

<http://tinyurl.com/ift3355>

IFT 3355: INFOGRAPHIE LES TEXTURES

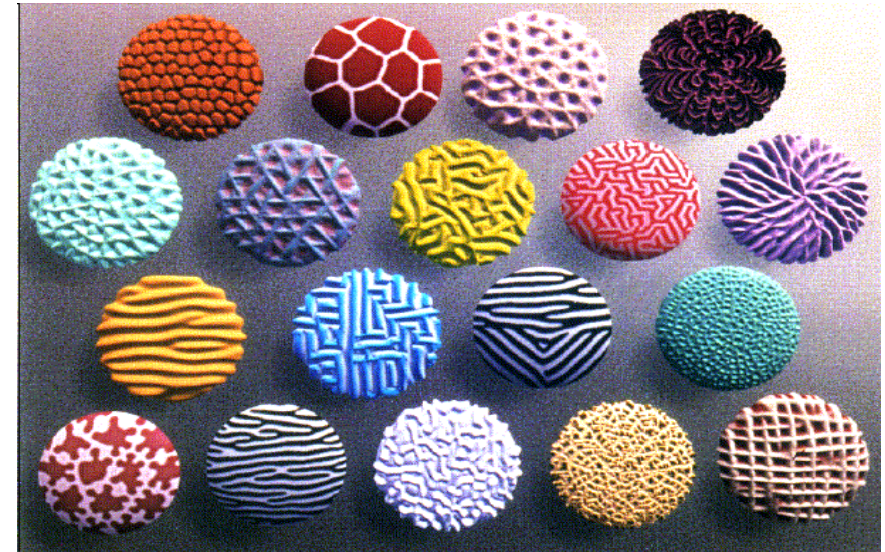
Livre de référence: G:11;
S: 15, 13 (optionnel)

Mikhail Bessmeltsev



TEXTURE MAPPING

- Les objets réels ont des couleurs et des normales non uniformes
- On peut simuler l'effet de ces détails via une **texture**
- Une texture peut souvent remplacer les détails géométriques complexes

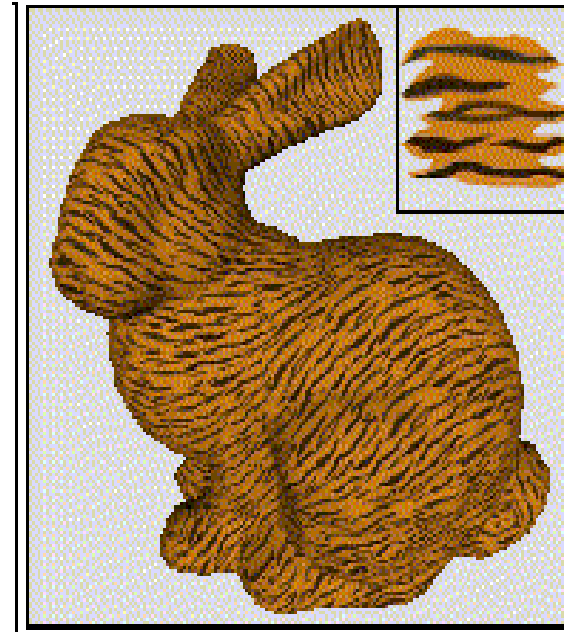
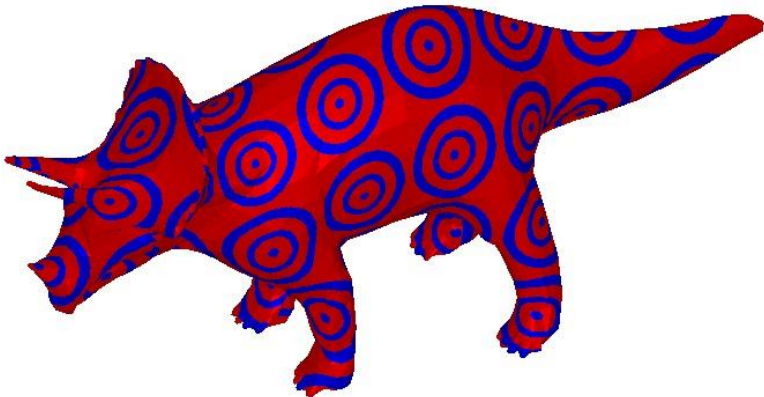


