texture courante**.

Cette texture sera utilisée pour toutes les opérations (plaquage, modification de paramètres, …) jusqu'au prochain appel de **`glBindTexture()`**.

15

### 3.1.3 Activer le plaquage de texture

```
glEnable(GL_TEXTURE_2D);
```

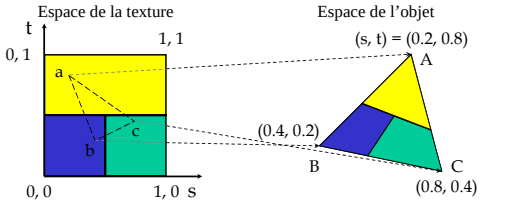
On peut aussi désactiver le plaquage :

```
glDisable(GL_TEXTURE_2D);
```

16

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

Pour plaquer une texture sur un objet géométrique, fournir les coordonnées de texture (normalisés entre 0 et 1).




17

```
glBindTexture(GL_TEXTURE_2D, texture);

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();
```

18

## 3.3 Paramètres de textures

Modes de filtrage

- Réduction, agrandissement
- Mip-mapping

Modes de bouclage

- Répéter, tronquer

Fonctions de textures

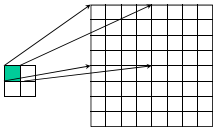
- Comment mélanger la couleur d'un objet avec sa texture

19

### 3.3.1 Modes de filtrage

- Réduction, agrandissement

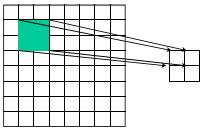
Les textures et les objets texturés ont rarement la même taille (en pixels). OpenGL défini des filtres indiquant comment un texel doit être agrandit ou réduit pour correspondre à la taille d'un pixel.



Texture Polygone

Agrandissement

`GL_TEXTURE_MAG_FILTER`



Texture Polygone

Réduction

`GL_TEXTURE_MIN_FILTER`

20

```
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.

21

`GL_NEAREST`

`GL_LINEAR`

22

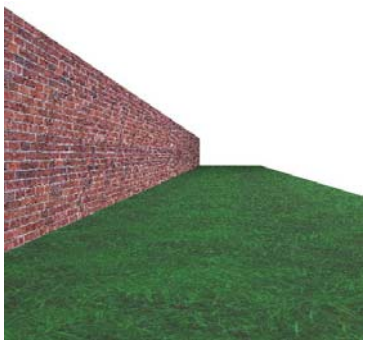
**Ex:**

```
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);
```

23

- Mip-mapping

Problème d'aliassage :



24

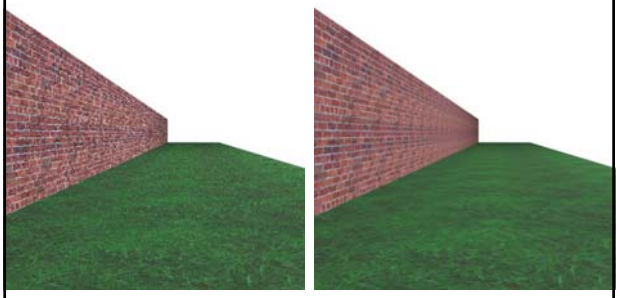
Solution : La technique de **mip-mapping** consiste en un précalcul de plusieurs versions réduites d'une même texture.


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

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

25

→ Permet de réduire les problèmes d'aliassage.



*Sans mip-mapping* *Avec mip-mapping*

26

## Création des textures de mip-mapping

1) Manuelle

On déclare le niveau de mip-map lors de la définition de la texture :

```
glTexImage2D( target, level,
              components,
              width, height,
              border, format, type, *texels );
```

27

2) Automatique

On utilise une fonction de OpenGL qui construit automatiquement les différentes textures de mip-map.

```
gluBuild2DMipmaps( GL_TEXTURE_2D,
                   components,
                   width, height,
                   format, type, *texels );
```

28

Dans les deux cas, il faut ensuite indiquer le mode de filtrage de la texture en indiquant qu'on veut s'en servir en tant que mip-map :

```
gluBuild2DMipmaps(GL_TEXTURE_2D, 3,
                  width, height,
                  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);
```

29

### 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)

30

*- 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

31

*- 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

32

**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 trois valeurs suivantes :

- **GL_DECAL** : remplace la couleur par le texel.
- **GL_MODULATE** : multiplie le texel par la couleur.
- **GL_BLEND** : mélange le texel, la couleur et **env_color**.

33

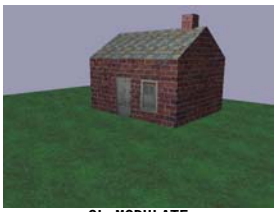
Définition de la couleur **env_color** à utiliser avec la fonction précédente dans le cas où **param = GL_BLEND** :

```
glTexEnvfv( GL_TEXTURE_ENV,
            GL_TEXTURE_ENV_COLOR ,
            env_color );
```

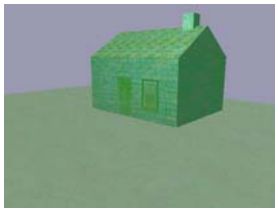
**env_color** est un tableau de 4 float représentant la couleur.

34

GL_DECAL



GL_MODULATE



**GL_BLEND**

avec
**env_color[]={0.0,0.5,0.0,1.0}**

35

**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
```

36

```
// 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;
}
```

37

```
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);
```

38

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

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

39

```
// 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
```

40

**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
```

41

```
// 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);
```

42

```
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);
```

43

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

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

44

