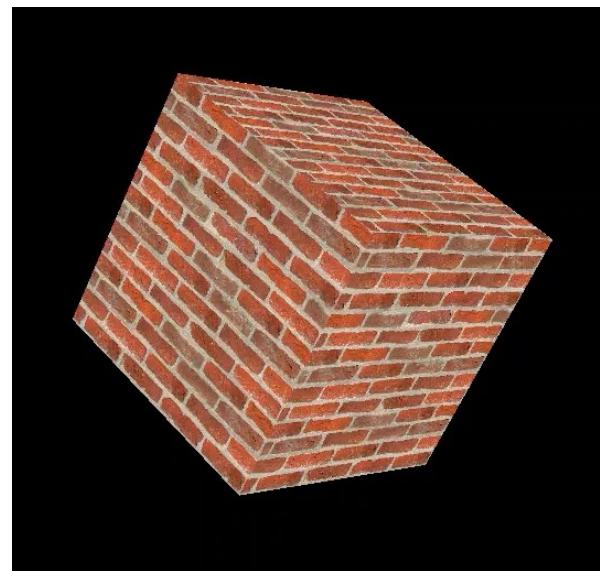
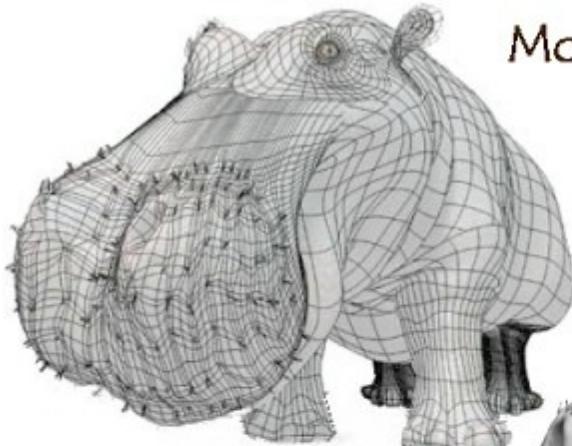


Les Textures

inspiré des cours de Sandrine LANQUETIN et
Frank Singhoff



En quête de réalisme...



Model



Model + Shading

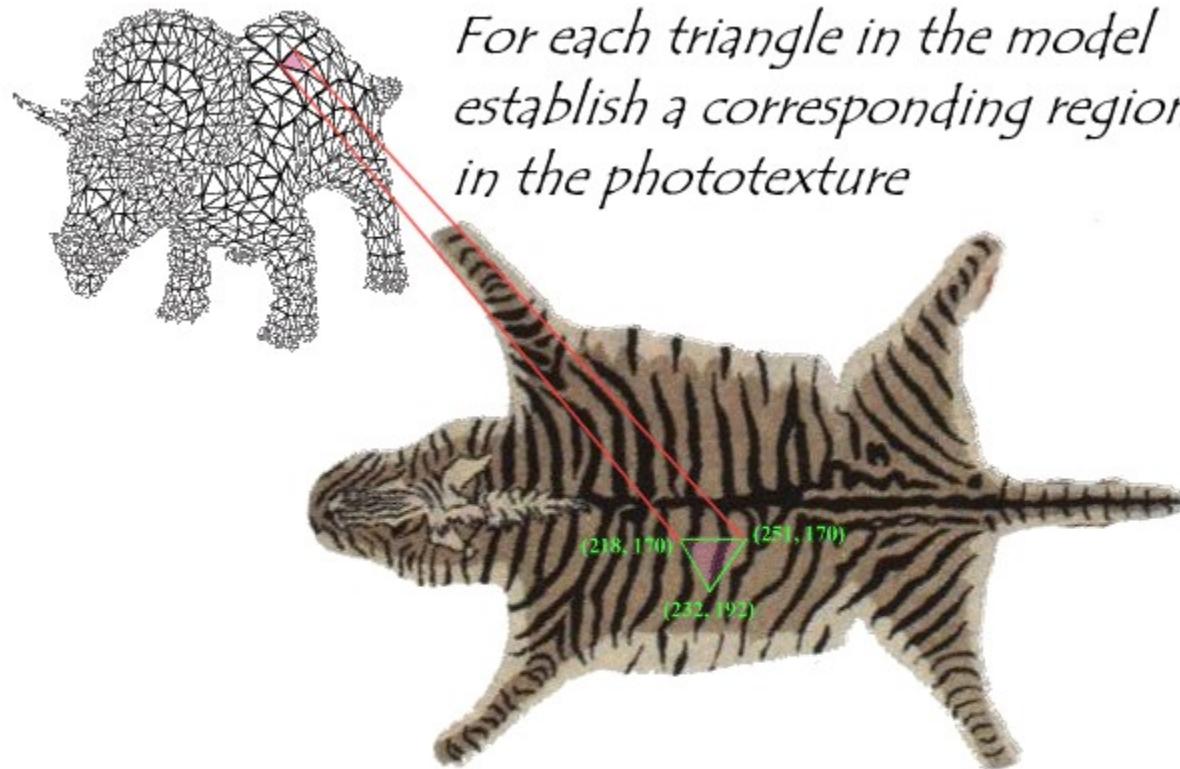


Model + Shading
+ Textures



A quel moment
les choses
deviennent-elles
réalistes ?

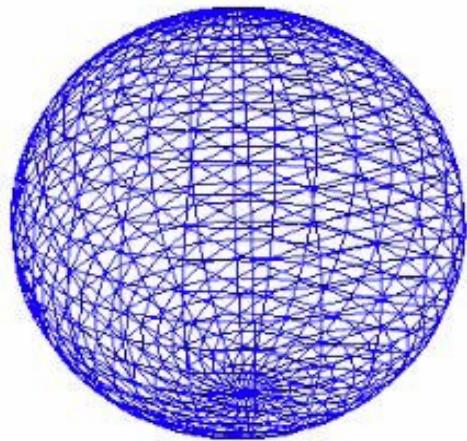
Plaquage de textures



During rasterization interpolate the coordinate indices into the texture map

Plaquage de textures

Triangle Mesh



textured with

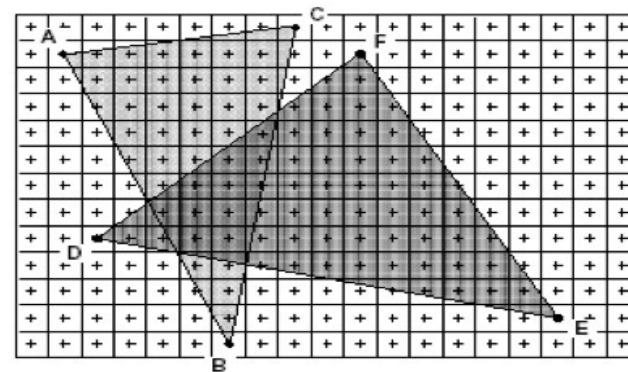


Base Texture



Rasterisation

- Découpage de la primitive 2D en pixel par interpolation des valeurs connues aux sommets: couleurs, profondeurs



Transformation du modèle

Illumination

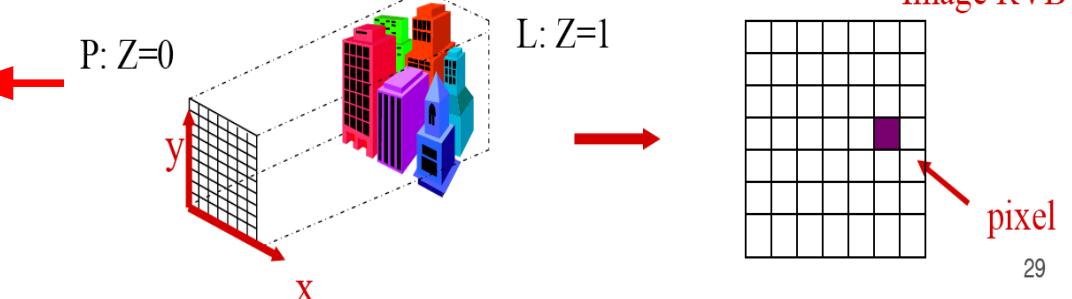
Transformation de vue

Découpage (Clipping)

Projection (dans l'espace écran)

Rastérisation

Visibilité/Affichage



29

Rasterisation

Transformation du modèle

Passage du continu au discret:

Illumination

Représentations d'objets géométriques en terme de pixels

Transformation de vue

Découpe la primitive 2D en pixels

Découpage (Clipping)

Discréteriser chaque facette en pixels

Projection (dans l'espace écran)

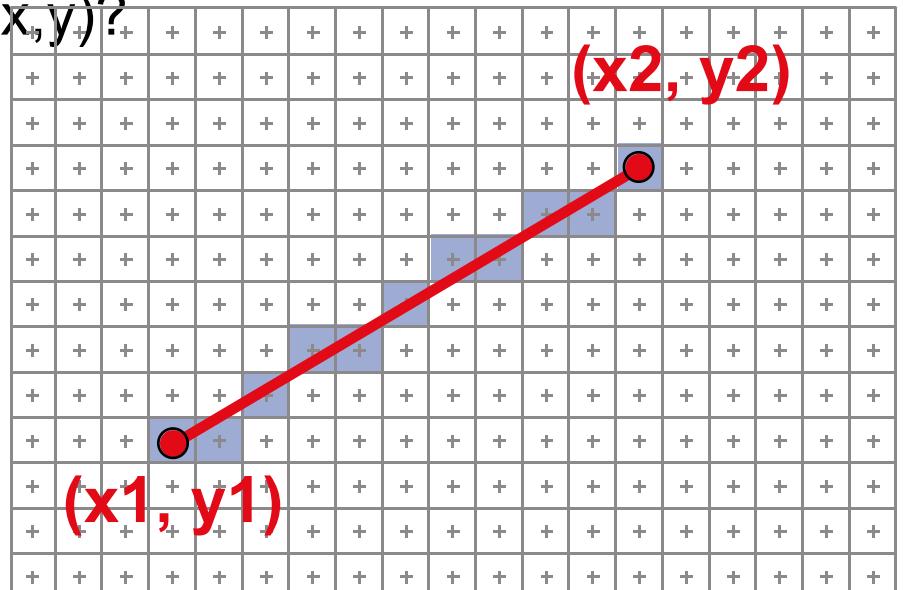
(x_1, y_1)

(x_2, y_2)

Rastérisation



Visibilité/Affichage



Rasterisation

Méthode naïve (peu précis!):

Transformation du modèle

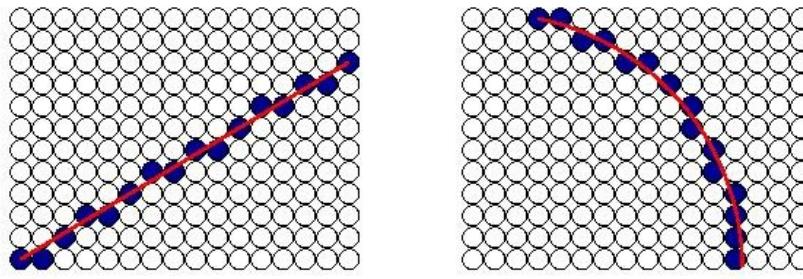
- Discrétisation de la primitive en n points,

Illumination

- Approximation au pixel le plus proche

• Méthode incrémentale :

- point suivant choisi tq minimise l'erreur d'approximation.



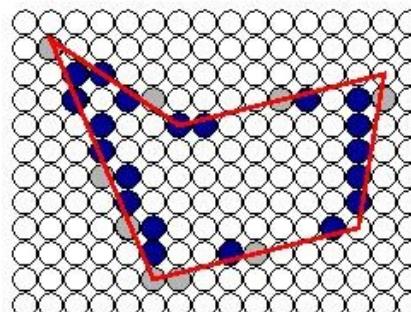
Projection (dans l'espace écran)

Droite

Cercle

Rastérisation

Visibilité/Affichage



Rasterisation

Transformation du modèle

Illumination

Transformation de vue

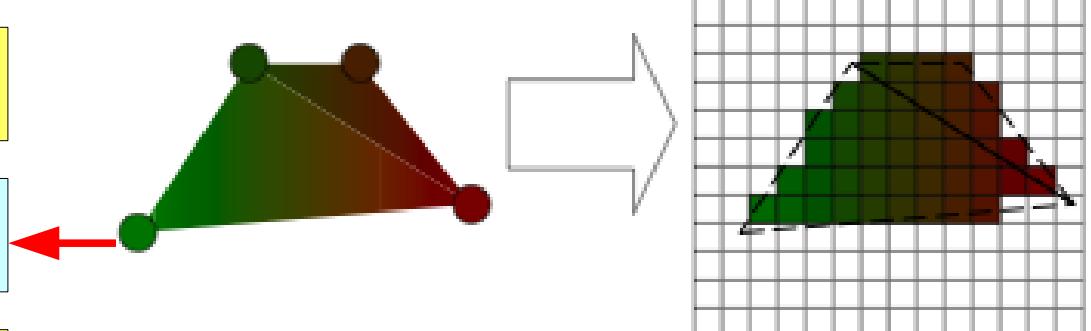
Découpage (Clipping)

Projection (dans l'espace écran)

Rastérisation

Visibilité/Affichage

- Les triangles de l'espace écran sont échantillonnés en fragments.
- Les couleurs sont interpolées



Rasterisation

Transformation du modèle

Illumination

Transformation de vue

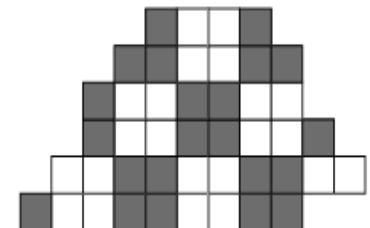
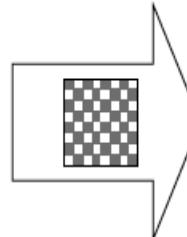
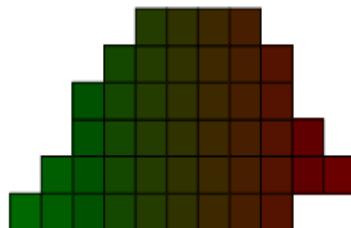
Découpage (Clipping)

Projection (dans l'espace écran)

Rastérisation

Visibilité/Affichage

- Transformation et projection des textures
- Calcul des adresses des textures
- Filtrage des textures

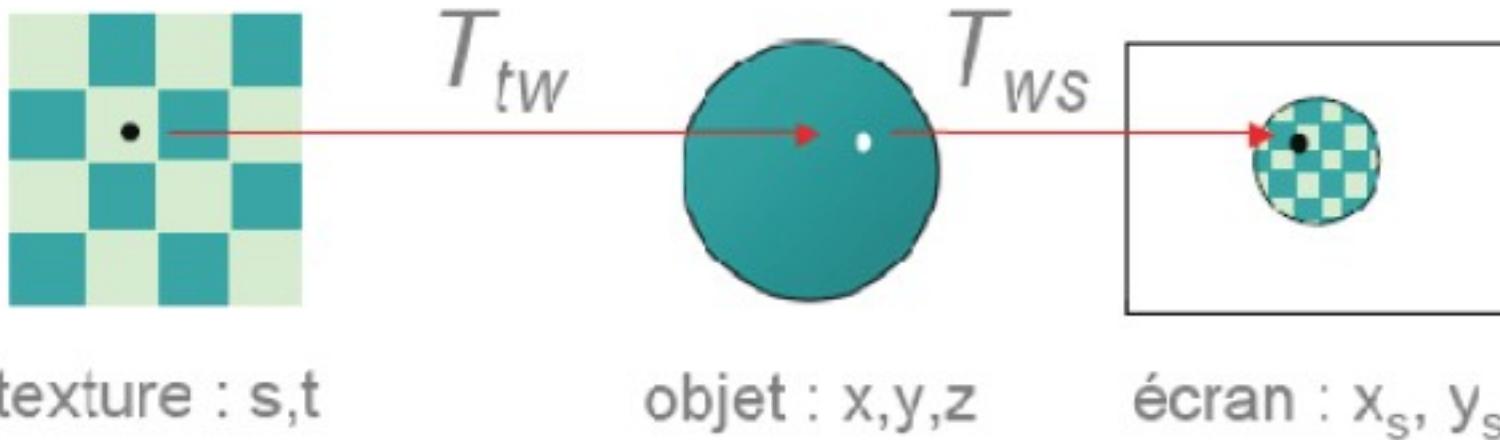


Fragments

Texture Fragments

Principe de base du placage de texture 2D

- Un texel est un couple (s,t) de coordonnées dans $[0,1]^*[0,1]$ identifiant un point de la texture
- Problème : comment trouver le(s) texel(s) correspondant à un fragment xs, ys ?
- Solution : couples sommet/texel et interpolation



Qu'est ce qu'une texture?

- Texture : tableau rectangulaire de données contenant, par exemple des données chromatiques (valeurs appelées des texels)
 - Ex. Couleur, luminance, alpha, normal...
- Texel : 1 élément du tableau
- Textures 1D, 2D voire 3D.
- On suppose souvent des textures 2D.
- Texture bitmap (ex : données scannées) ou procédurale.

