Rendu 3D – Les textures

David Murray

Source: Gael Guennebaud, Pierre Bénard

Modélisation d'aspect

- On sait comment calculer l'éclairage sur objet !
- Mais couleur et attributs défini par sommet…
  - Besoin d'une géométrie très fine
  - Très compliqué à construire et à éditer !
  - Faire des petits objets ? Pareil…
  - Solution: plaquer une image sur l'objet !
    - Texture !

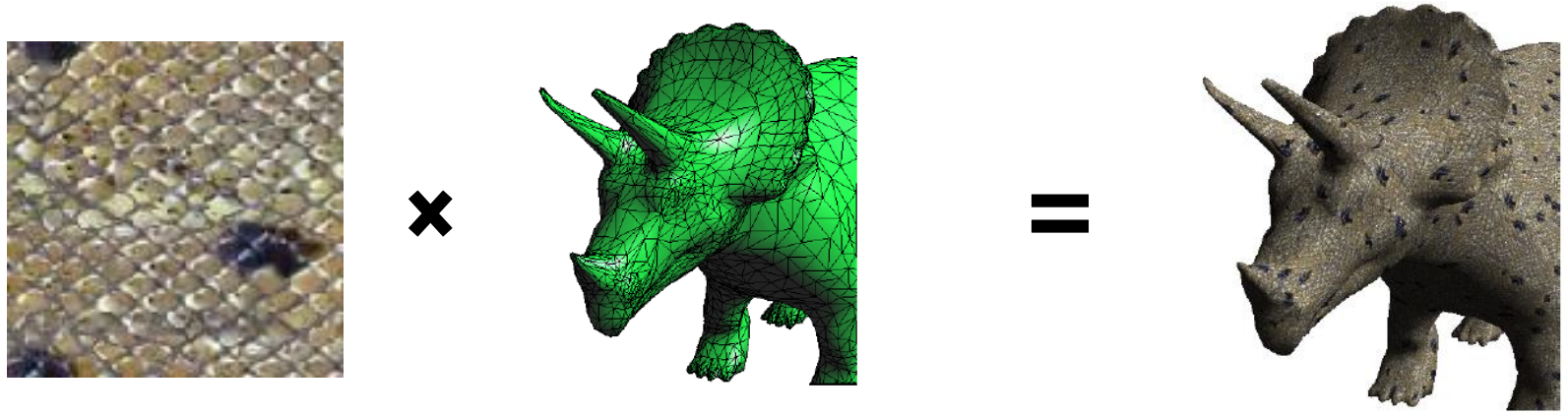
Les textures: la base

- Texture = champ scalaire discret

- Défini sur un domaine:
  - Linéaire (1D)
  - Rectangulaire (2D)
  - Cubique (3D)

Les textures: la base

- Comment plaquer une texture sur un objet: Coordonnées de texture
  - Spécifiées par sommet
  - Interpolées par fragment
    - Comme les normales !
    - Support materiel ☺



Les textures: pour quels attributs ?

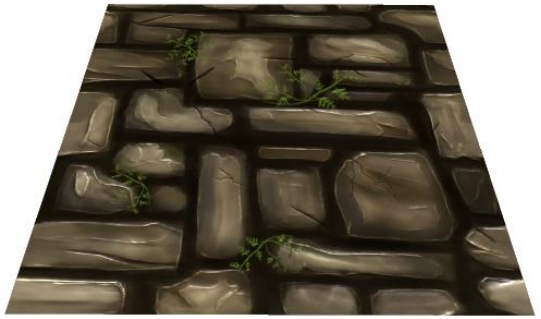
- La couleur : contribution diffuse, spéculaire, la rugosité, la transparence…