TEXTURE MAPPING

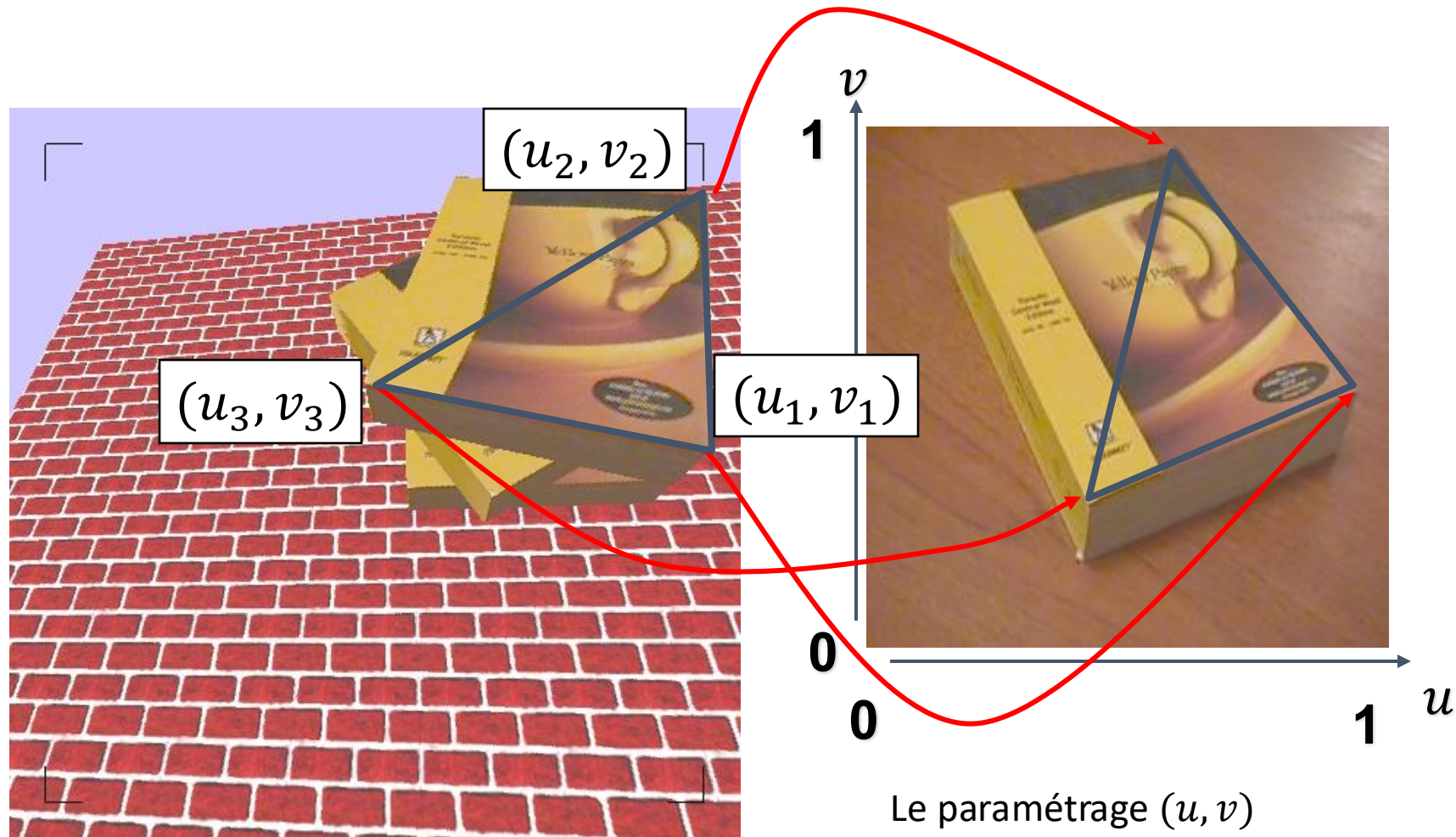
- Cacher la simplicité de la géométrie
 - « Papier peint sur une surface »
 - Apposer une image du mur de briques sur un polygone plat
 - Créer un effet bosselé sur la surface
- Normalement:
Associer l'information 2D (image) à l'information 3D (la surface)
 - Un point de la surface $\leftrightarrow$ un point de la texture
 - Peindre une image sur le polygone

TEXTURE MAPPING DE LA COULEUR

- Définir la couleur (RVB) pour chaque point de la surface
- Les autres méthodes:
 - Les textures volumétriques
 - Les textures procédurales