Méthode générale pour appliquer une texture sur un polygone

- 1 Activer les textures.
- 2 Définir un ou des objets de texture :
 - définition des texels
 - comportement lors de l'application de la texture sur le polygone
- 3 Lors de la description de scène :
 - Spécification de la texture courante + description géométrique de la scène.
 - Spécification des coordonnées de textures : indique les parties de la texture à "plaquer" sur le polygone.
 - Spécification de la combinaison chromatique:
 - couleur polygone
 - texture
 - éclairage

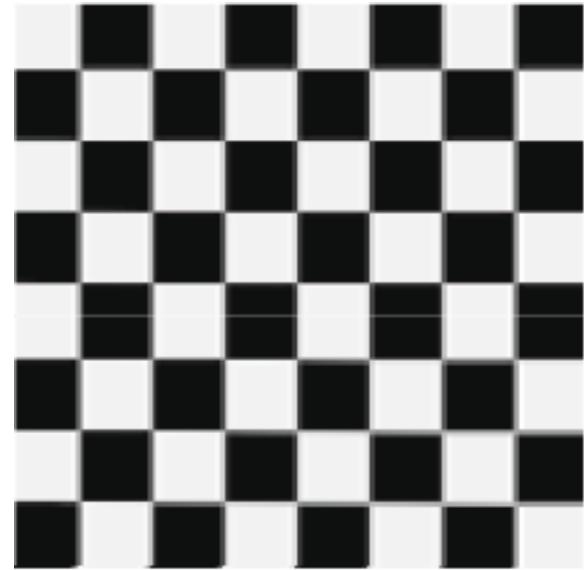
Définir un ou des objets de texture

- Préparer l'image comme un gros morceau de mémoire
 - La générer ou la charger (librairies)
- Créer une texture à partir d'une image
 - Quelles informations conserver en mémoire ?
 - RGB? RGBA?
 - Hauteur/Largeur
 - Format de données?
 - float? unsigned int?

Exemple de création de texture

- static GLubyte checkImage0[256][256][4];
- // Générer une image damier avec du code

```
void makeCheckImage(void){  
GLubyte c;  
for(int i=0; i<256; i++)  
    for(int j=0; j<256; j++) {  
        c = (((((i&0x20)==0)^((j&0x20))==0))*255;  
        checkImage0[i][j][0] = checkImage0[i][j][1] =  
            checkImage0[i][j][2] = c;  
        checkImage0[i][j][3] = 255;  
    }  
}
```



Mise en place

- Activer les fonctionnalités de texture

- Activé ou Désactivé

`glEnable(GL_TEXTURE_2D)`

- Spécifier comment appliquer la texture sur les pixels
 - Remplace/Mélange/Module/Décale

`glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);`

- Autres paramètres
 - filtres:

`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);`

`glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);`

- Spécification de la texture

```
void glTexImage2D(GLenum target,GLint level,GLint  
component,GLsizei l,GLsizei h,GLint b,GLenum  
format,GLenum type,const GLvoid *pix);
```

- Définit une texture 2D.
- target: `GL_TEXTURE_2D` ou `GL_PROXY_TEXTURE_2D` (test de disponibilité mémoire)
- level: 0 pour une texture à 1 niveau
- component: nombre de composantes à gérer
 - 1 -> rouge
 - 2 -> R et A
 - 3 -> R, V et B
 - 4 -> R, V, B et A

Commandes

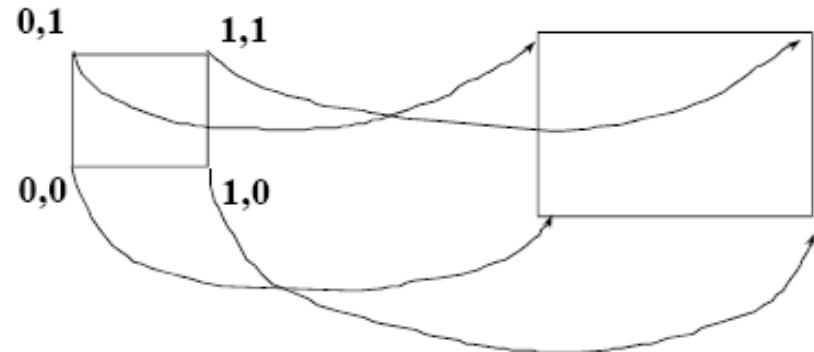
- l et h: dimension de l'image représentant la texture.
 - l et h sont souvent des puissances de 2.
- b: largeur du bord (usuellement 0).
 - Si b <> 0 alors l et h sont incrémentés de 2xb.
- format et type: format et type de données de l'image
- pix: image à placer stockée dans un tableau

exemple:

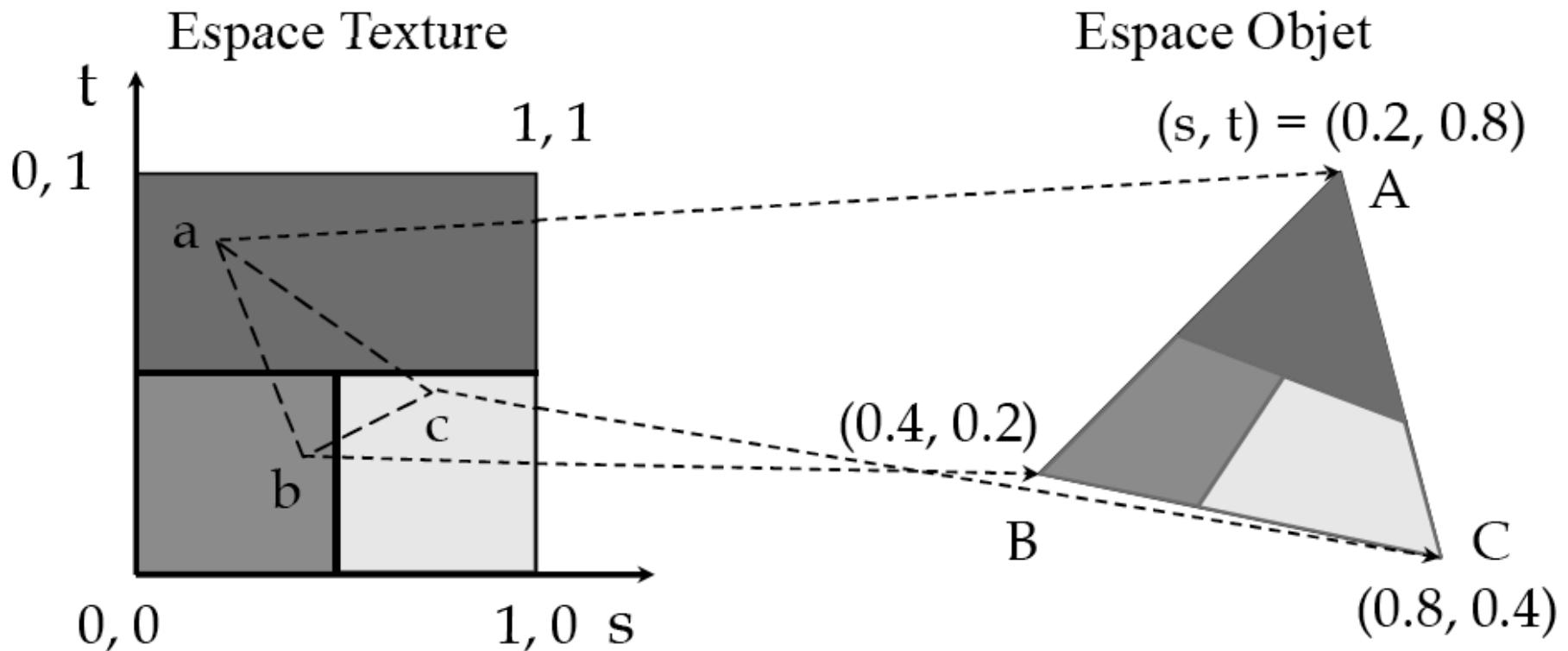
```
glTexImage2D(GL_TEXTURE_2D, 0, type,  
texture[0].width, texture[0].height, 0, GL_RGB,  
GL_UNSIGNED_BYTE, texture[0].imageData);
```

Le Rendu

- Comment indiquer où et quels sont les texels d'une texture qui doivent être plaqués sur un polygone ?
- Coordonnées de texture:
 - Associe à chaque sommet des coordonnées de texture
 - Laisse OpenGL générer des coordonnées de texture
- Les texels peuvent être référencés selon leurs coordonnées grâce à la primitive `glTexCoord`.
- Exemple des textures à 2 dimensions (S,T) :
 - Axe S = abscisse. Axe T = ordonnée.
 - (0,0) désigne le coin bas/gauche.
 - (0,1) désigne le coin haut/gauche.
 - (1,1) désigne le coin haut/droit.
 - (1,0) désigne le coin bas/droit.



Spécifier à chaque sommet le lien avec un texel



• Coordonnées de la texture

- Pour chaque sommet définissant un objet, les coordonnées lui correspondant dans la texture doivent être précisées.
- Les coordonnées dans la texture des pixels à l'intérieur des facettes sont obtenues par interpolation entre ces valeurs.
- 1, 2, 3 ou 4 coordonnées (s, t, r, q) suivant le type de texture utilisée:
 - s affecté pour les textures 1D (t et r à 0, q à 1)
 - s et t affectés pour les textures 2D (r à 0, q à 1)
 - s , t et r affectés pour les textures 3D (q à 1)
 - q est l'équivalent de w en coordonnées homogènes.

Code C++

```
glBindBuffer( GL_ARRAY_BUFFER, vboids[ 1 ] );  
  
glBufferData( GL_ARRAY_BUFFER, texcoords.size() *  
sizeof( float ),texcoords.data(), GL_STATIC_DRAW );  
  
auto coord = glGetAttribLocation( progid, "in_coord" );  
  
glVertexAttribPointer( coord, 2, GL_FLOAT, GL_FALSE, 0,  
0 );  
  
 glEnableVertexAttribArray( coord );
```

Shader

```
in vec2 in_coord;  
  
out vec2 texcoord;  
  
void main(void)  
{gl_Position = mvp *  
vec4( in_pos, 1.0 );  
  
texcoord = in_coord;
```

```
in vec2 texcoord;  
  
out vec4 frag_color;  
  
uniform sampler2D tex;  
  
void main(void)  
{ frag_color =  
texture( tex, texcoord );
```

Exemple :

```
std::vector< float > verticesdur = {-0.5,-0.5,0.5,  
                                     0.5,-0.5,0.5,  
                                     -0.5,0.5,0.5,  
                                     0.5,0.5,0.5,  
                                     0.5,-0.5,-0.5,...}  
  
std::vector< float > texcoords = {  
                                     0.3f, 0.48f,  
                                     0.3f, 0.25f,  
                                     0.55f, 0.48f,  
                                     0.55f, 0.25f,...}
```

```
glBegin(GL_QUADS);

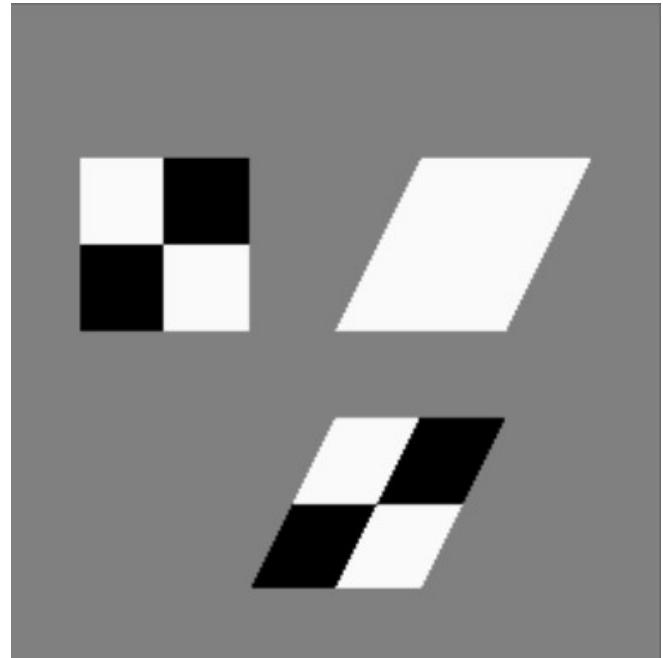
glTexCoord2f(0.0, 0.0); glVertex3f(0,0,0);

glTexCoord2f(0.0, 1.0); glVertex3f(1,2,0);

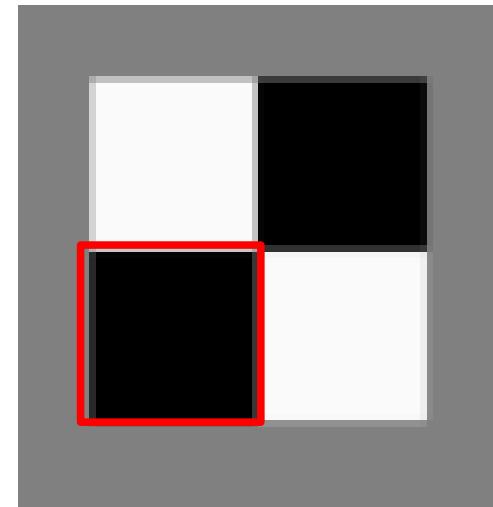
glTexCoord2f(1.0, 1.0); glVertex3f(3,2,0);

glTexCoord2f(1.0, 0.0); glVertex3f(2,0,0);

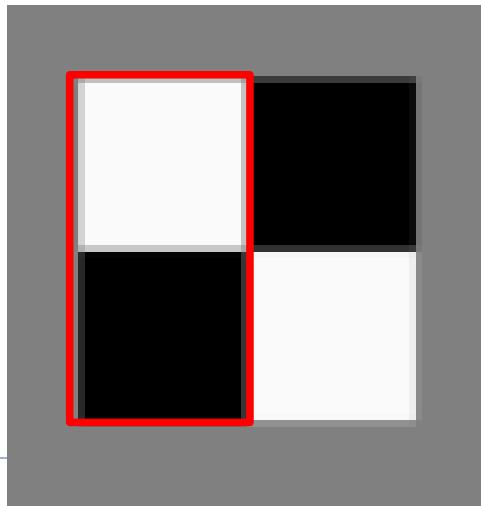
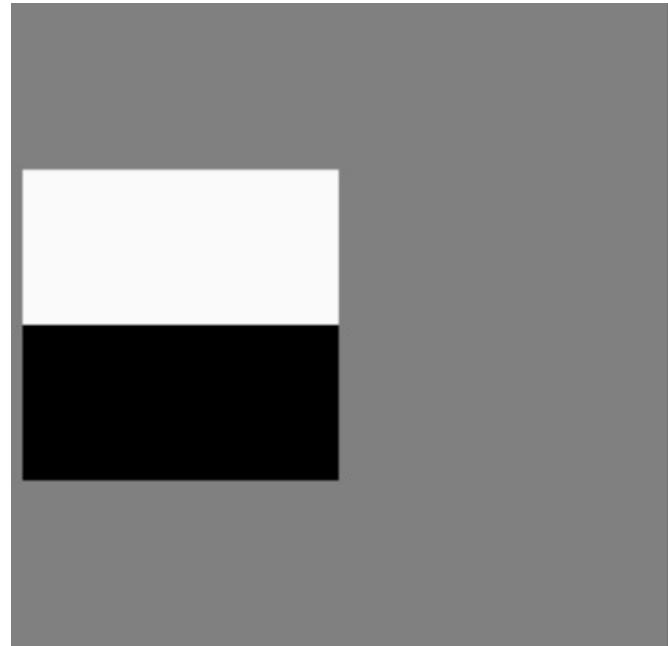
glEnd();
```