Diffuse map

+

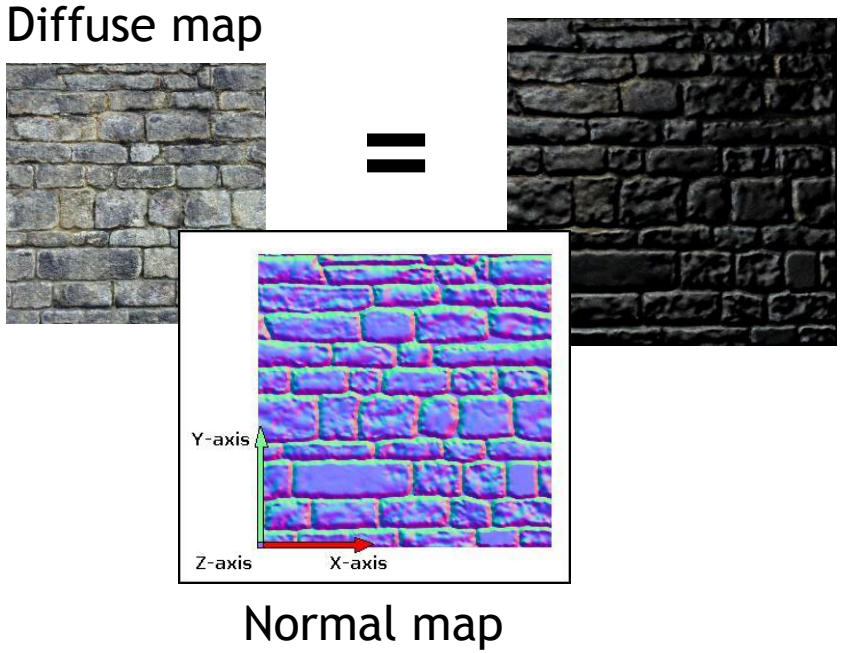
Specular map

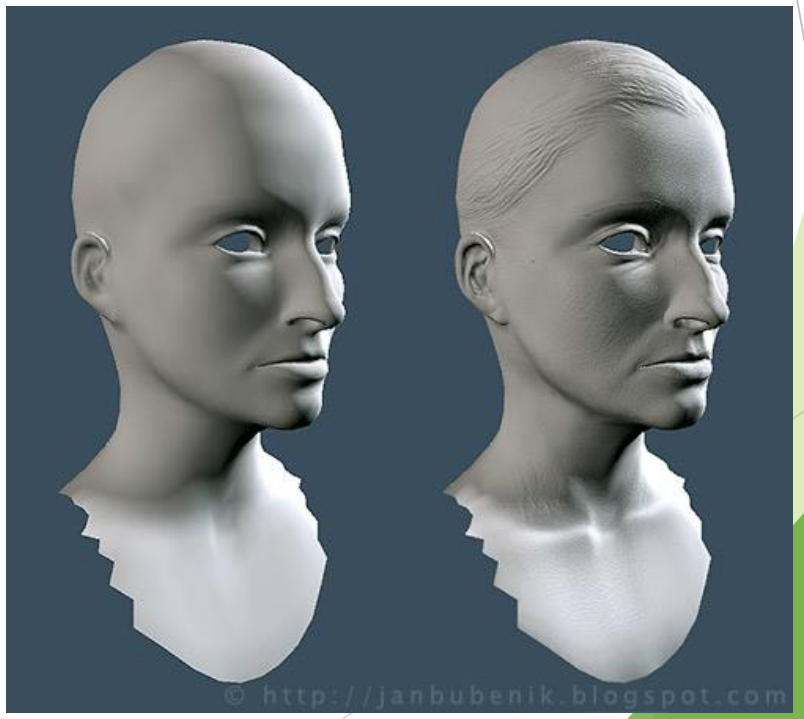
=



Les textures: pour quels attributs ?

- Attributs géométrique: normales (normal map)
- Permet de rajouter du relief sans complexifié un maillage



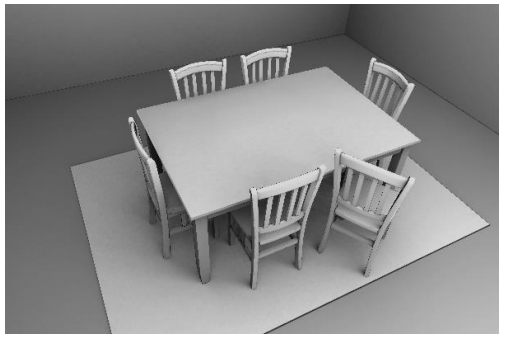


Les textures: pour quels attributs ?