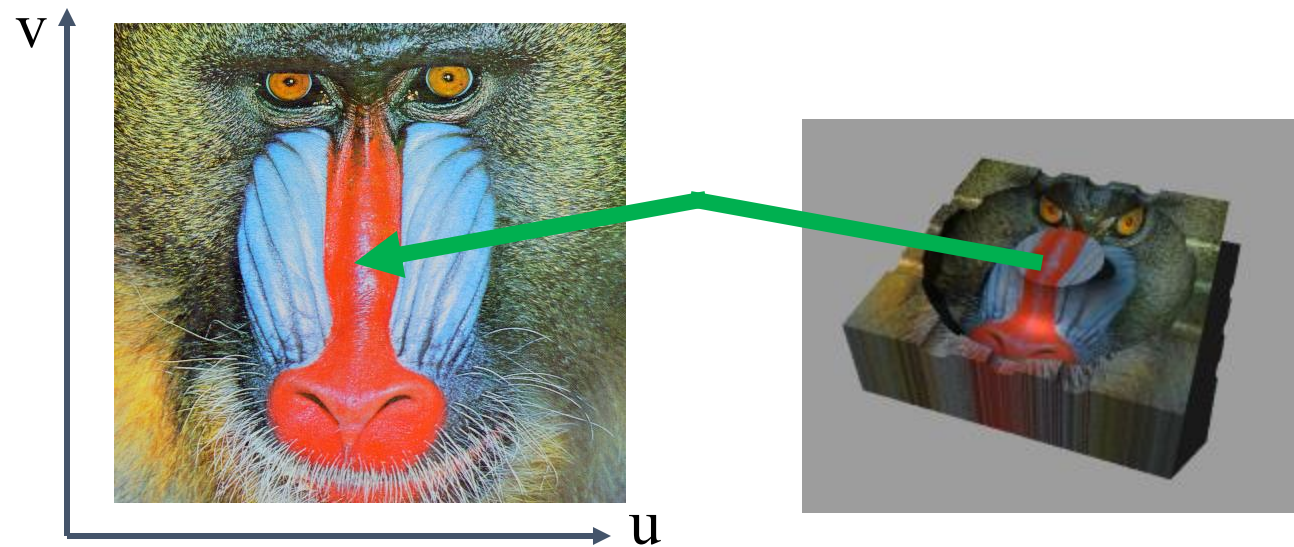


TEXTURE MAPPING

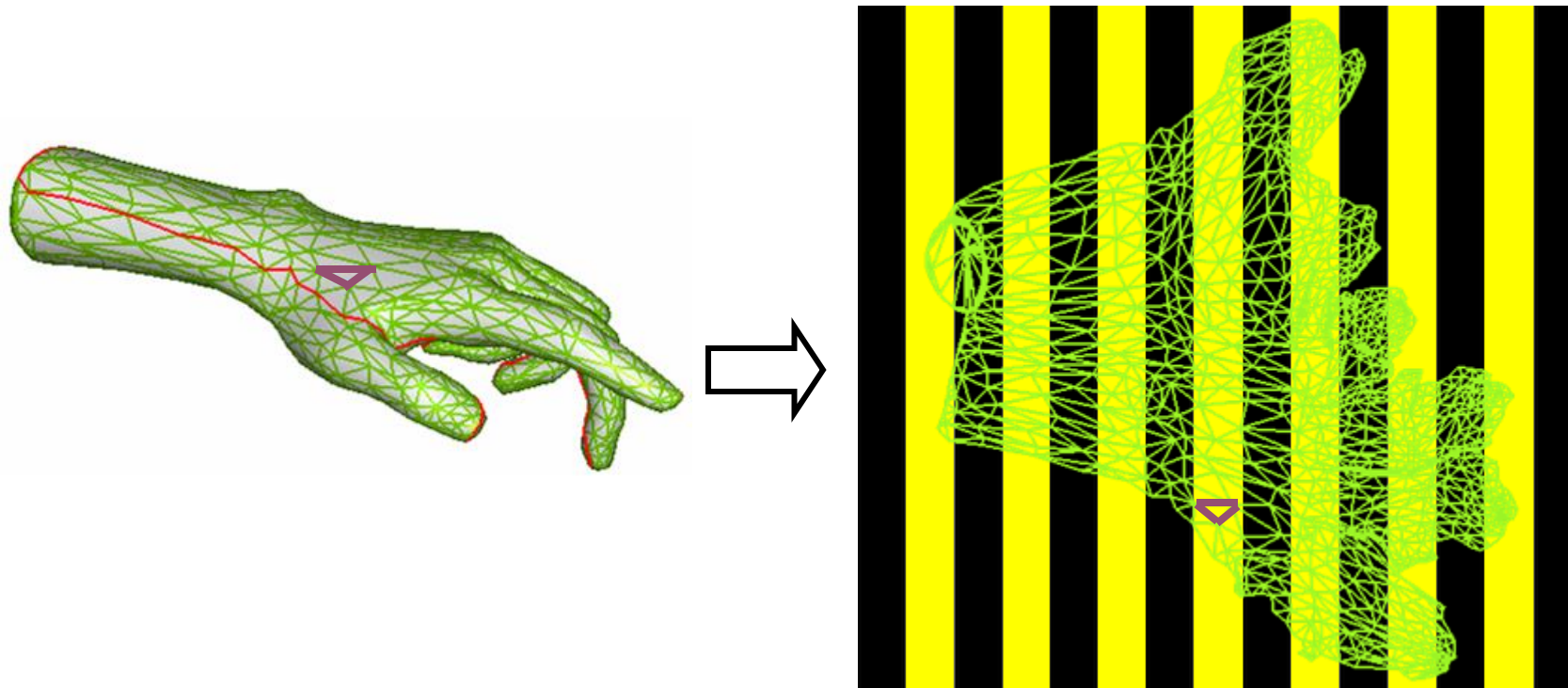
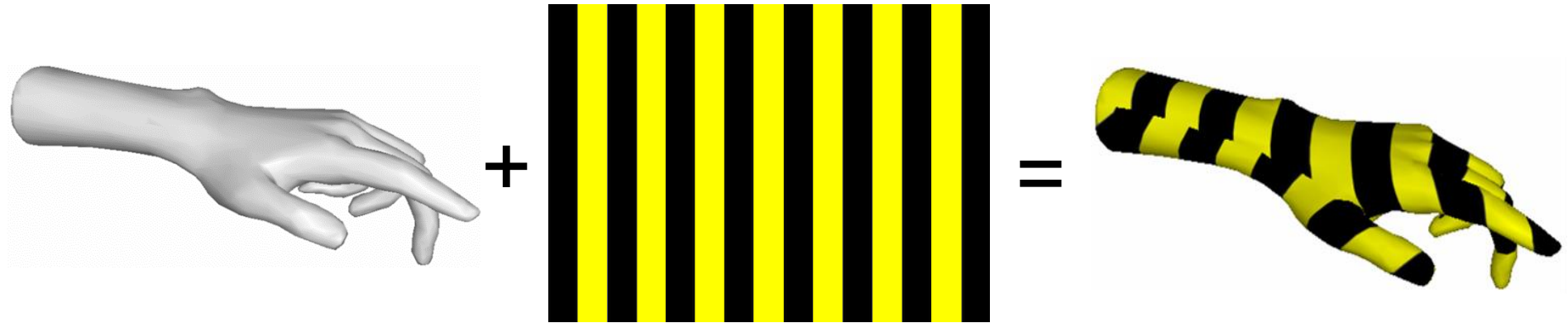