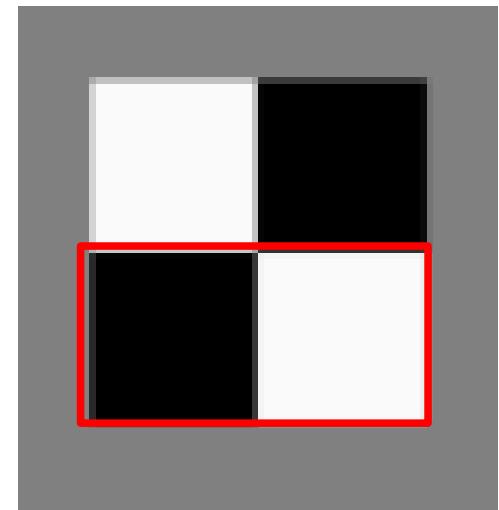
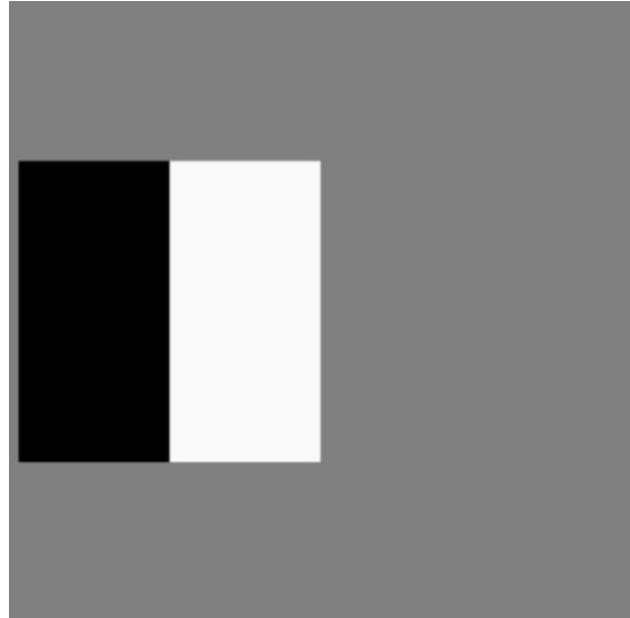
```
glBegin(GL_QUADS);  
  
glTexCoord2f(0.0, 0.0); glVertex3f(0,0,0);  
  
glTexCoord2f(0.0, 0.5); glVertex3f(0,1,0);  
  
glTexCoord2f(0.5, 0.5); glVertex3f(1,1,0);  
  
glTexCoord2f(0.5, 0.0); glVertex3f(1,0,0);  
  
glEnd();
```



```
glBegin(GL_QUADS);  
glTexCoord2f(0.0, 0.0);  
glVertex3f(0,0,0);  
glTexCoord2f(0.0, 1.0);  
glVertex3f(0,1,0);  
glTexCoord2f(0.5, 1.0);  
glVertex3f(1,1,0);  
glTexCoord2f(0.5, 0.0);  
glVertex3f(1,0,0);  
glEnd();
```

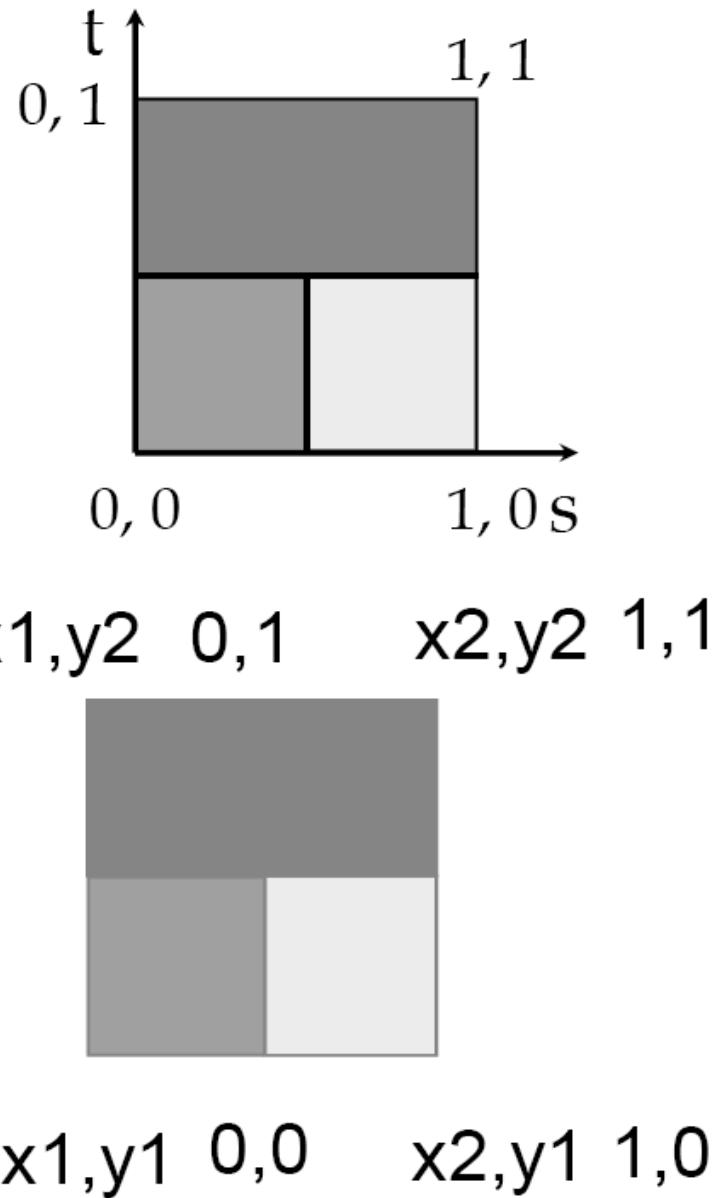


```
glBegin(GL_QUADS);  
glTexCoord2f(0.0, 0.0);  
 glVertex3f(0,0,0);  
glTexCoord2f(0.0, 0.5);  
 glVertex3f(0,1,0);  
glTexCoord2f(1.0, 0.5);  
 glVertex3f(1,1,0);  
glTexCoord2f(1.0, 0.0);  
 glVertex3f(1,0,0);  
glEnd();
```



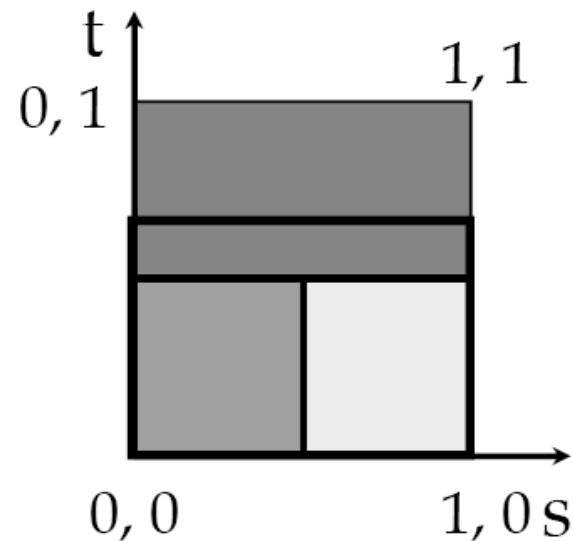
Carré texturé

```
glEnable(GL_TEXTURE_2D) ;  
  
glBegin(GL_QUADS) ;  
  
glTexCoord2f(0,0) ;  
glVertex2f(x1,y1) ;  
glTexCoord2f(1 0) ;  
glVertex2f(x2,y1) ;  
  
glTexCoord2f(1,1) ;  
glVertex2f(x2,y2) ;  
glTexCoord2f(0,1) ;  
glVertex2f(x1,y2) ;  
  
glEnd() ;
```



Rectangle texturé

```
glEnable(GL_TEXTURE_2D) ;  
  
glBegin(GL_QUADS) ;  
  
glTexCoord2f(0,0) ;  
glVertex2f(x1,y1) ;  
glTexCoord2f(1 0) ;  
glVertex2f(x2,y1) ;  
  
glTexCoord2f(1,2/3) ;  
glVertex2f(x2,y2) ;  
glTexCoord2f(0,2/3) ;  
glVertex2f(x1,y2) ;  
  
glEnd() ;
```



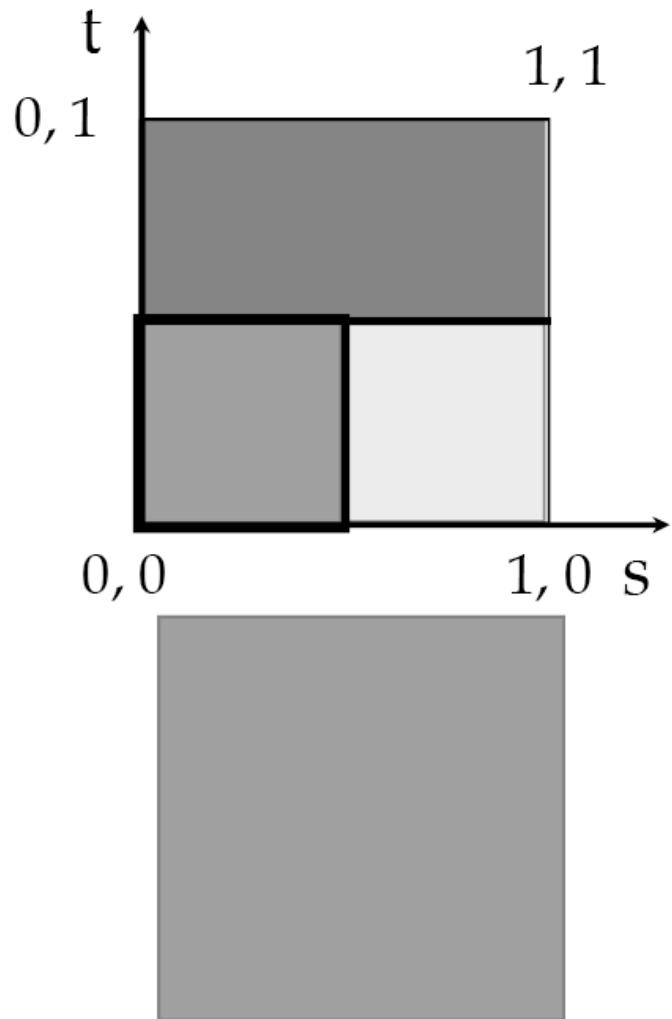
x1,y2 0,2/3 x2,y2 1,2/3



x1,y1 0,0 x2,y1 1,0

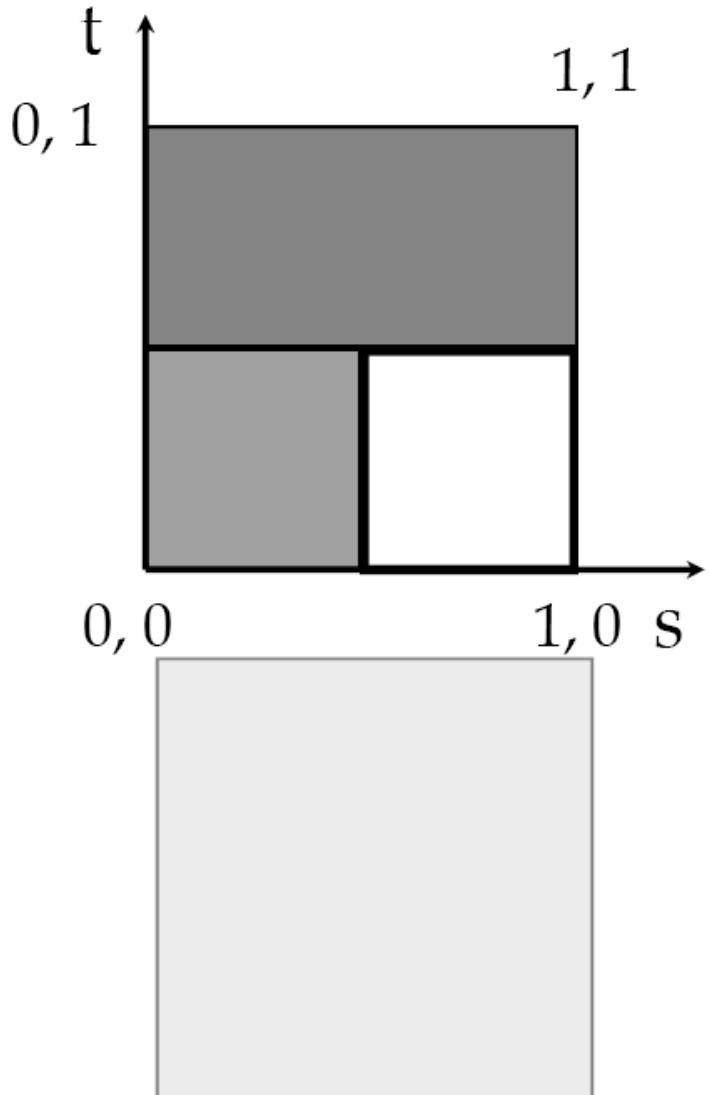
Rectangle texturé

```
glBegin(GL_POLYGON);
glTexCoord2f(0, 0);
glVertex3f(0, 0, 0);
glTexCoord2f(0.5, 0);
glVertex3f(1, 0,0);
glTexCoord2f(0.5, 0.5);
glVertex3f(1, 1, 0);
glTexCoord2f(0, 0.5);
glVertex3f(0, 1,0);
glEnd();
```



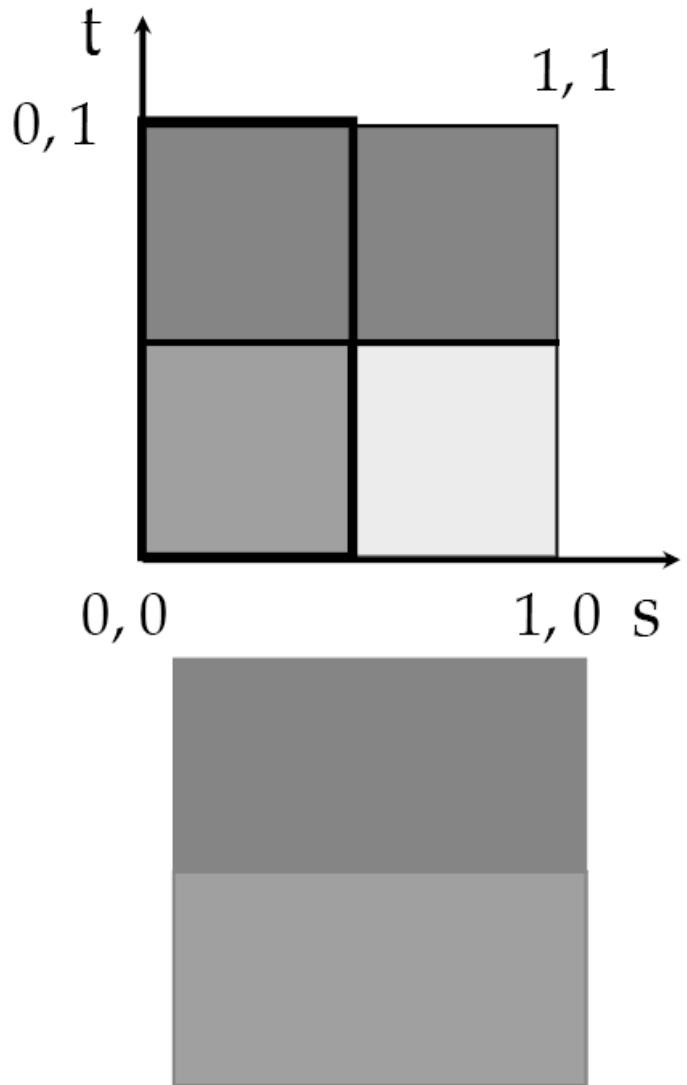
Rectangle texturé

```
glBegin(GL_POLYGON);  
glTexCoord2f(0.5, 0);  
 glVertex3f(0, 0, 0);  
glTexCoord2f(1, 0);  
 glVertex3f(1, 0, 0);  
glTexCoord2f(1, 0.5);  
 glVertex3f(1, 1, 0);  
glTexCoord2f(0.5, 0.5);  
 glVertex3f(0, 1, 0);  
glEnd();
```



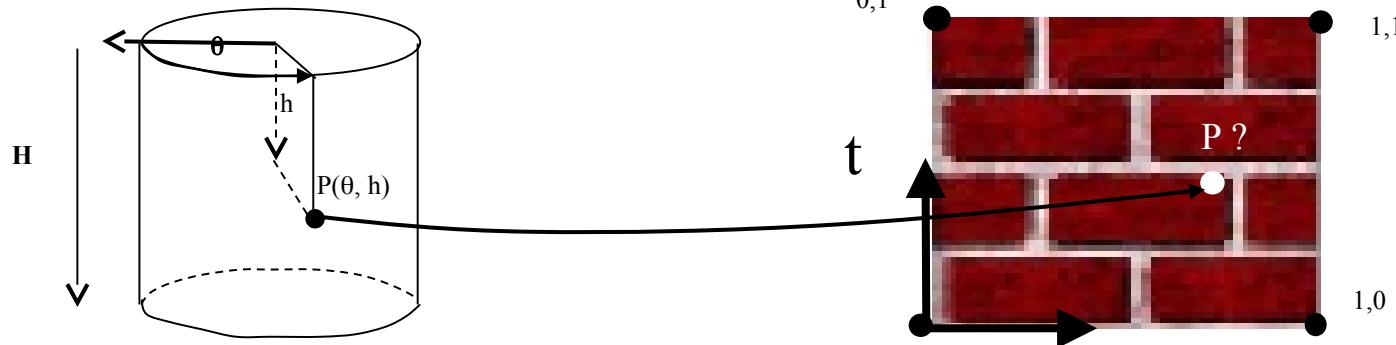
Rectangle texturé

```
glBegin(GL_POLYGON);
glTexCoord2f(0, 0);
glVertex3f(0, 0, 0);
glTexCoord2f(0.5, 0);
glVertex3f(1, 0, 0);
glTexCoord2f(0.5, 1);
glVertex3f(1, 1, 0);
glTexCoord2f(0, 1);
glVertex3f(0, 1, 0);
glEnd();
```

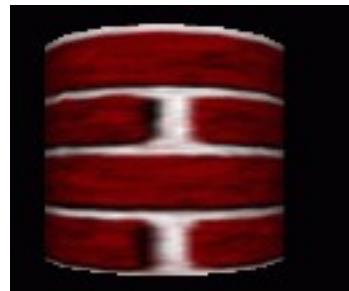


Plaquage de textures

- Plaquer une texture sur un objet consiste à trouver pour chaque point P de l'objet à texturer son correspondant dans l'image.



$$s = \theta / 2\pi$$
$$t = h / H$$



Il existe des textures 1 D et 3D

- `void glTexImage1D(GLenum target,GLint level,GLint component,GLsizei l,GLint b,GLenum format,GLenum type,const GLvoid *pix);`
- Définit une texture 1D.
- target: `GL_TEXTURE_1D`
- `void glTexImage3D(GLenum target,GLint level,GLint component,GLsizei l,GLsizei h,GLsizei z,GLint b,GLenum format,GLenum type,const GLvoid *pix);`

Oui mais?