```
// 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
```

45

## 4. Fonctions OpenGL avancées

- Fonctions de correction de texture
- Gestion de textures en mémoire
- Alpha-Blending
- Multitexturing
- Bump mapping
- Environment mapping

46

## 4.1 Fonctions de correction de texture

Lors de l'affichage, la texture peut être interpolée :

- Linéairement dans l'espace de l'écran
- En utilisant les valeur de profondeur/perspective (plus lent)

```
glHint( GL_PERSPECTIVE_CORRECTION_HINT, hint )
```

**hint** pouvant prendre les valeurs :

- **GL_DONT_CARE**
- **GL_NICEST**
- **GL_FASTEST**

47

*Effets de distorsion de textures dus à l'absence de correction de perspective (jeux PlayStation)*

48

## 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.

49

### 4.2.1 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.

50

### 4.2.2 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.

51

### 4.2.3 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** indices des textures à supprimer.

52

## 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
- → …

53

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)

54

**4.3.2 Transparence**

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.

55

Les modes de **`glAlphaFunc()`** sont :

- **`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.

56

**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.

57

**4.3.3 Translucidité**

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`** .

58

**Exemple :**

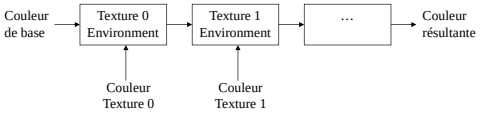
```
glBlendFunc(GL_SRC_ALPHA,
            GL_ONE_MINUS_SRC_ALPHA);

glEnable(GL_BLEND);

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

glDisable(GL_BLEND);
```

Les polygones affichés seront plus ou moins translucides.

59

## 4.4 Multi-texturing

Les cartes graphiques disposent de plusieurs « unités de texture », ce qui permet d'appliquer plusieurs textures sur un même polygone, en combinant ces textures selon différentes opérations.

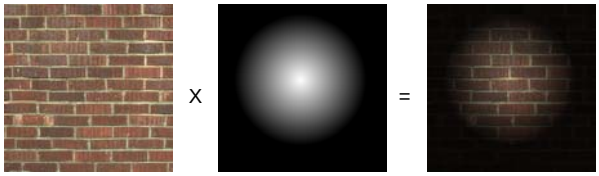



Ces nouvelles fonctionnalités sont disponibles dans OpenGL sous la forme d'extensions.

60

**Application : « *lightmaps* »**

Mélange 2 textures pour obtenir une texture éclairée. Permet de simuler plus de sources de lumière (OpenGL : 8) ou des ombres.



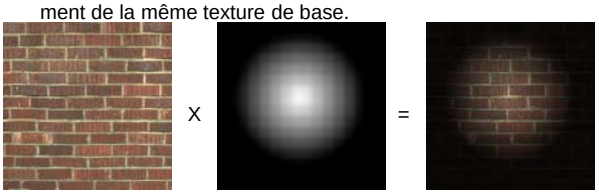
Texture de base | Lightmap | Multitexturing

Technique très utilisée dans les jeux.

61

**Intérêt :**

- 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.



Texture de base | Lightmap (256x256) | Multitexturing

Lightmap 16 x 16

62

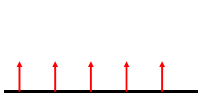
**Artefact : effet d'escalier**

L'agrandissement exagéré des textures de lightmap rend très apparents les texels, surtout sur les bords → effets d'escalier.

63

## 4.5 Bump mapping

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.

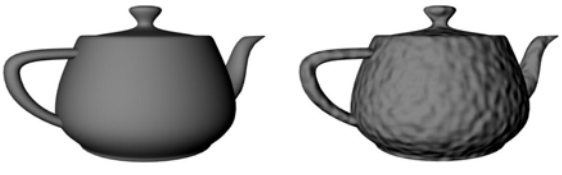


*Surface simple*

*Véritable géométrie à simuler*

*Surface simple avec bump-mapping*

64



*Sans bump-mapping* | *Avec bump-mapping*

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

65

## 4.6 Environment mapping

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

66

**Principe**

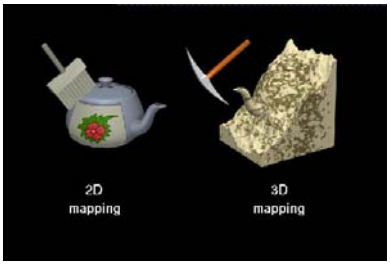
les coordonnées de texture de l'objet réfléchissant sont calculées dans une texture représentant l'environnement selon la position de l'observateur.

Plusieurs méthodes intégrées dans les cartes graphiques :

- Sphere mapping
- Cube mapping
- …

67

## 4.7 Textures 3D




La couleur en un point 3D de l'objet est donnée par une matrice 3D de texels (→ l'objet est « sculpté » dans cette matière).

68

Permet d'éviter les problème de plaquage et de raccord des textures 2D.

Mais les textures 3D sont très coûteuses en mémoire.

69

### → Textures 3D procédurales

La couleur en un point 3D de l'objet est donnée par une **fonction** $f(x,y,z)$

→Textures de bois, de marbre, etc.

70

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

Elles peuvent être utilisées avec OpenGL par le biais des **Vertex Shaders** et des **Pixel Shaders**, qui offrent des possibilités de programmation avancées des cartes graphiques.

71

Ces fonctions nécessitent très peu de mémoire (à peine quelques octets)

Mais 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.

72