UNE TEXTURE DE LA SURFACE

- Définir la texture dans le 2D domaine (u, v) (l'image)
 - L'image = un tableau 2D des *texels* (*texture elements*)
- Attribuer les coordonnées (u, v) à chaque point de la surface de l'objet
 - Comment: les expressions analytiques pour les surfaces simples; les algorithmes complexes en général
- Pour l'intérieur des triangles
 - Utiliser les coordonnées barycentriques

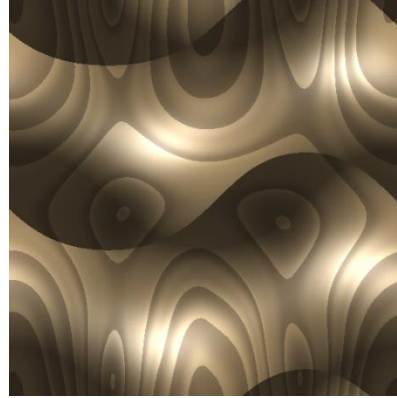


EXEMPLE DE *TEXTURE MAPPING*



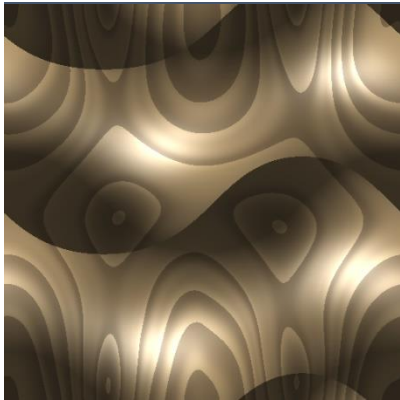
LES COORDONNÉES DE LA TEXTURE FRACTIONNAIRES

**texture
image**



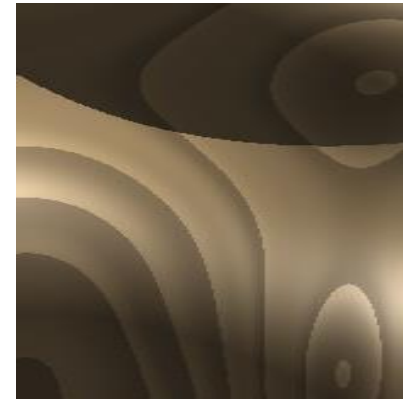
(0,1)

(1,1)



(0,0)

(1,0)



THREE.JS

- Passer la texture comme une *uniform*:

```
var uniforms = {  
  texture1: { type: "t", value: THREE.ImageUtils.loadTexture( "texture.jpg" ) } };  
var material = new THREE.ShaderMaterial( { uniforms, ...} );
```

- Les coordonnées *uv* seront passées au *vertex shader* (pas besoin d'écrire ça):

```
attribute vec2 uv;
```

- Utilisez-les, e.g., dans *Fragment Shader*:

```
uniform sampler2D texture1;  
varying vec2 texCoord;  
vec4 texColor = texture2D(texture1, texCoord);
```

COMMENT UTILISER LES TEXTURES DE COULEUR

- Remplacer

- Définir la couleur du fragment comme la couleur de la texture

```
gl_FragColor = texColor;
```