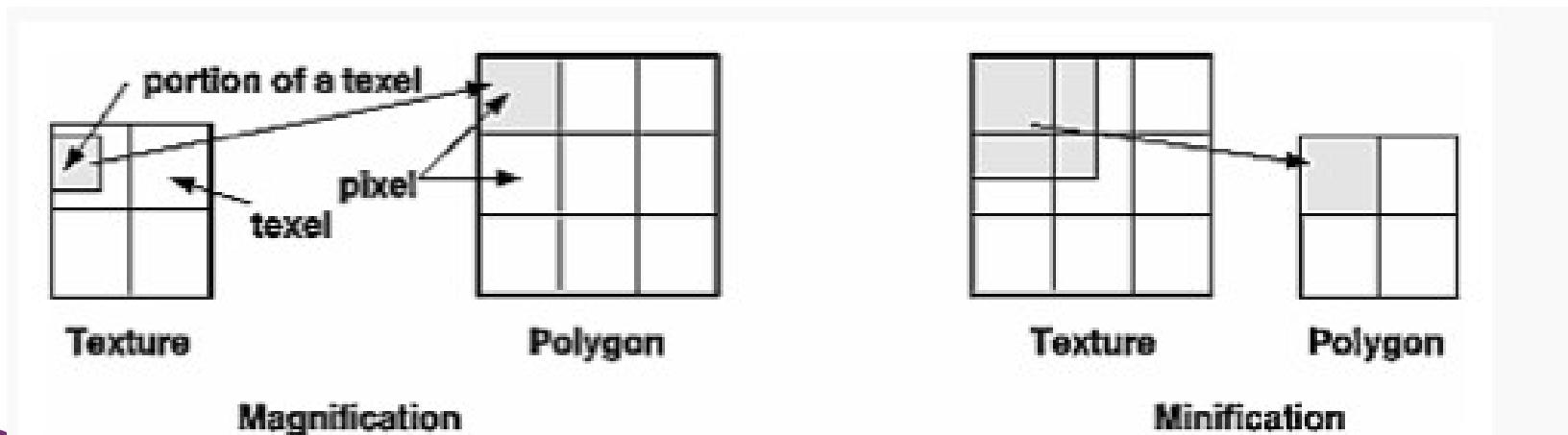
- Que faire quand le polygone n'est pas un quadrilatère ?
- Comment rétrécir une texture ?
- Comment combiner les informations chromatiques d'une texture avec la couleur du polygone, de l'éclairage ?

⇒

- Réalisée par le shader de fragment :
- Exemple:
-
- `frag_color += texture(tex, texcoord);`

Spécification des techniques de placage

- Échantillonnage
- Après transformation en coordonnées écran, les pixels de la texture (texels) correspondent rarement à des pixels de l'image
 - un pixel = un morceau de texel -> agrandissement,
 - un pixel = plusieurs texels -> réduction
 - Visualisation claire des texels s'ils sont trop grands à l'écran
 - non visualisation des texels existant entre deux pixels s'ils sont trop petits.



Spécification des techniques de placage

- OpenGL propose le choix de la ou des techniques à employer pour effectuer les calculs d'"échantillonnage" nécessaire à la détermination de la couleur d'affichage dans les cas d'agrandissement (resp. réduction) quand moins (resp. plus) d'un texel correspond à un pixel de l'image.
- Échantillonnage de point (le texel le plus proche) ou filtrage linéaire (filtre 2 x 2) pour obtenir les valeurs de la texture
 - **GL_NEAREST**: choix du texel le plus proche du centre du pixel considéré et utilisation de sa couleur
 - plus grande rapidité mais affichage de moins bonne qualité
 - **GL_LINEAR**: choix et moyenne pondérée du carré de 2x2 texels le plus proche du centre du pixel
 - rapidité moins importante mais affichage de meilleure qualité
 - Le filtrage linéaire nécessite une bordure d'un texel supplémentaire pour filtrer aux bords (border = 1)

Spécification des techniques de placage

```
void glTexParameter{if}{v}(GLenum target, GLenum pn, TYPE param);
```

- Configuration du placage de texture
 - target: **GL_TEXTURE_2D** ou **GL_TEXTURE_1D**
 - pn: paramètre sur lequel est effectuée la configuration
 - param: valeur adoptée
- pn: **GL_TEXTURE_MAG_FILTER**: méthode d'agrandissement
 - param: **GL_NEAREST**
 - param: **GL_LINEAR**
- pn: **GL_TEXTURE_MIN_FILTER**: méthode de réduction
 - param: **GL_NEAREST**
 - param: **GL_LINEAR**

Spécification des techniques de placage

```
void glTexParameter{if}{v}(GLenum target, GLenum pn, TYPE param);
```

- Quatre choix de filtres supplémentaires sont possibles avec réduction:
 - Pour un plan mipmap donné on peut choisir :
 - `GL_NEAREST_MIPMAP_NEAREST`
 - Filtrage: `GL_NEAREST_MIPMAP_LINEAR`
 - Pour choisir les 2 meilleurs plans utiliser :
 - `GL_LINEAR_MIPMAP_NEAREST`
 - `GL_LINEAR_MIPMAP_LINEAR`

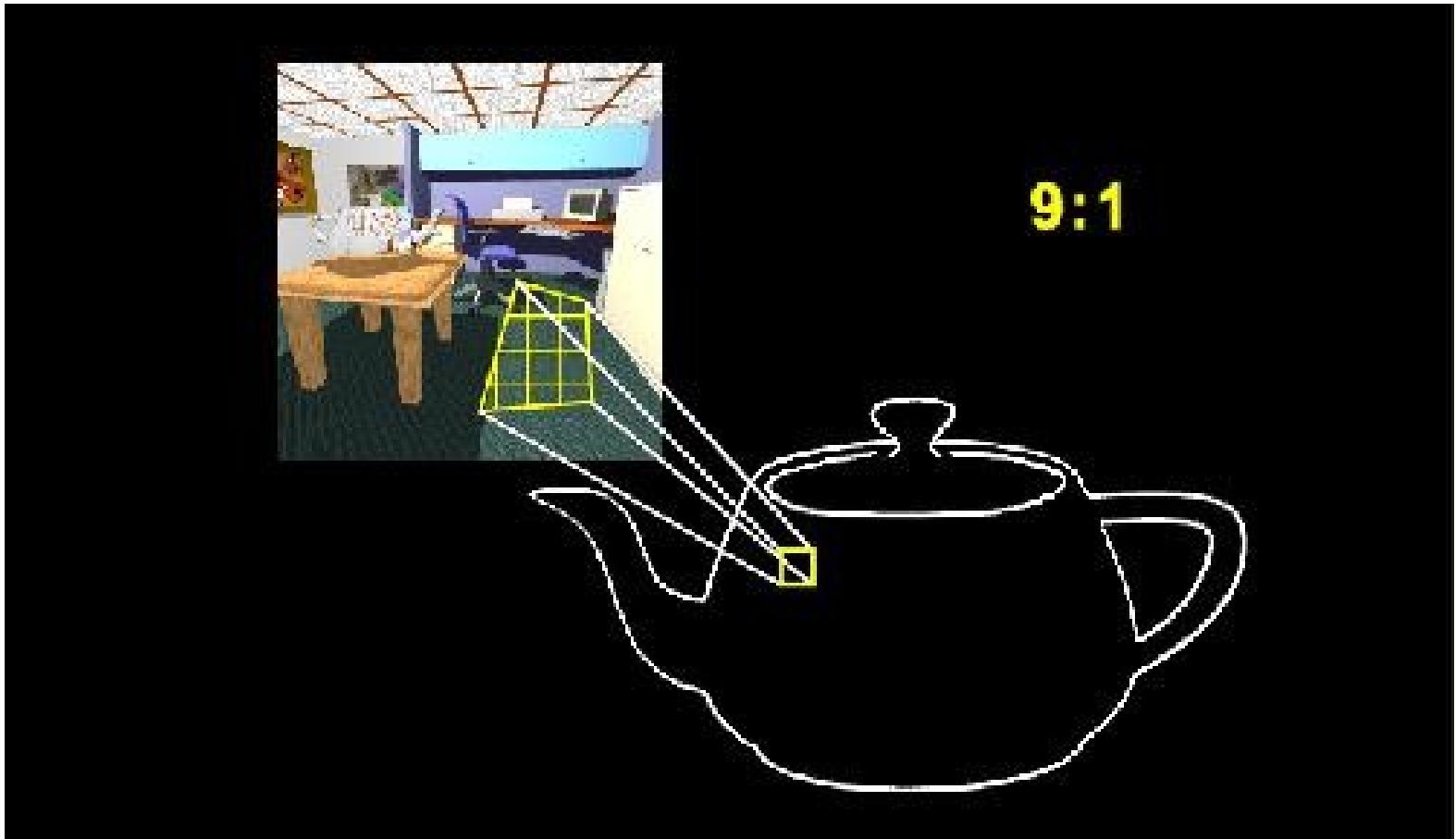
Mip-Mapping (1/3)

- Calcul de différentes résolutions de textures



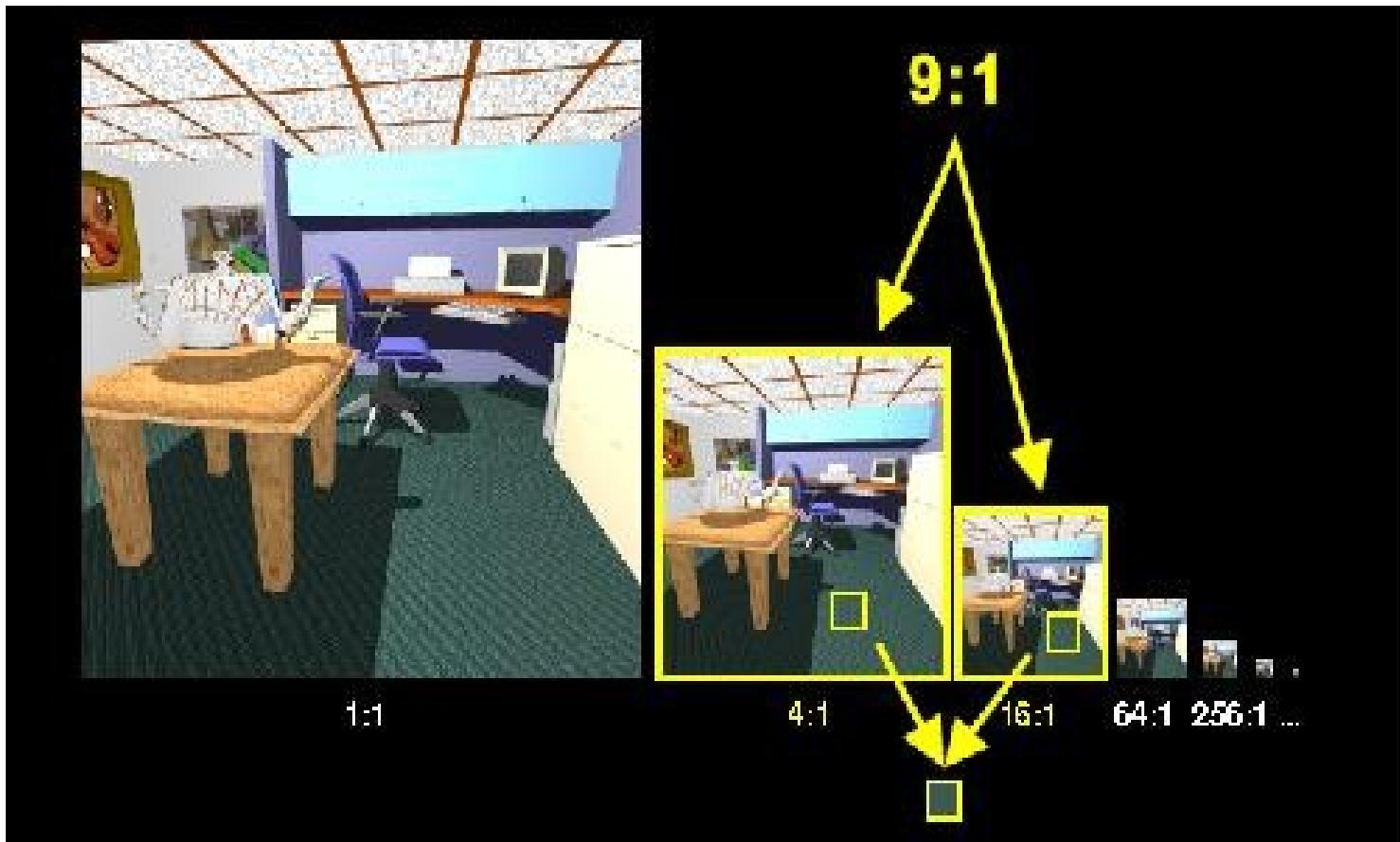
Mip-Mapping (2/3)

- Projection du pixel objet sur la texture originale
- Détermination du nombre de pixels texture recouverts



Mip-Mapping (3/3)

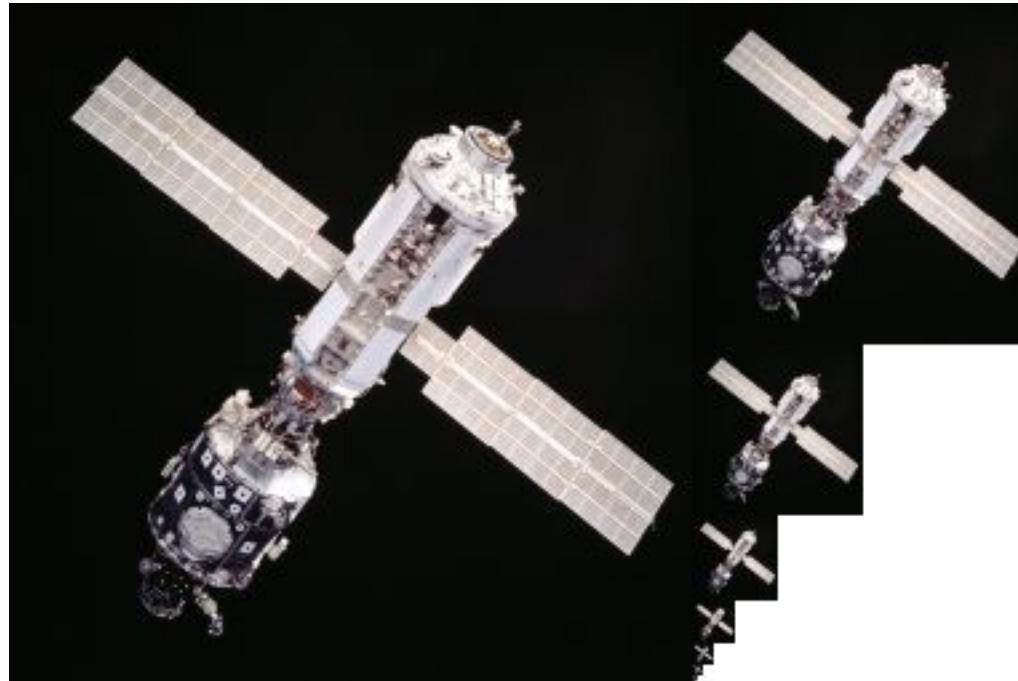
- Recherche des textures les plus proches
- Calcul de la moyenne des pixels recouverts dans ces textures



D'abord spécifier la technique d'interpolation

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

```
glTexParameteri(GL_TEXTURE_2D,  
    GL_TEXTURE_MIN_FILTER,  
    GL_NEAREST_MIPMAP_LINEAR)
```



Ensuite

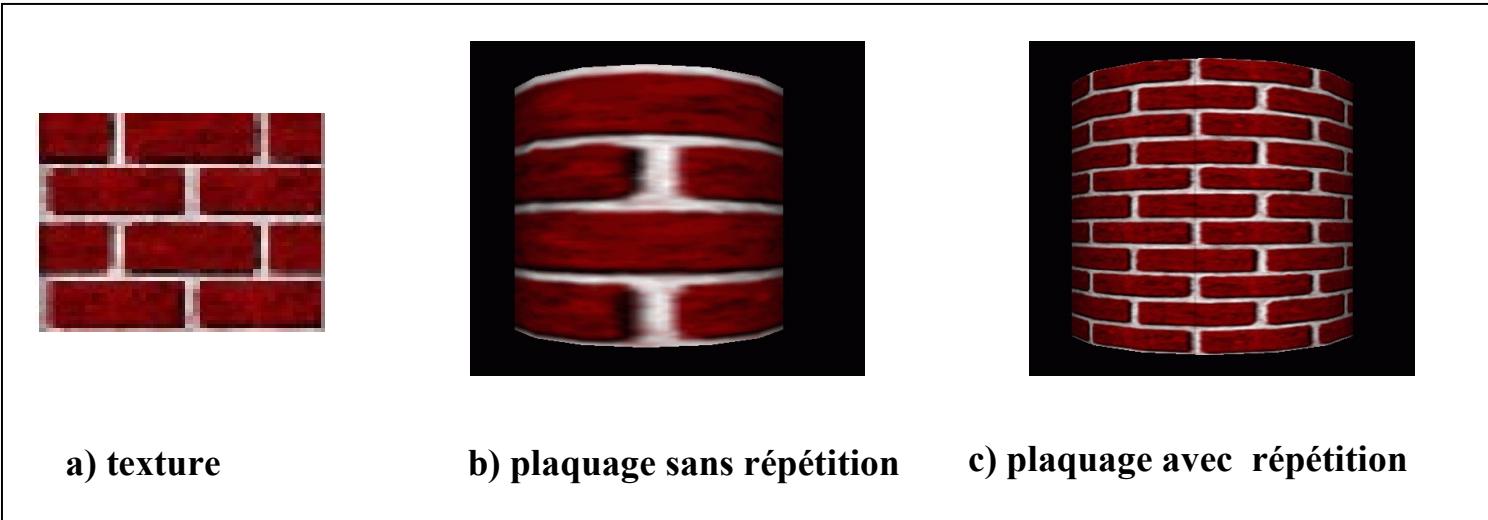
```
glTexParameteri(GL_TEXTURE_2D,  
    GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);  
  
glTexParameteri(GL_TEXTURE_2D,  
    GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width,  
    height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);  
  
glGenerateMipmap(GL_TEXTURE_2D);
```

Répétition et seuillage de texture

- Les coordonnées textures sont définies dans les images sources entre les coordonnées 0 et 1. Au delà elles peuvent être répétées ou seuillées.
- Répétition: toute coordonnée supérieure à 1 ou inférieure à 0 voit sa partie entière ignorée.
- Seuillage: toute coordonnée supérieure à 1 (resp. inférieure à 0) est ramenée à 1 (resp. à 0).