- Illumination précalculé (Light Map)

Diffuse map

×

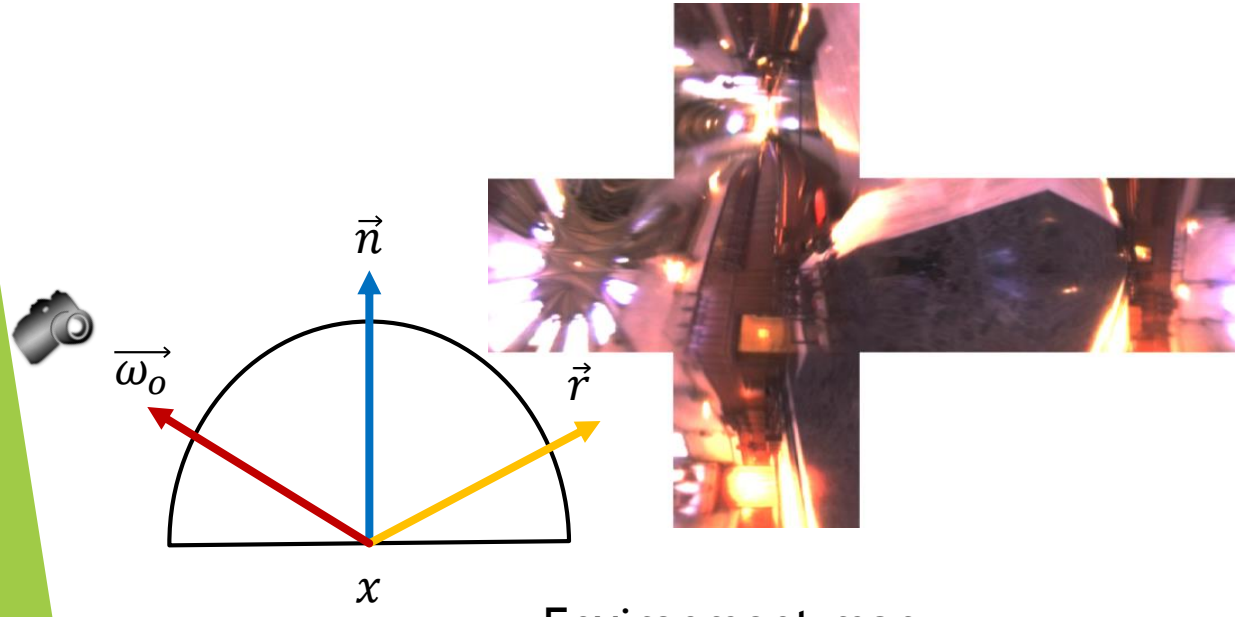


Light map

=

Les textures: pour quels attributs ?

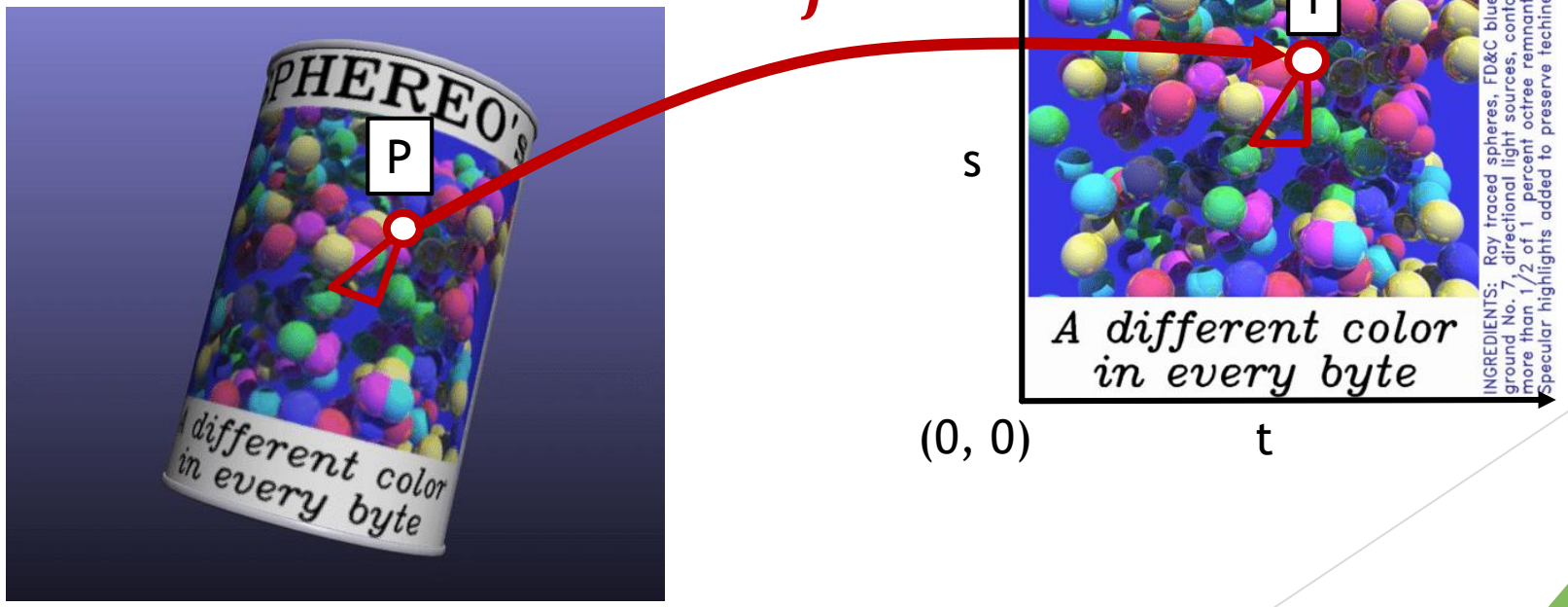
- Simuler un environnement -> une texture cubique autour de l'objet