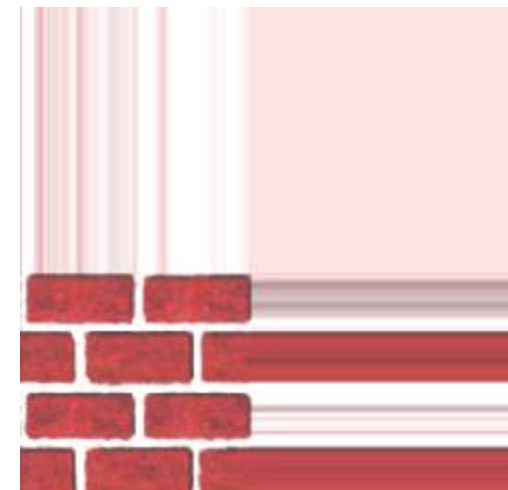
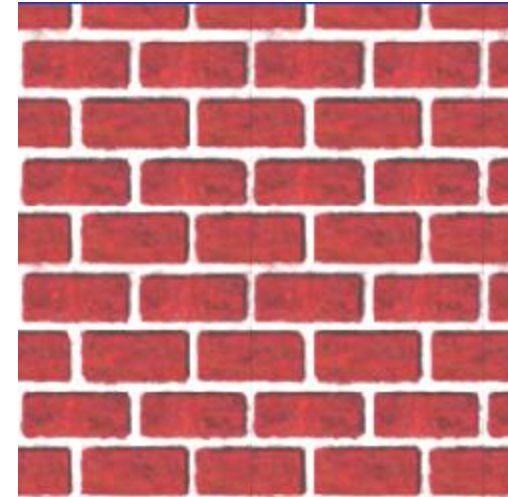
- Moduler

- Utiliser la couleur de la texture comme les coefficients de réflexion

```
kd = texColor; ka = texColor;  
gl_FragColor = ka*ia + kd*id*dotProduct + ...;
```

TILING ET CLAMPING

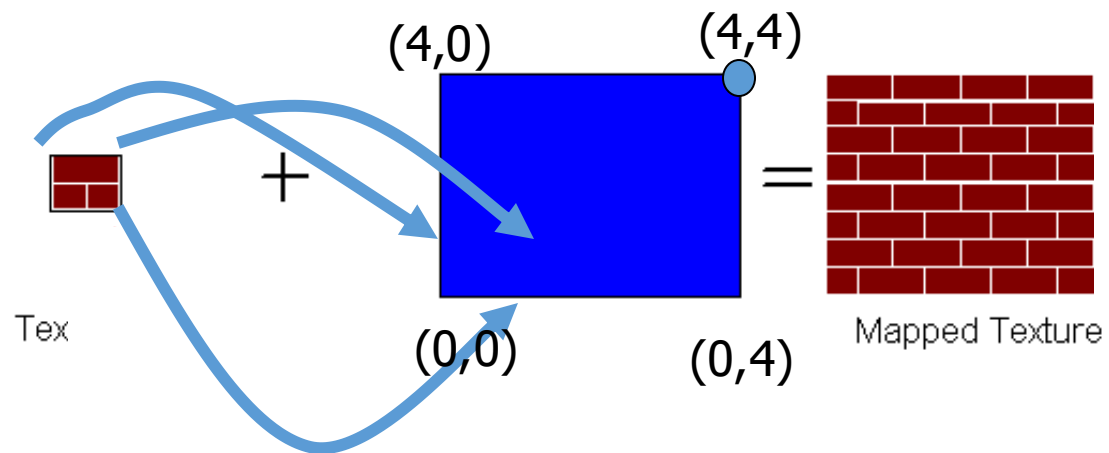
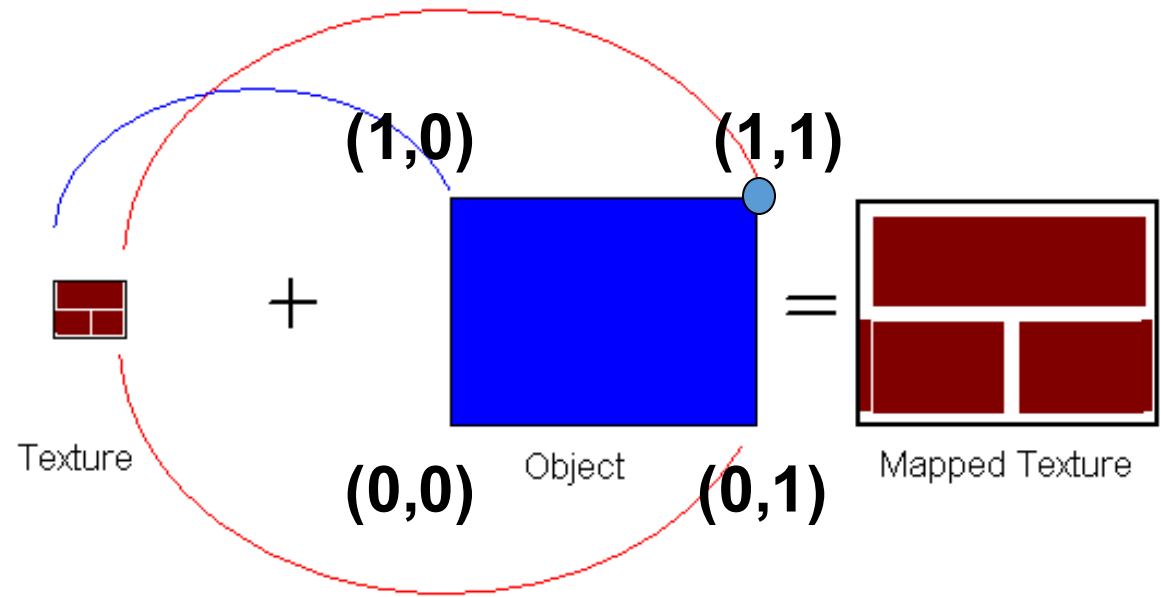
- Si s ou t est hors de $[0 \dots 1]$?
- Plusieurs choix
 - Utiliser la partie fractionnaire seulement
 - La répétition
 - Couper chaque composant à $[0 \dots 1]$
 - Réutiliser les valeurs sur la frontière