Plaquage de texture sur un cylindre

- Facteur de répétition



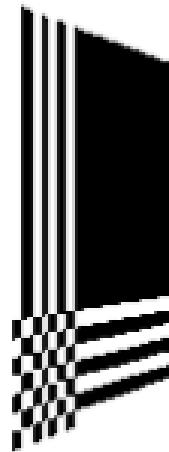
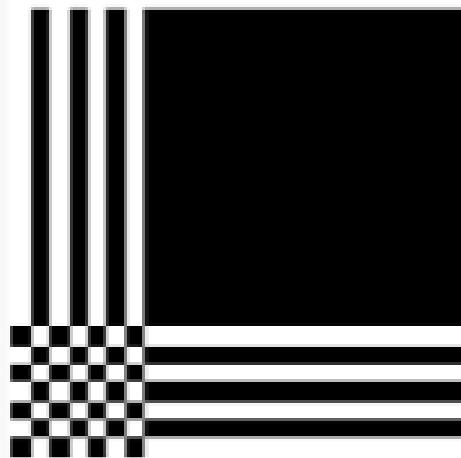
Répétition et seuillage de texture

```
void glTexParameter{if}{v}(GLenum target, GLenum pn,  
    TYPE param);
```

- Configuration du placage de texture
 - target: **GL_TEXTURE_2D** ou **GL_TEXTURE_1D**
 - pn: paramètre sur lequel est effectuée la configuration
 - param: valeur adoptée
- pn: **GL_TEXTURE_WRAP_S**
 - param: **GL_CLAMP** / **GL_REPEAT**
- pn: **GL_TEXTURE_WRAP_T**
 - param: **GL_CLAMP** / **GL_REPEAT**
- pn: **GL_TEXTURE_BORDER_COLOR**
 - param: quatre valeurs entre 0 et 1

GL_CLAMP_TO_EDGE:
seuillage
GL_REPEAT:
répétition

Répétition et seuillage de texture



GL_CLAMP_TO_EDGE:
seuillage
GL_REPEAT:
répétition

Figure 9-8 : Clamping a Texture

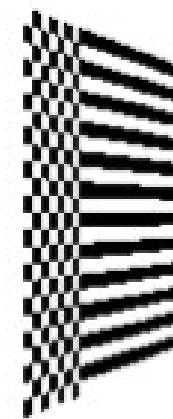
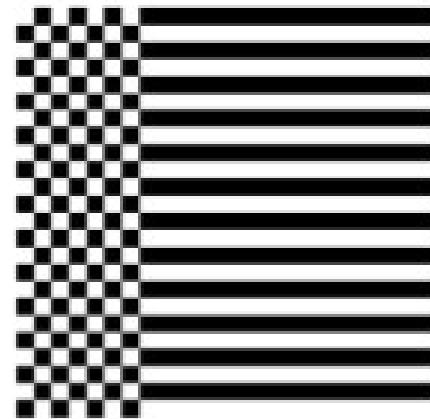
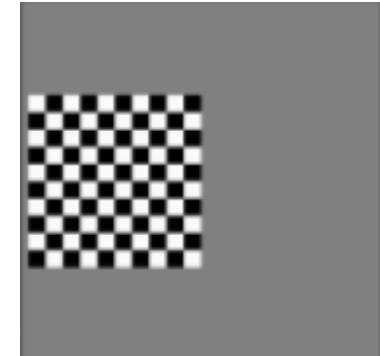
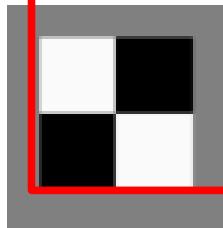


Figure 9-9 : Repeating and Clamping a Texture

```
glTexParameteri(GL_TEXTURE_2D,  
GL_TEXTURE_WRAP_S, GL_REPEAT);  
glTexParameteri(GL_TEXTURE_2D,  
GL_TEXTURE_WRAP_T, GL_REPEAT);  
glBegin(GL_QUADS);  
  
glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);  
glTexCoord2f(0.0, 5.0); glVertex3f(0, 1, 0);  
glTexCoord2f(5.0, 5.0); glVertex3f(1, 1, 0);  
glTexCoord2f(5.0, 0.0); glVertex3f(1, 0, 0);  
  
glEnd();
```



```
glTexParameteri(GL_TEXTURE_2D,  
GL_TEXTURE_WRAP_S, GL_REPEAT);  
glTexParameteri(GL_TEXTURE_2D,  
GL_TEXTURE_WRAP_T, GL_REPEAT);  
glBegin(GL_QUADS);  
glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);  
glTexCoord2f(0.0, 3.2); glVertex3f(0, 1, 0);  
glTexCoord2f(2.0, 3.2); glVertex3f(1, 1, 0);  
glTexCoord2f(2.0, 0.0); glVertex3f(1, 0, 0);  
glEnd();
```



```

glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);

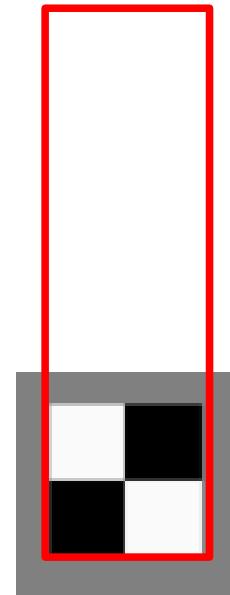
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_T, GL_REPEAT);

glBegin(GL_QUADS);

glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
glTexCoord2f(0.0, 3.0); glVertex3f(0, 1, 0);
glTexCoord2f(3.0, 3.0); glVertex3f(1, 1, 0);
glTexCoord2f(3.0, 0.0); glVertex3f(1, 0, 0);

glEnd();

```



```

glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

glBegin(GL_QUADS);

glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
glTexCoord2f(0.0, 3.0); glVertex3f(0, 1, 0);
glTexCoord2f(3.0, 3.0); glVertex3f(1, 1, 0);
glTexCoord2f(3.0, 0.0); glVertex3f(1, 0, 0);

glEnd();

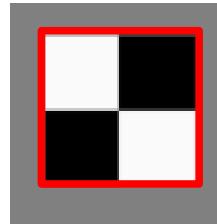
```



```

glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
glTexCoord2f(0.0, 3.0); glVertex3f(0, 1, 0);
glTexCoord2f(3.0, 3.0); glVertex3f(1, 1, 0);
glTexCoord2f(3.0, 0.0); glVertex3f(1, 0, 0);
glEnd();

```



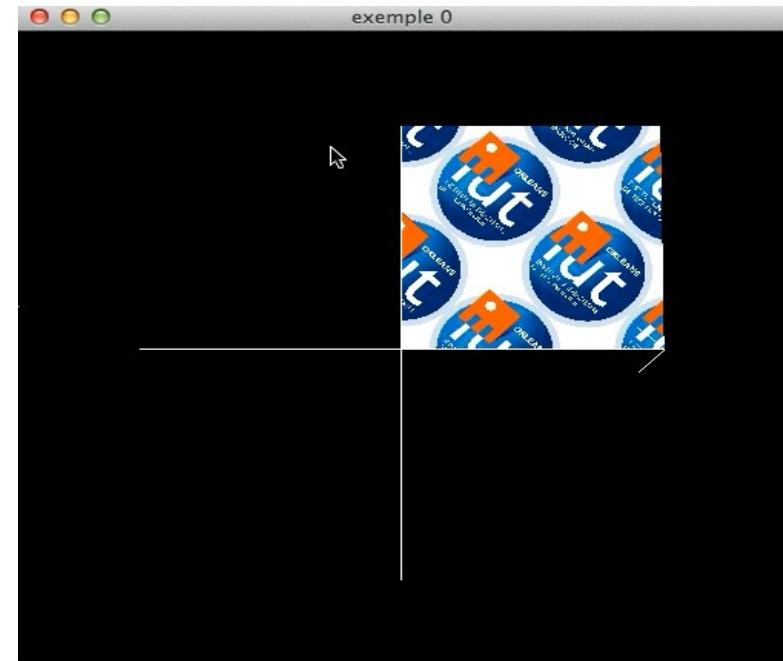
Autorisation du placage

- glEnable(int mode);
- mode: GL_TEXTURE_2D ou GL_TEXTURE_1D

Matrice de Texture

- Sur les textures, il est possible d'effectuer les mêmes transformations (rotation, translation, redimensionnement, ...) que pour les sommets.
- On utilise les transformations matricielles
- Exemple:

```
texcoord=(mv *  
vec4(in_coord,0.0,1.0)).st;
```



Objets de texture

- Passer plus rapidement d'une texture à l'autre
 - `TexImage(...)` 1 fois /texture
- Etapes pour utiliser des objets de texture
 - Générer des noms de texture
 - Lier les objets de texture aux données de texture
 - Fixer les propriétés pour chaque objet de texture
 - Pour le rendu, rappeler les objets de texture

Objets de texture

- Générer des noms de texture
 - `glGenTextures(n, *texIds);`
 - Un nom de texture est un entier
 - Le nom de texture est ensuite attribué à des données de texture
 - n = nombre de textures souhaitées
 - texIds = tableau d'entiers identifiant les textures
- Exemple

```
unsigned int texIds[5];  
glGenTextures(5, texIds)
```

Objets de texture

- Lier les textures avec:

```
glBindTexture( target, id );
```

- Attribuer une id de texture à un type de texture
- 3 sortes de textures
 - GL_TEXTURE_1D
 - GL_TEXTURE_2D
 - GL_TEXTURE_3D

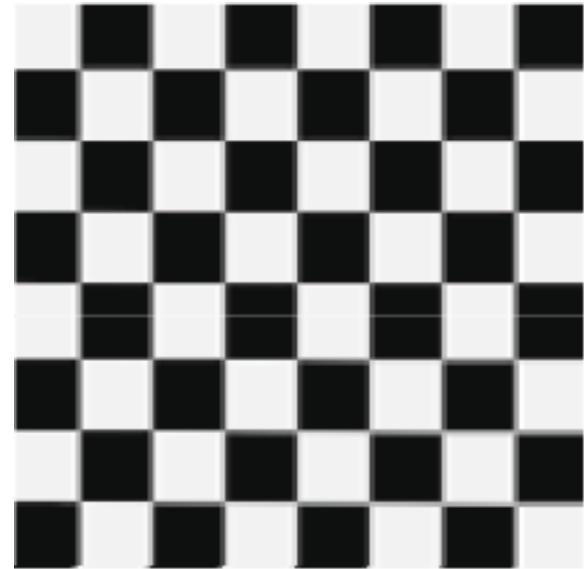
- Exemple

```
glBindTexture(GL_TEXTURE_2D, texIds[0]);
```

Exemple de création de texture

- static GLubyte checkImage0[256][256][4];
- // Générer une image damier avec du code

```
void makeCheckImage(void){  
GLubyte c;  
for(int i=0; i<256; i++)  
    for(int j=0; j<256; j++) {  
        c = (((((i&0x20)==0)^((j&0x20))==0))*255;  
        checkImage0[i][j][0] = checkImage0[i][j][1] =  
            checkImage0[i][j][2] = c;  
        checkImage0[i][j][3] = 255;  
    }  
}
```



Exemple:

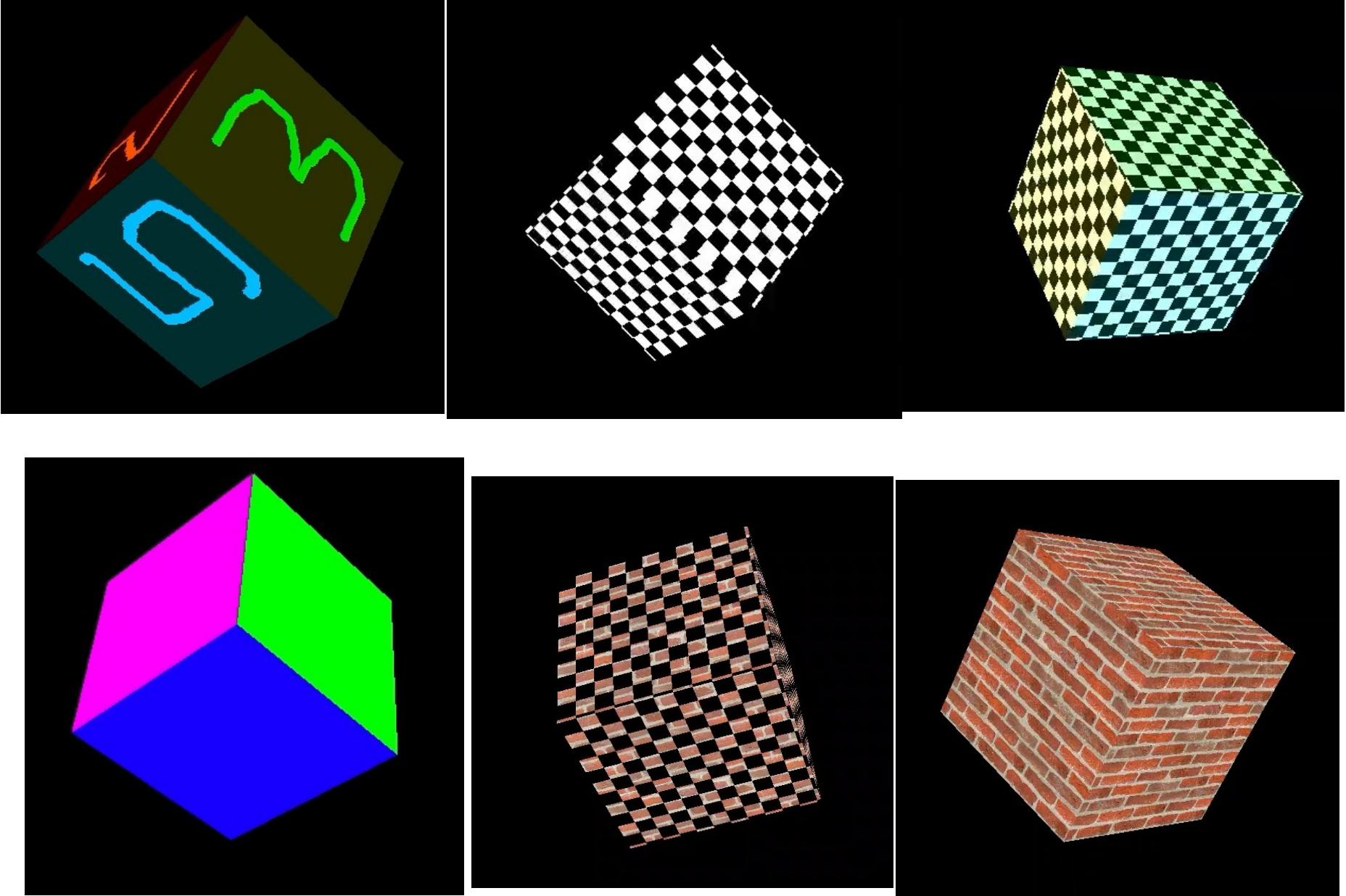
```
void GL_init(void)
{
    makeCheckImage();
    // spécifier la texture avec l'image
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGBA,
    GL_UNSIGNED_BYTE, checkImage0);
    // Comment la texture interagit avec la couleur du pixel
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    // filtres
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    // Activation du placage de texture 2D
    glEnable(GL_TEXTURE_2D);
    unsigned int tex = glGetUniformLocation(progId, "tex");
    glUniform1i(tex, 0); //variable uniform pour ref. unité 0 ds le shader
```