Environment map

Les textures: comment on s'en sert ?

- Texture 2D -> Image I(s, t)
- Fonction de plaquage (mapping) :
  - $f: P(x, y, z) \rightarrow I(s, t);\ [0,1]$

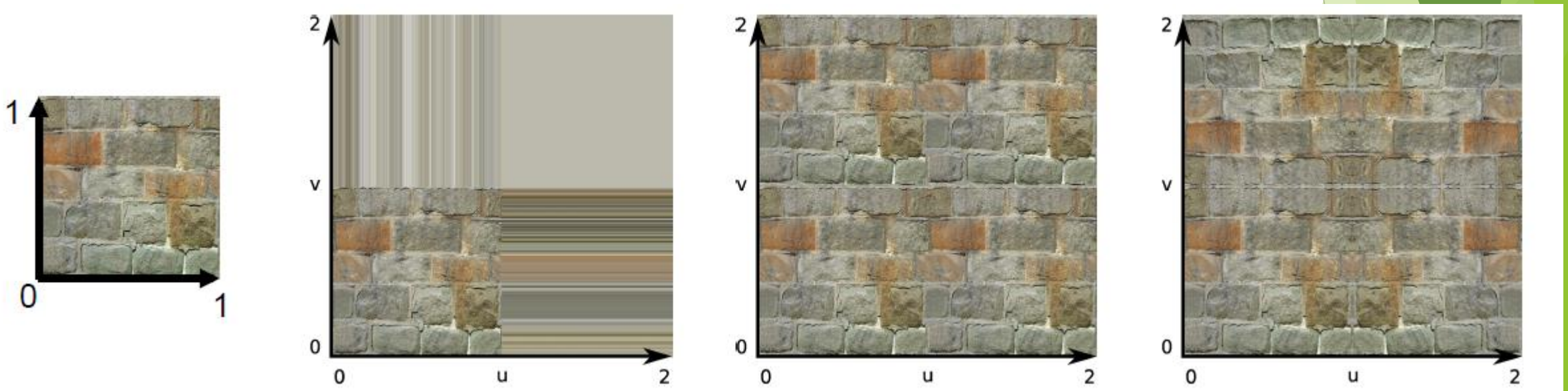


Les textures: comment on s'en sert ?

- Fonction de plaquage, exemples :
- Planaire $f(x, y, z) = (\|x\|, \|y\|)$
- Cylindrique $f(\theta, y) = \left(\frac{\theta}{2\pi}, y\right)$
  - $\theta = \text{atan}\left(\frac{z}{x}\right)$
- Sphérique $f(\theta, \phi) = \left(\frac{\theta}{2\pi}, \frac{\phi}{\pi}\right)$
  - $\theta = \text{atan}\left(\frac{z}{x}\right); \phi = \text{a}sin(y)$

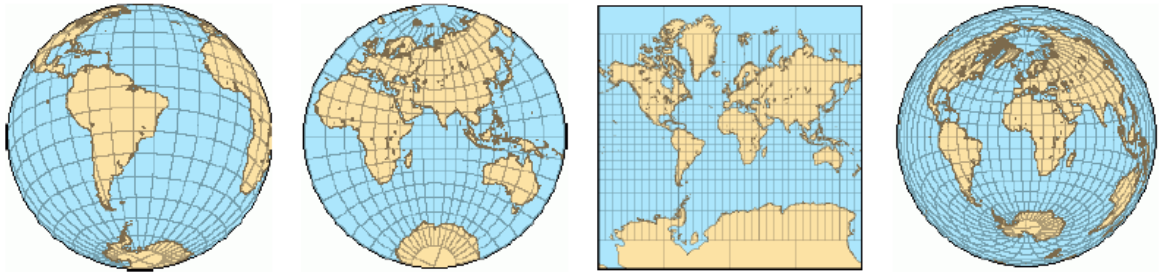
Problème au bord

- Comment interpréter une texture HORS de [0, 1] ?
  - « GL_CLAMP_TO_EDGE »: $s = s < 0 ? 0 : (s > 1 ? 1 : s)$
  - « GL_REPEAT »: $s = s - \lfloor s \rfloor$
  - « GL_MIRRORED_REPEAT »