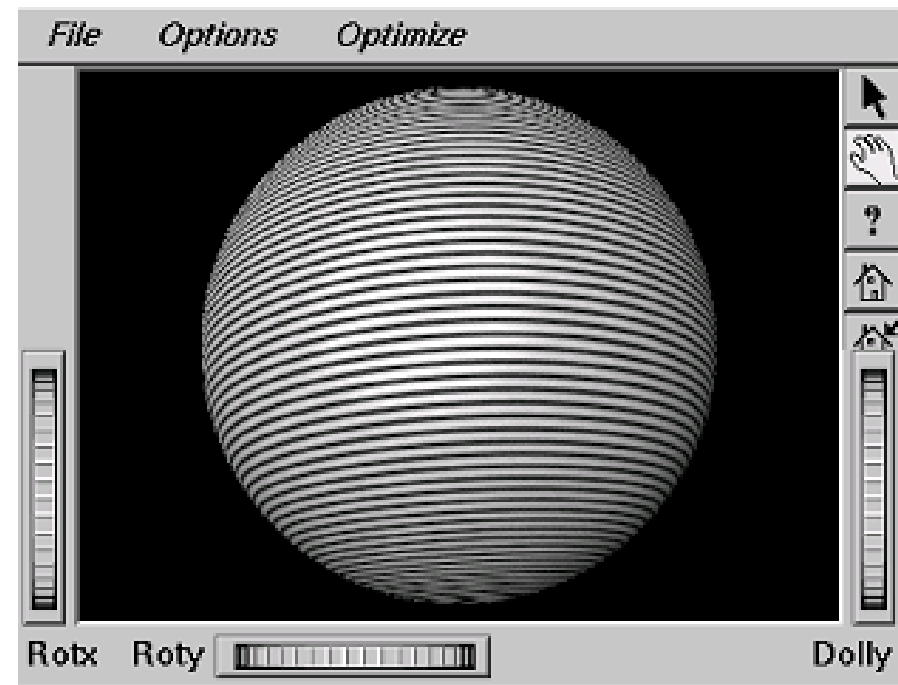
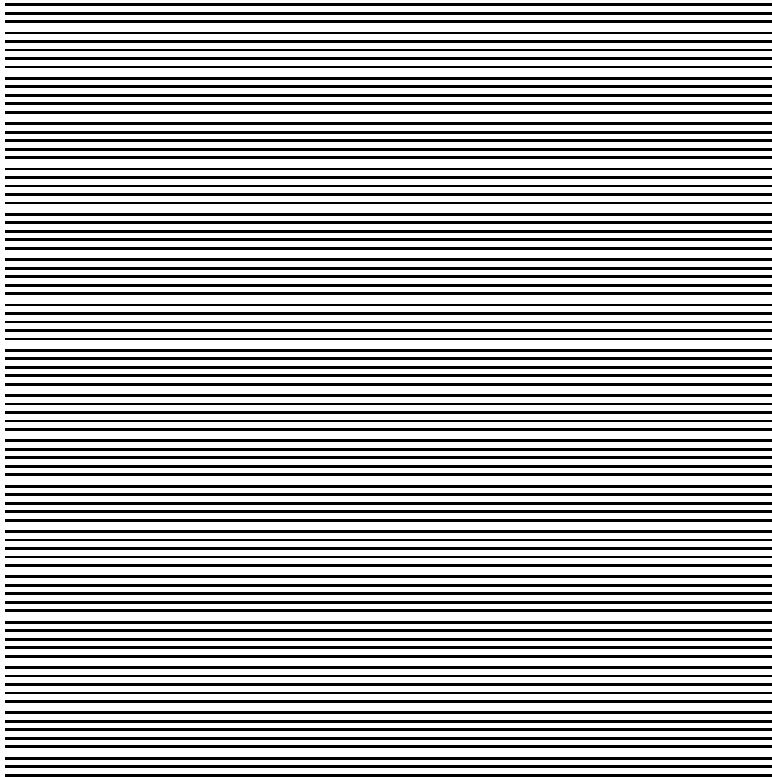
DANS THREE.JS

```
var texture = THREE.ImageUtils.loadTexture(  
    "textures/water.jpg" );  
texture.wrapS = THREE.RepeatWrapping;  
texture.wrapT = THREE.ClampToEdgeWrapping;  
texture.repeat.set( 4, 4 );
```