Exemple avec opencv:

```
void initTextures(){ cv::Mat img = cv::imread  
"../res/images/numbers.png",cv::IMREAD_UNCHANGED );  
  
auto rows = img.rows;auto cols = img.cols;auto data = img.data;  
unsigned int texture; glGenTextures( 1, &texture );  
  
glBindTexture( GL_TEXTURE_2D, texture );  
  
glActiveTexture( GL_TEXTURE0 );//automatique si une seule unité de  
texture  
  
unsigned int tex = glGetUniformLocation(progId, "tex");  
glUniform1i(tex,0);//variable uniform pour ref. unité 0 ds le shader  
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST );  
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );  
  
makeCheckImage();  
  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_BGRA,  
GL_UNSIGNED_BYTE, checkImage0);
```

2 Textures

```
GLuint texObject[2]; // def objets texture
glGenTextures(2, texObject); // génère 2 nouveaux objets de texture
glActiveTexture( GL_TEXTURE0 );//activation de l'unité de texture 0
 glBindTexture(GL_TEXTURE_2D, texObject[0]);// LierTexture 1
 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256,
0,GL_RGBA,GL_UNSIGNED_BYTE, image1);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
GLTexParameteri(...);
glActiveTexture( GL_TEXTURE1 );//activation de l'unité de texture 1
 glBindTexture(GL_TEXTURE_2D, texObject[1]);// Lier Texture 2
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexParameteri(...);
 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0, GL_RGBA,
GL_UNSIGNED_BYTE, image2);
unsigned int tex = glGetUniformLocation(progId, "tex");
 glUniform1i(tex,0);//variable uniform pour ref. unité 0 ds le shader
unsigned int tex2 = glGetUniformLocation(progId, "tex2");
 glUniform1i(tex2,1);//variable uniform pour ref. unité 1 ds le shader
```

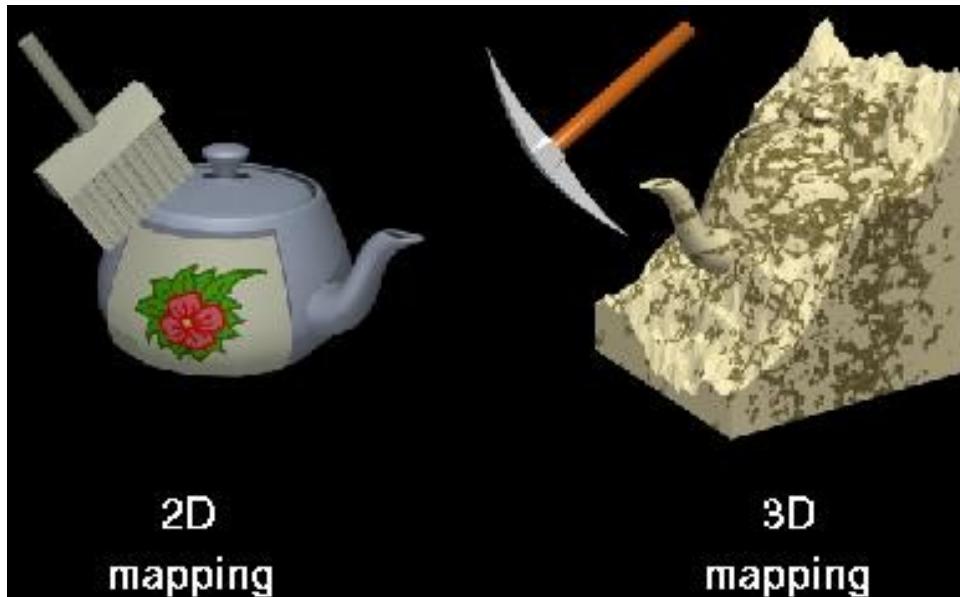


Textures 2D & 3D

- Depuis OpenGL 1.2, il est possible d'employer des textures 3D.
- Le mode opératoire et les fonctions correspondantes sont une simple généralisation des versions 1D et 2D.

Textures 2D & 3D

- Textures 2D : « décalcomanie »
- Textures 3D : « sculpture »
 - souvent exprimées de manière mathématique car trop volumineuse



texture 3D

1998:multi-textures

Texture de base



Light map



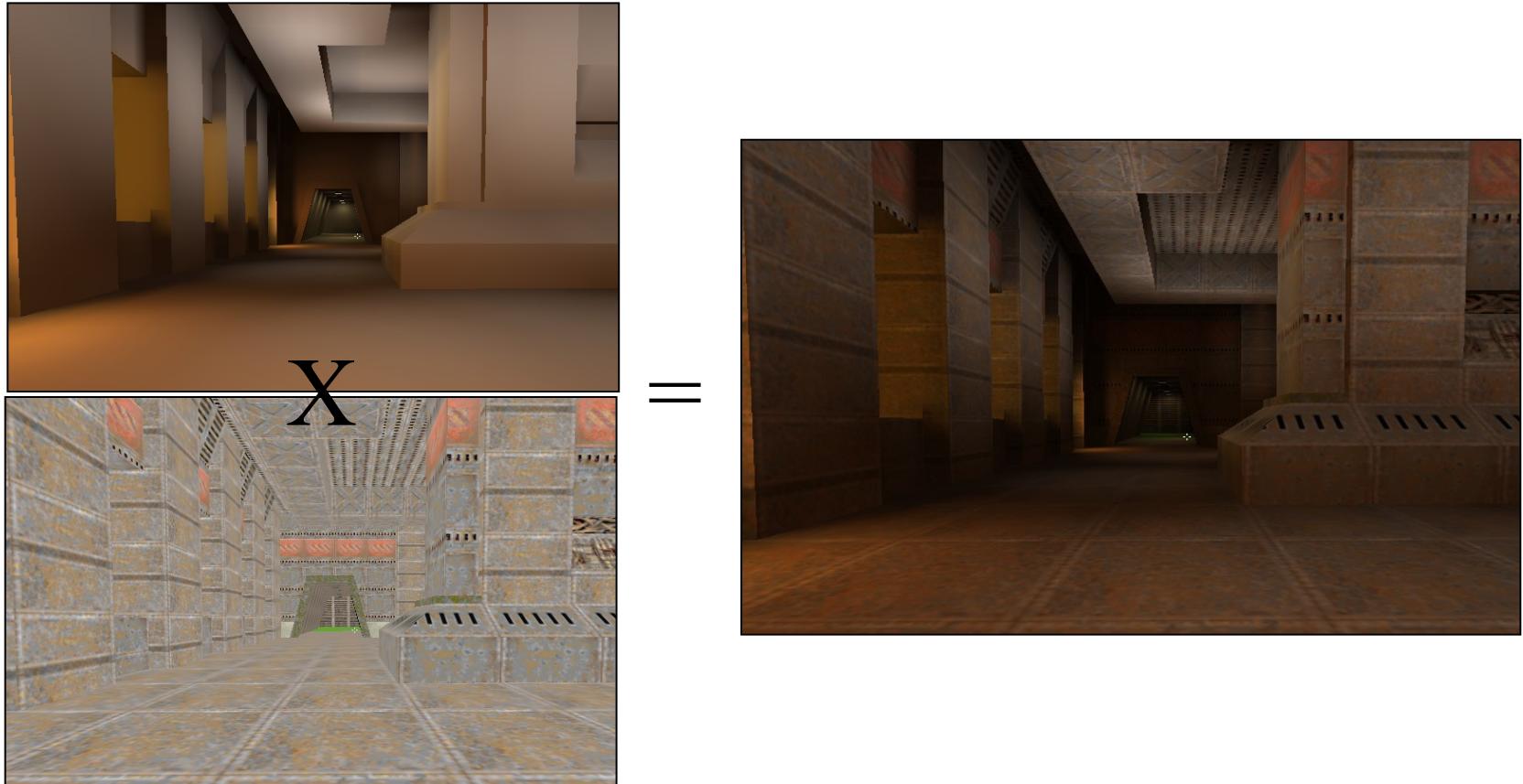
X

=

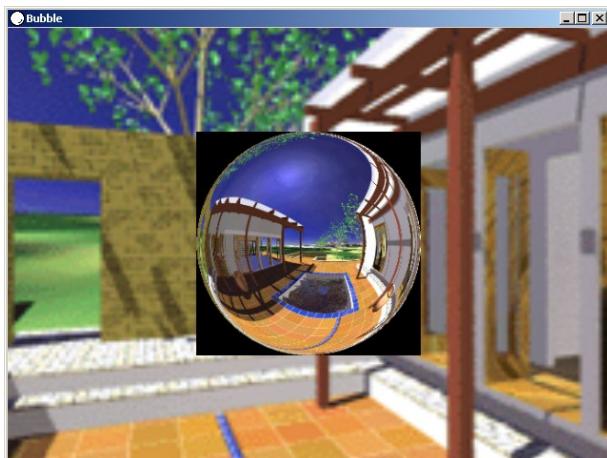


Textures multi-couches

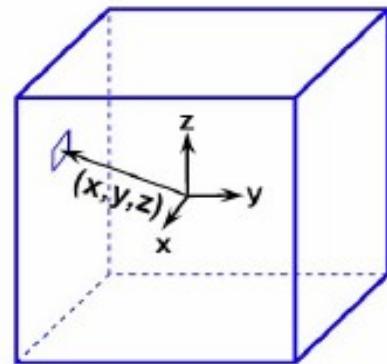
Light Map = produit d'une texture d'éclairage avec une texture de motif.



- Cube texture mapping



- Cube texture mapping



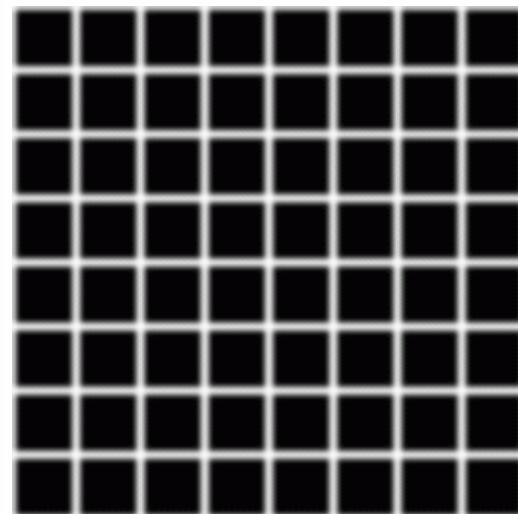
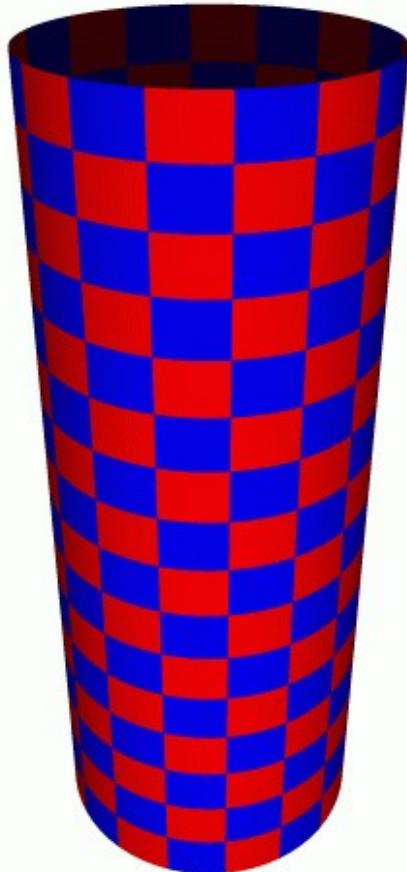
Plaquage simple



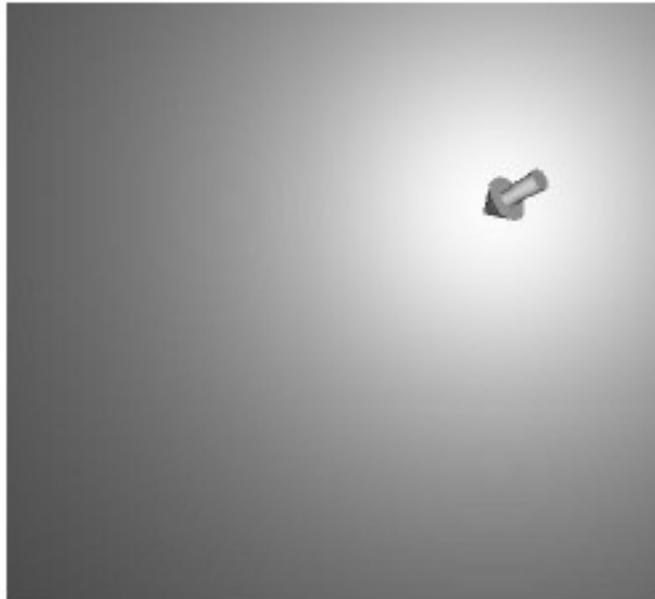
Et les sphere map:



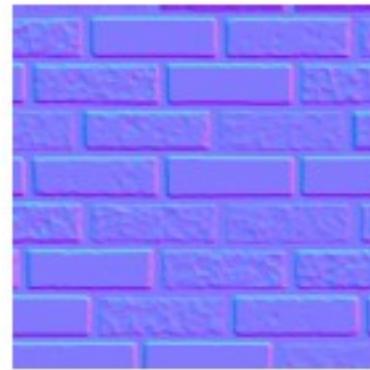
Bump mapping



Bump mapping



Éclairage diffus sans bump

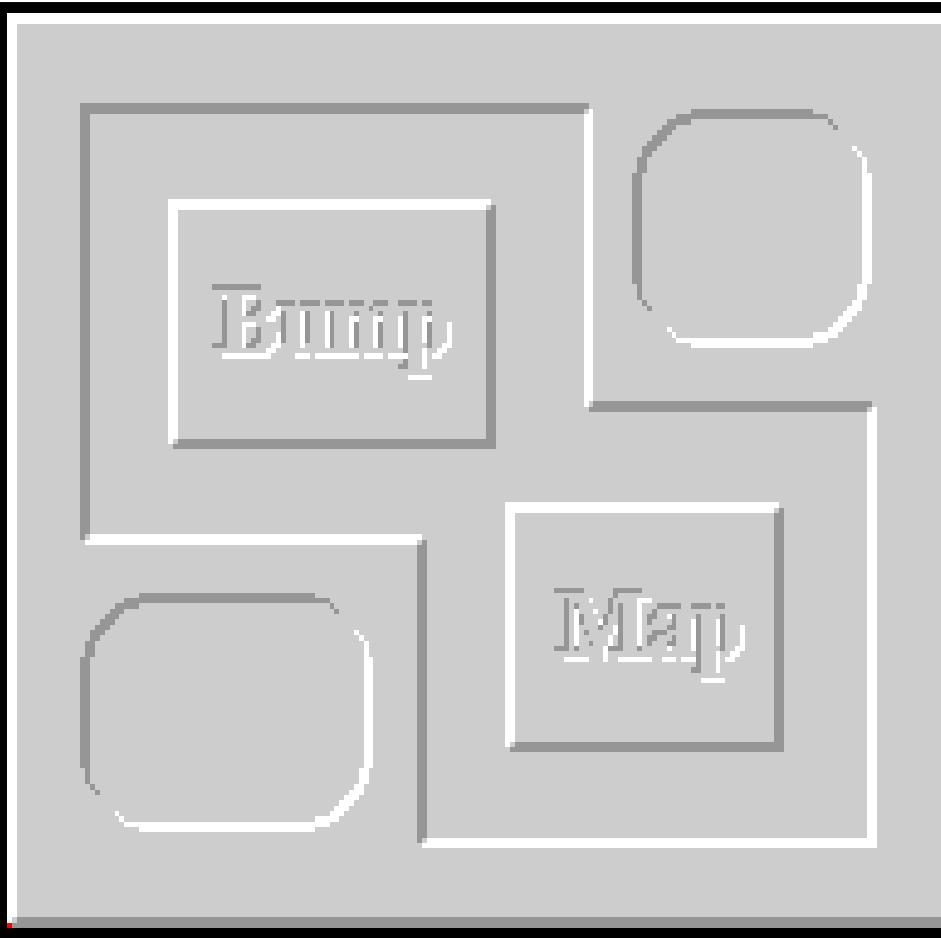


Normal map

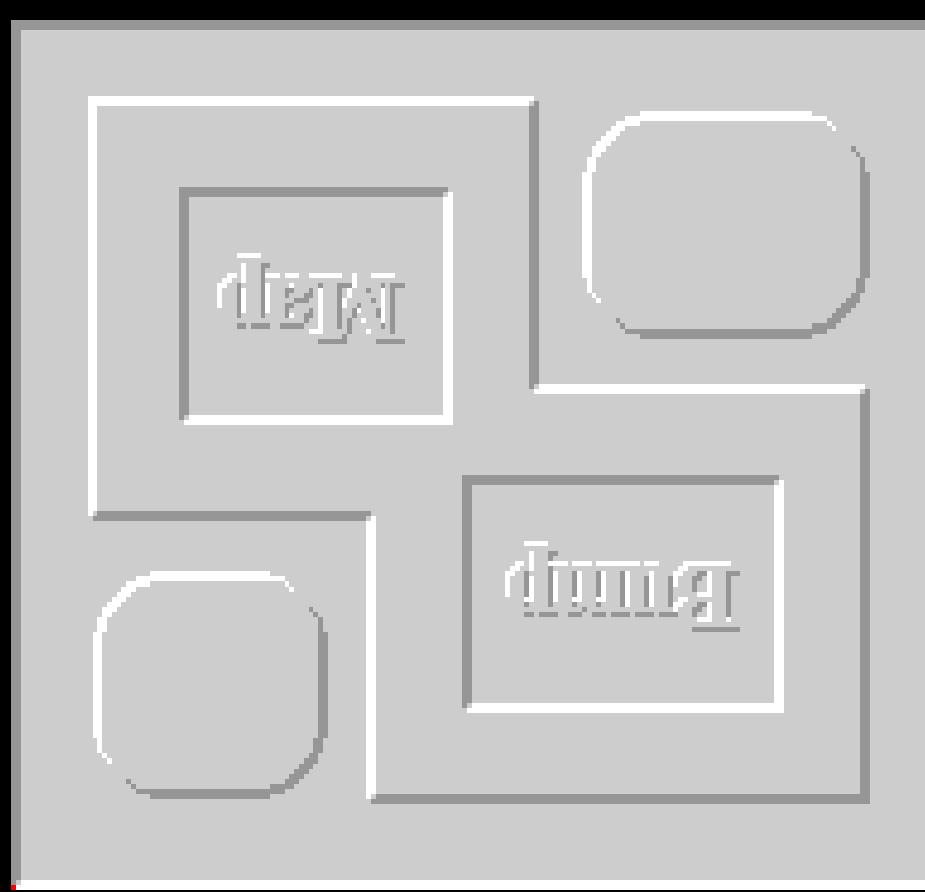


Éclairage diffus avec bumps

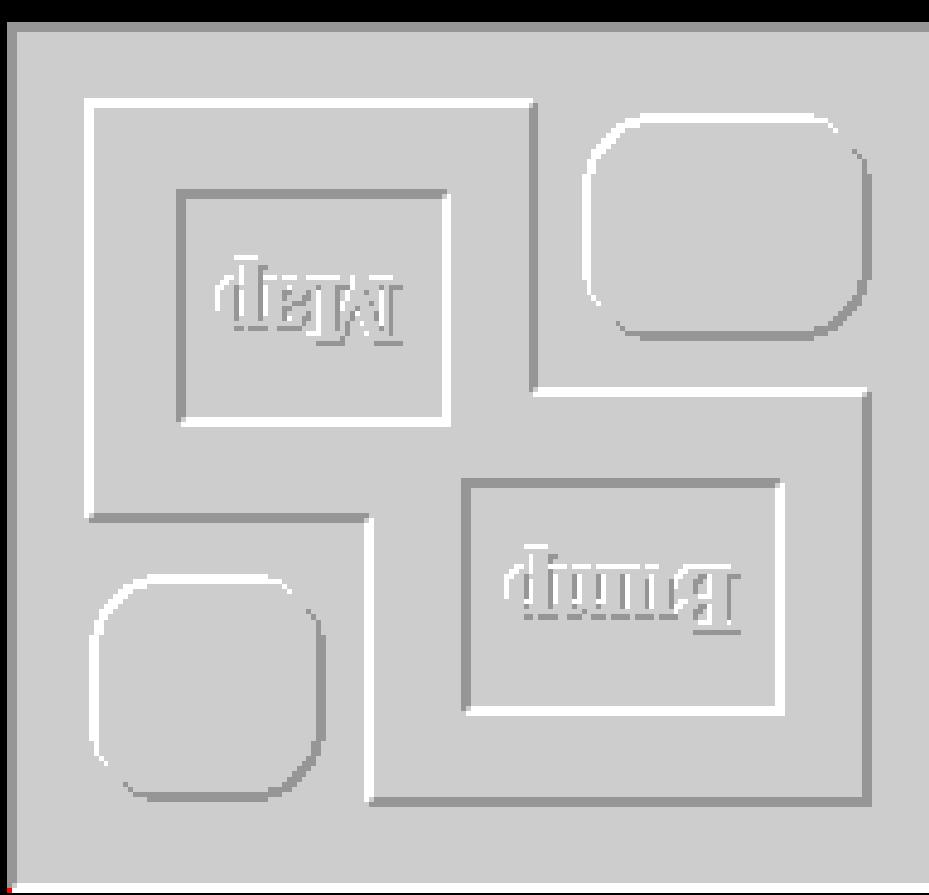
Où sont les creux?



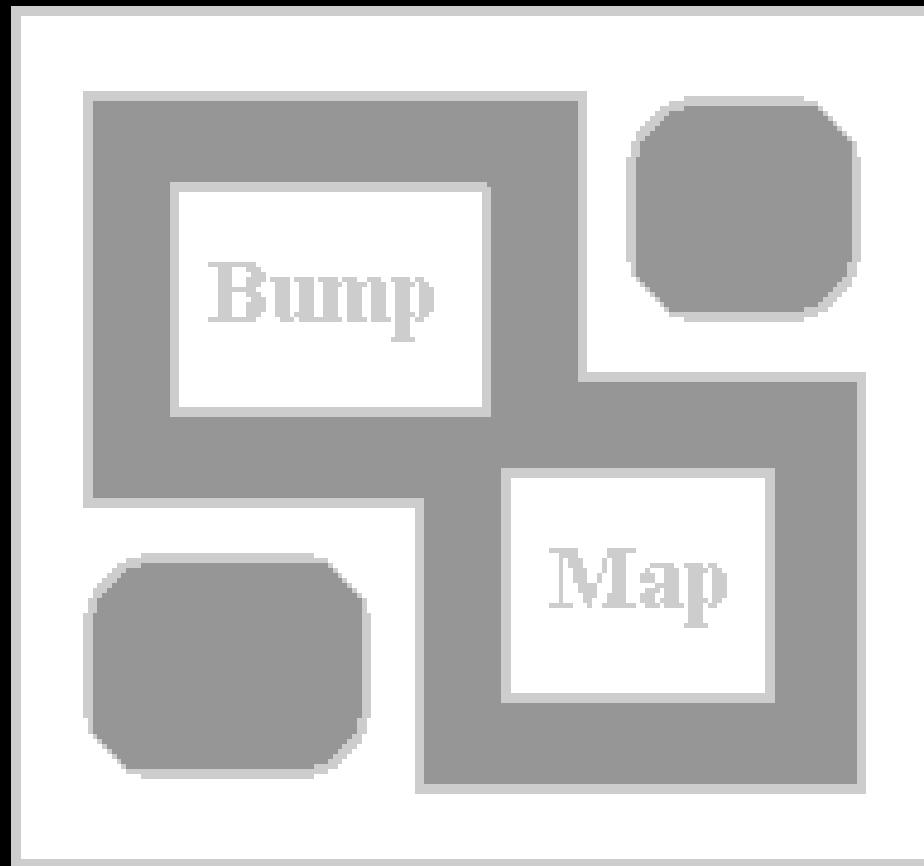
Où sont les ceux?



...et si la lumière vient du bas?

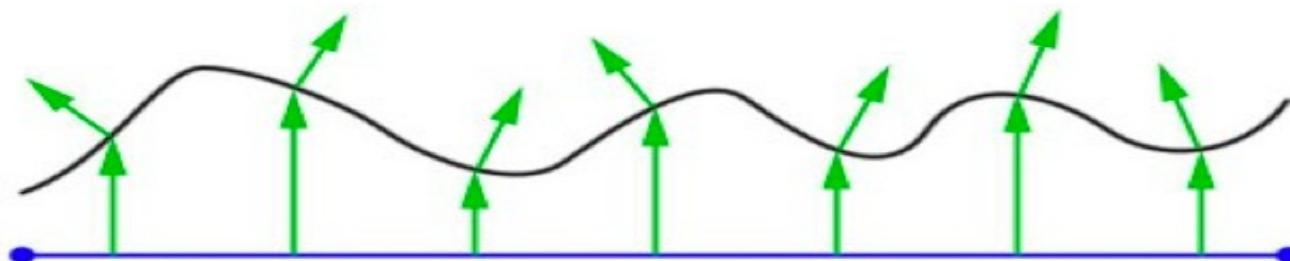


bump map = height field

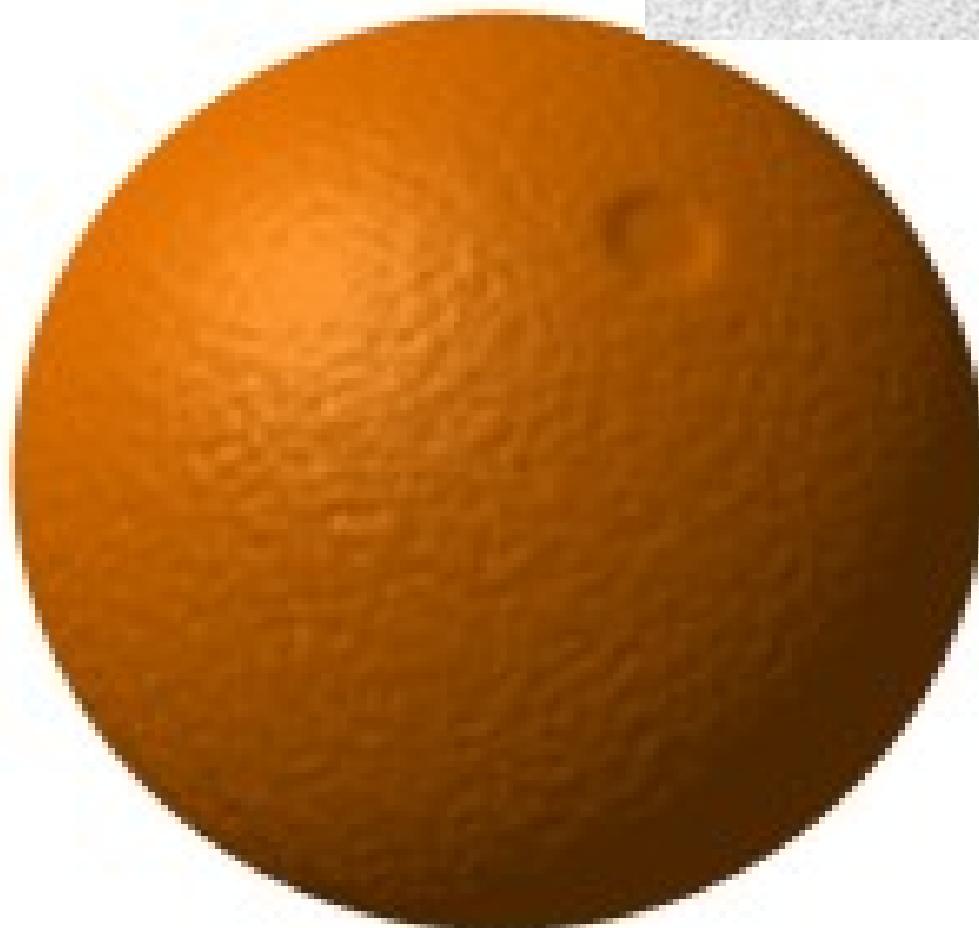
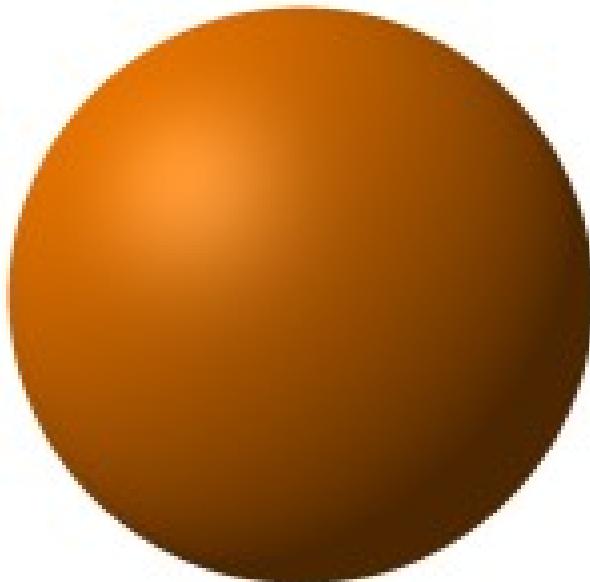
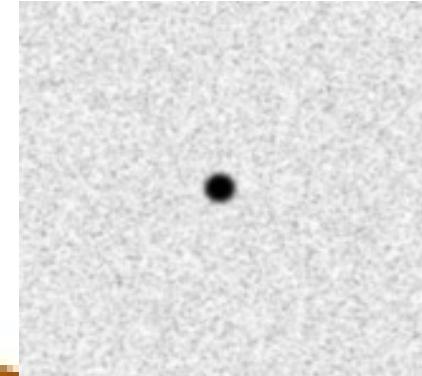
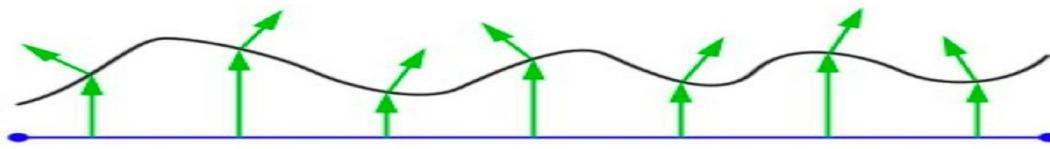


Bump mapping

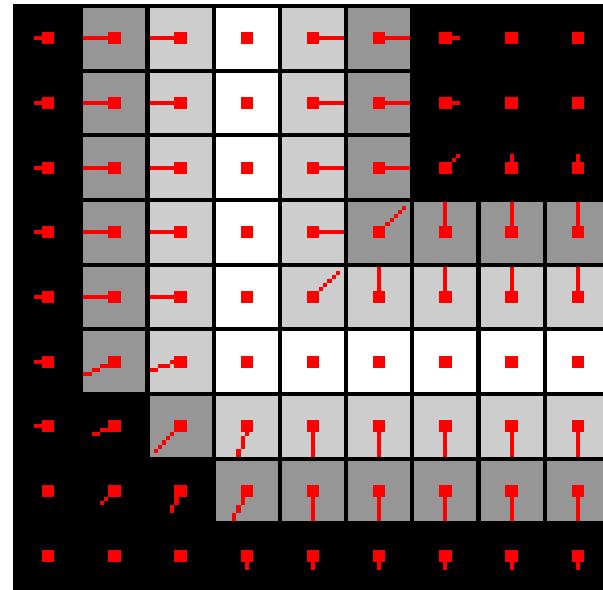
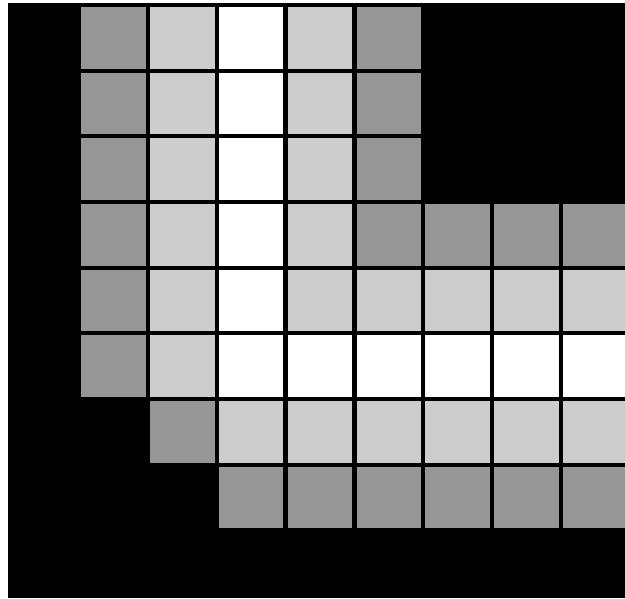
- Bump Mapping = perturbation de la normale en direction et amplitude
- Usage de texture pour savoir comment déplacer les normales.
- Les déplacements sont calculés par rapport au dérivées des textures
- Présupposés:
 - Les déplacements sont faibles (ils ne faudrait pas avoir à re-normaliser)
 - Les déplacements sont dans le plan tangent à la surface
 - La texture est plaquée sur les plans tangents à la surface