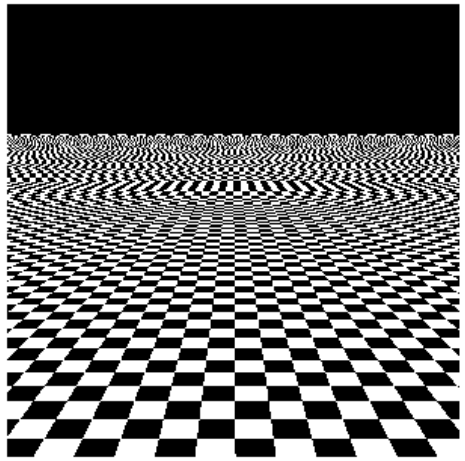


Textures: Problème et limites

- Distorsion
  - Dépend du mapping

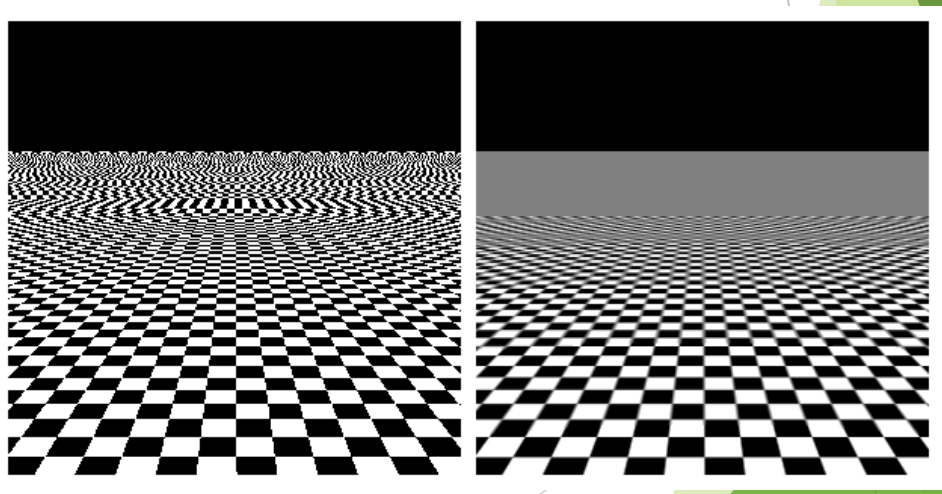


- Crénelage:
  - Plusieurs éléments sont projeté sur le même pixel



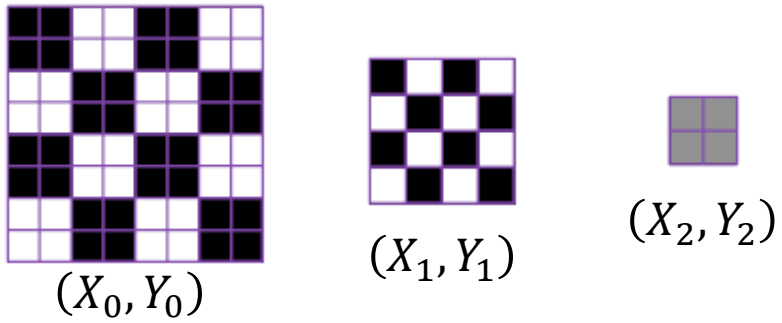
Textures: Aliasing

- Crénelage:
  - Plusieurs éléments sont projeté sur le même pixel
- Plusieurs solution:
  - Sur-échantillonnage -> couteux
  - Filtrage spatial
    - MIP-Mapping