TILED TEXTURE MAP



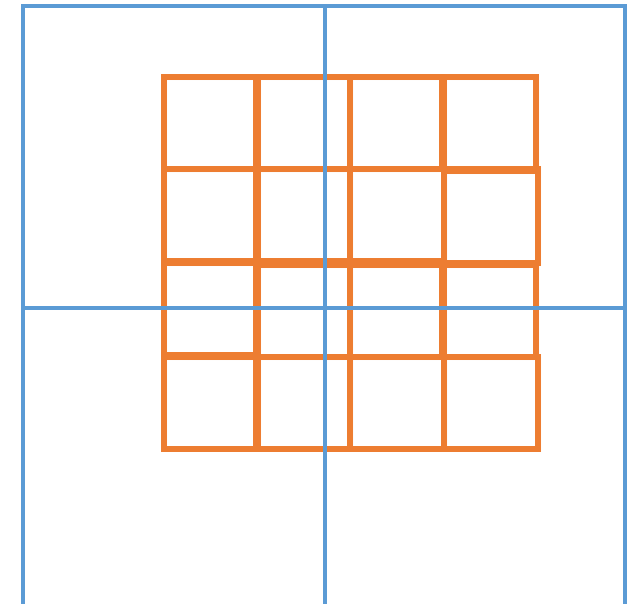
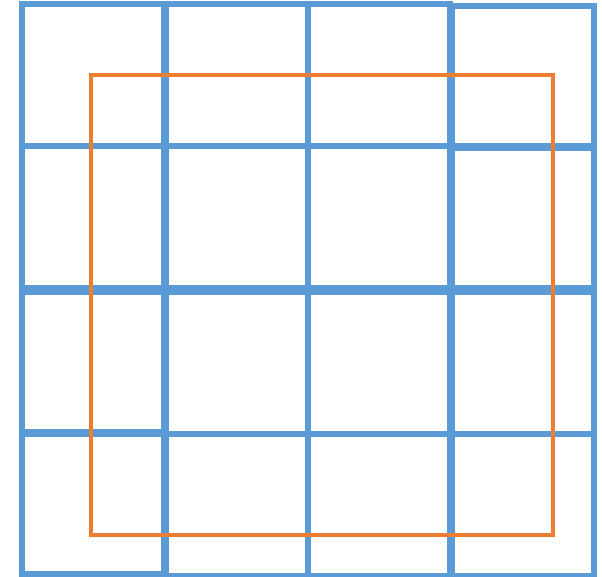
LA RECONSTRUCTION



(image courtesy of Kiriakos Kutulakos, U Rochester)

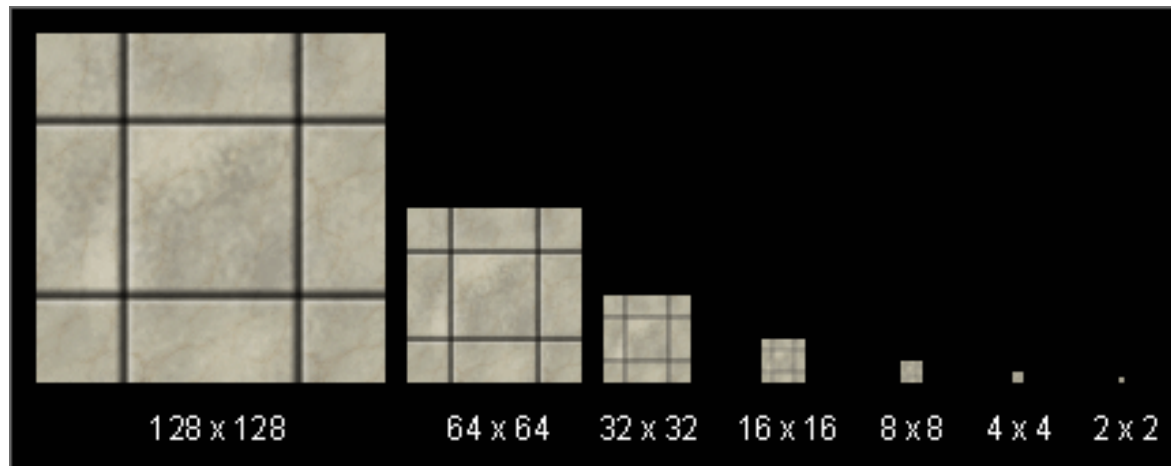
LA RECONSTRUCTION

- Comment faire avec:
 - Les **pixels** qui sont beaucoup plus grands que les **texels**?
 - La minification
 - Les **pixels** qui sont beaucoup plus petits que les **texels**?
 - La magnification