Bump mapping

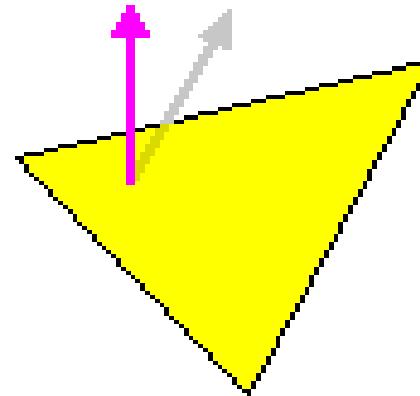
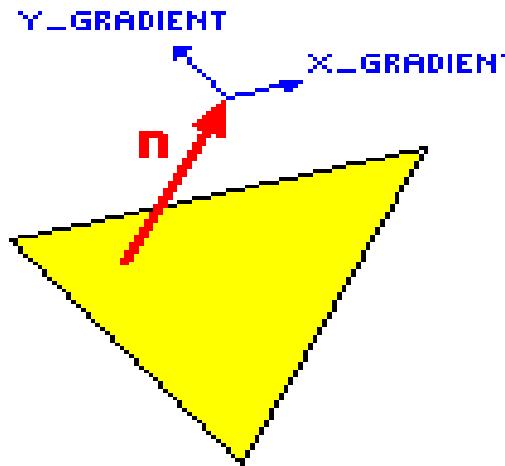


Exploitation de la bump map

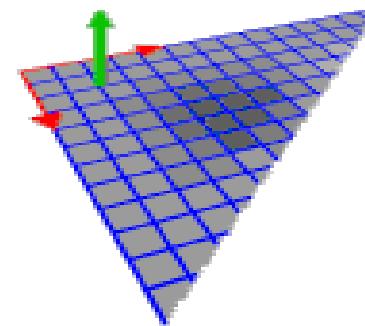
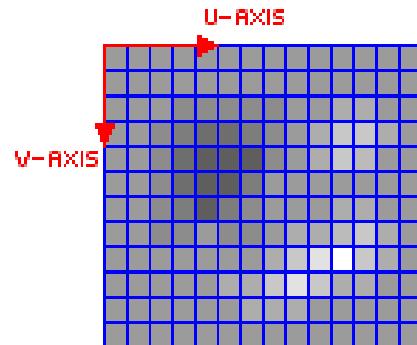


```
x_gradient = pixel(x-1, y) - pixel(x+1, y)  
y_gradient = pixel(x, y-1) - pixel(x, y+1)
```

Exploitation de la bump map



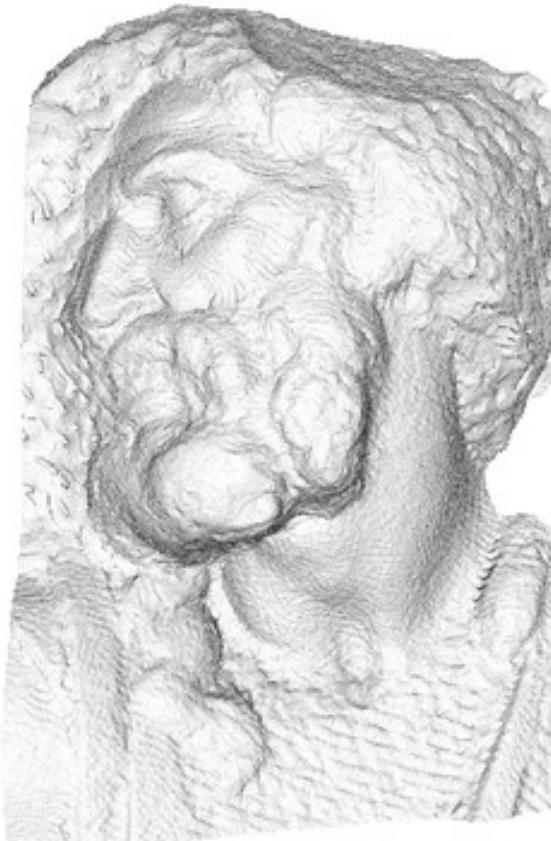
$\text{New_Normal} = \text{Normal} + (U * \text{x_gradient}) + (V * \text{y_gradient})$
U et V sont les vecteurs du système des coordonnées de textures



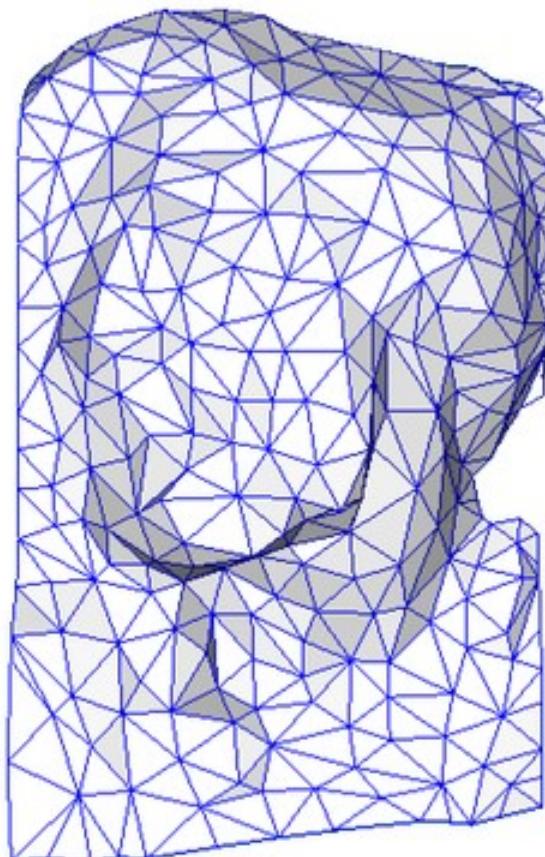
Application: Bump Mapping



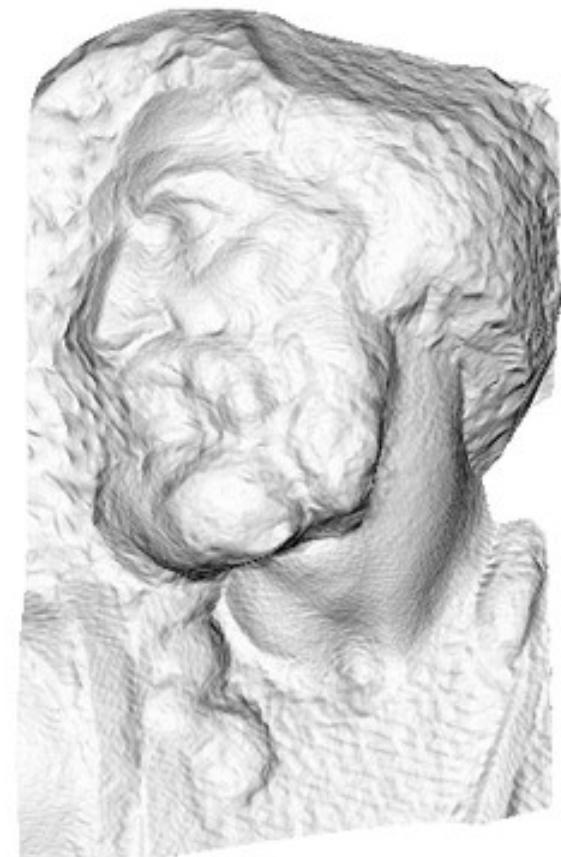
Application: Normal Mapping



original mesh
4M triangles



simplified mesh
500 triangles

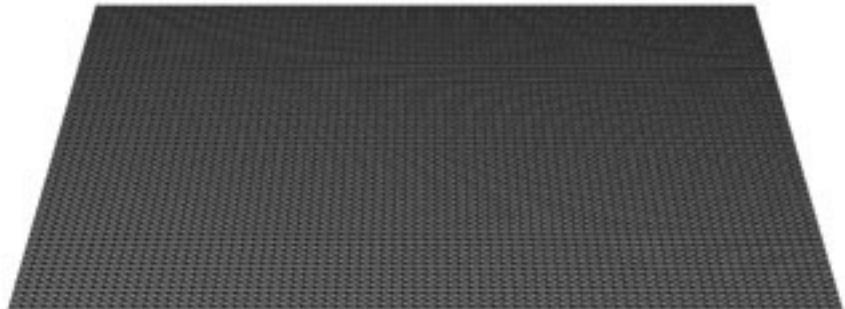


simplified mesh
and normal mapping
500 triangles

Displacement mapping

Displacement mapping :
Changement de la géométrie
en fonction d'une texture

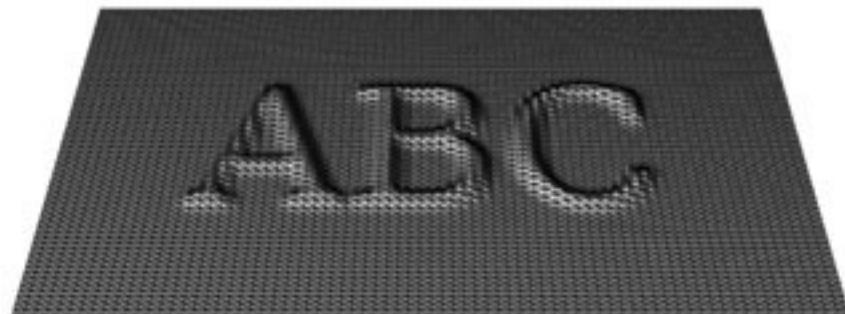
Supporté par les architectures
qui supportent le shade
model 3.0



ORIGINAL MESH



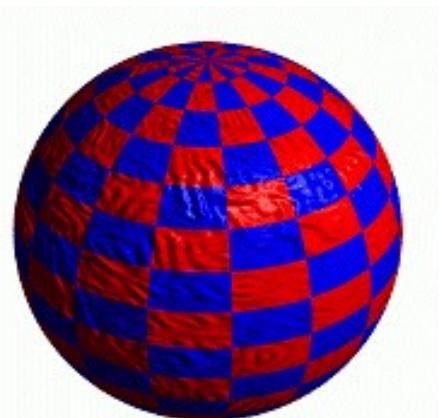
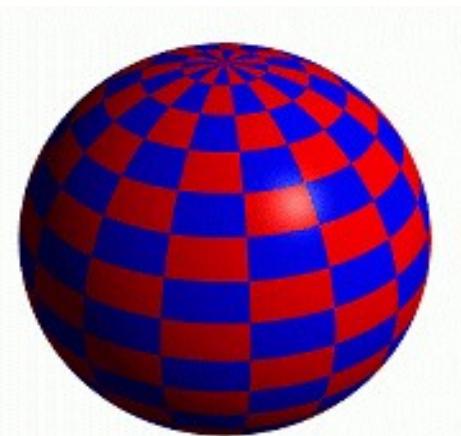
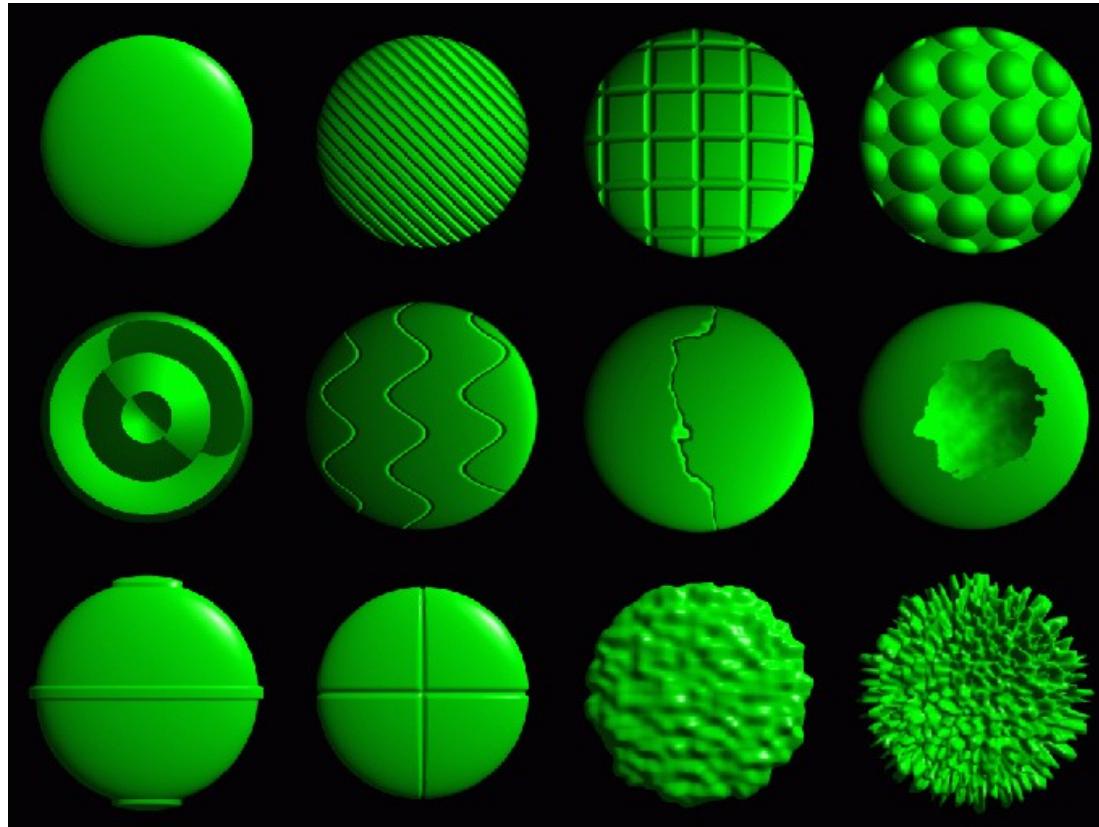
DISPLACEMENT MAP



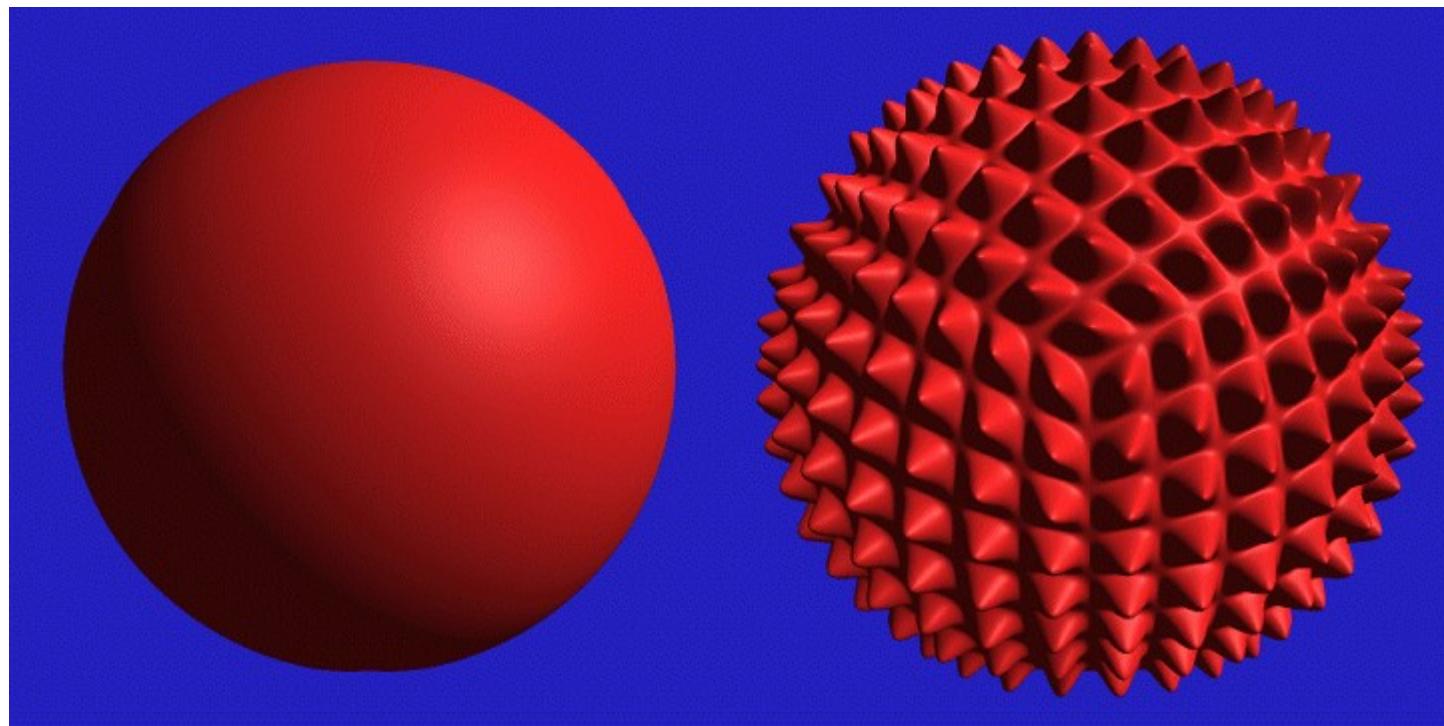
MESH WITH DISPLACEMENT

Effets de surface

- Apparence bosselée, rugueuse ...
- Displacement Mapping = bruitage de la surface originale



Displacement mapping



Effets de surface