Textures: MIP Mapping

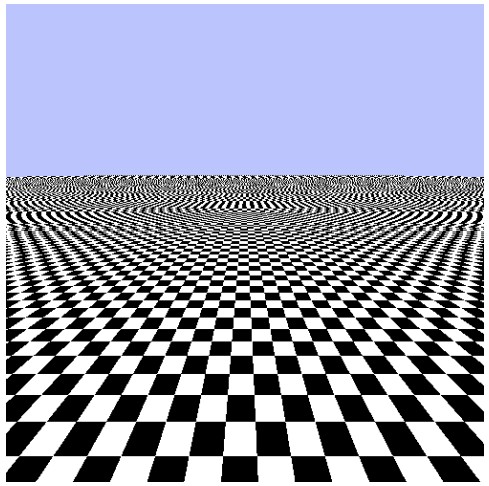
- Idée : créer une pyramide d'images pré-filtrées
  - Chaque étage a une résolution $(X_i, Y_i) = \left(\frac{X_{i-1}}{2}, \frac{Y_{i-1}}{2}\right)$



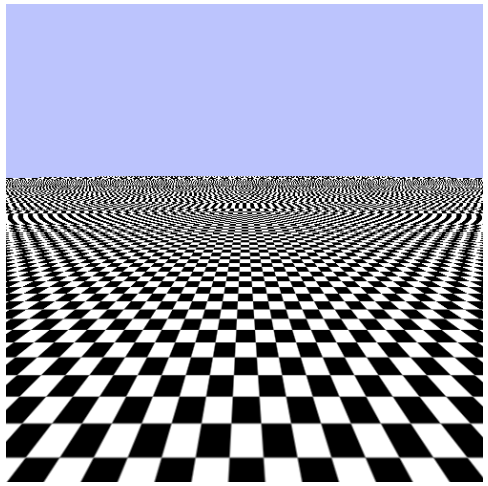
  - On choisit le niveau à utiliser en fonction de la distance
    - 0 = proche
    - N = éloigné

Textures: MIP Mapping

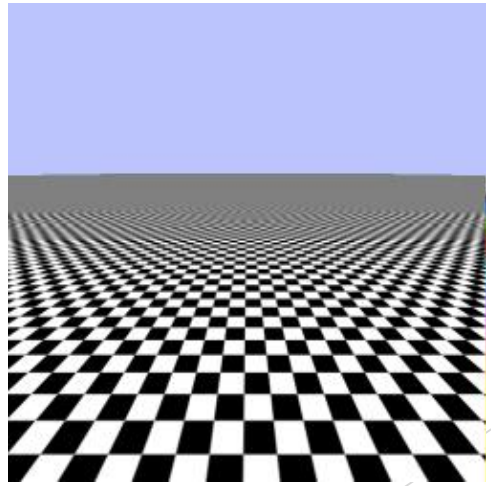
- Comment les calculer ?
  - Génération automatique par OpenGL
  - Plusieurs modes de calcul



Plus proche voisin
« GL_NEAREST »



Linéaire
« GL_LINEAR »



MIP Mapping
« GL_LINEAR_MIPMAP_LINEAR »

En pratique: En C++

- Créer une texture :
  - glGenTexture : crée un identifiant pour votre texture
  - glBindTexture : lie votre texture avec le type souhaité (1D, 2D…)
  - glTexParameter : spécifie les paramètres (filtrage, répétition…)
  - glTexImage2D : envoie les données de la texture (pour du 1D et 2D)
  - glGenerateMipmap : génère les niveaux de mip map

En pratique: En C++

- Afficher une texture :
  - glActiveTexture : spécifie quelle unité de texture utiliser pour afficher la texture
  - glBindTexture : lie la texture à l'unité
- Ne pas oublier :
  - glUniform1i(samplerLocation, activeUnit)
  - samplerLocation -> à récupérer comme pour les matrices
  - activeUnit -> l'unité de texture que vous utilisez
  - A noter : uniform sampler2D -> un entier !

A vous !

- Exemple pour l’affichage de deux 2 textures:

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, firstTex);

glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, secondTex);

glUnidorm1i(firstTexLoc, 0);
glUnidorm1i(secondTexLoc, 1);
```

En pratique: En GLSL

- Déclaration : uniform sampler2D toto
- Accès avec la fonction :
  - vec4 texture(toto, texCoord)
- Exemple

```
// vertex shader
uniform mat4 mvp;
in vec4 vtx_position;
in vec2 vtx_texcoord;
out vec2 texcoord;
void main(void) {
	texcoord = vtx_texcoord;
	gl_Position = mvp * vtx_position;
}
```

```
// fragment shader
uniform sampler2D image;
in vec2 texcoord;
out vec4 out_color;
void main(void) {
	out_color = texture(image,texcoord);
}
```

A vous !