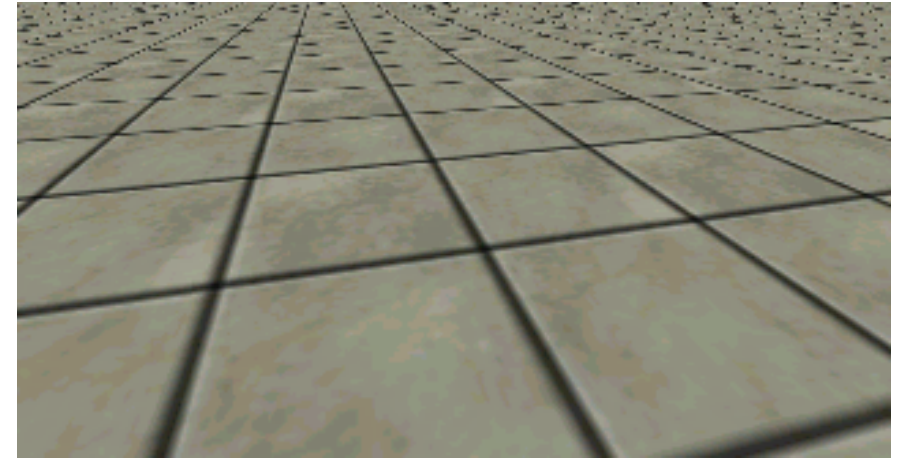
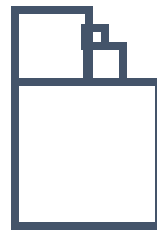


MIPMAPPING

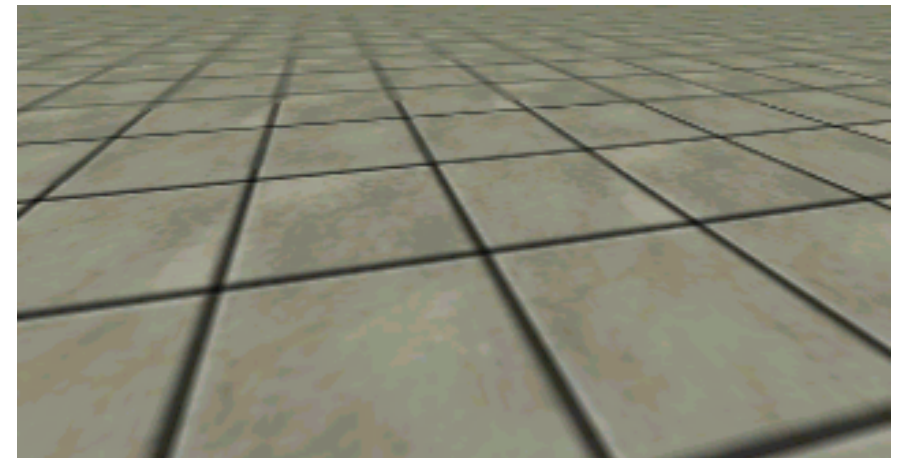
Utiliser “une pyramide d’image” pour
précalculer les versions moyennes d’image



Stocker la pyramide entière
comme un seul bloc de
mémoire



Sans MIP-mapping

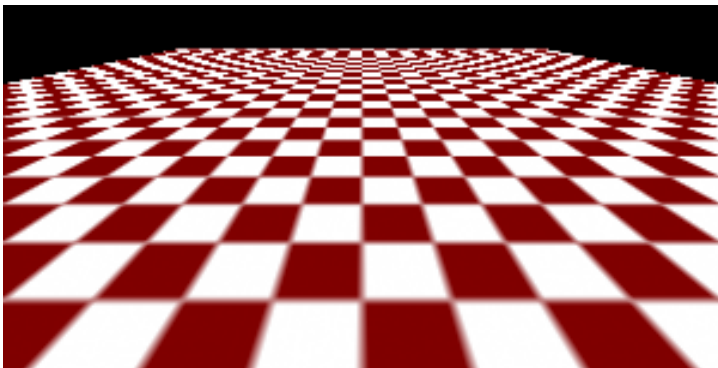


Avec MIP-mapping

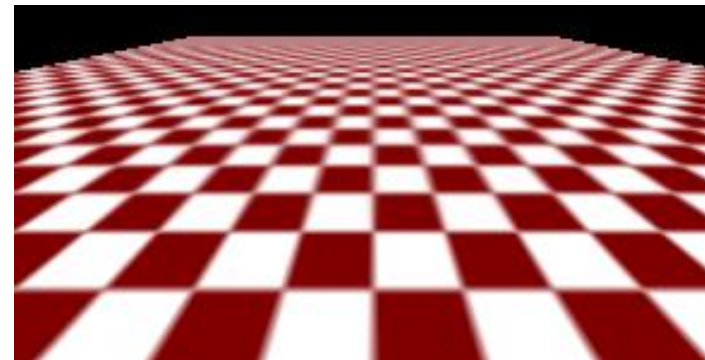
MIPMAP

- **multum in parvo** – beaucoup de choses dans un petit espace
 - Spécifier plusieurs textures préfiltrées à différentes résolutions
 - On a besoin plus de stockage
 - On peut éviter le scintillement
- `texture.generateMipmaps = true`
 - Construire une famille de textures de la taille d'origine à 1x1 (automatiquement)
- `texture.mipmaps [...]`

sans



avec



LE STOCKAGE DE MIPMAP

- Seulement $1/3$ plus d'espace est nécessaire
 - $4/3$ au total



COMMENT INTERPOLER U, V ?

Rappel

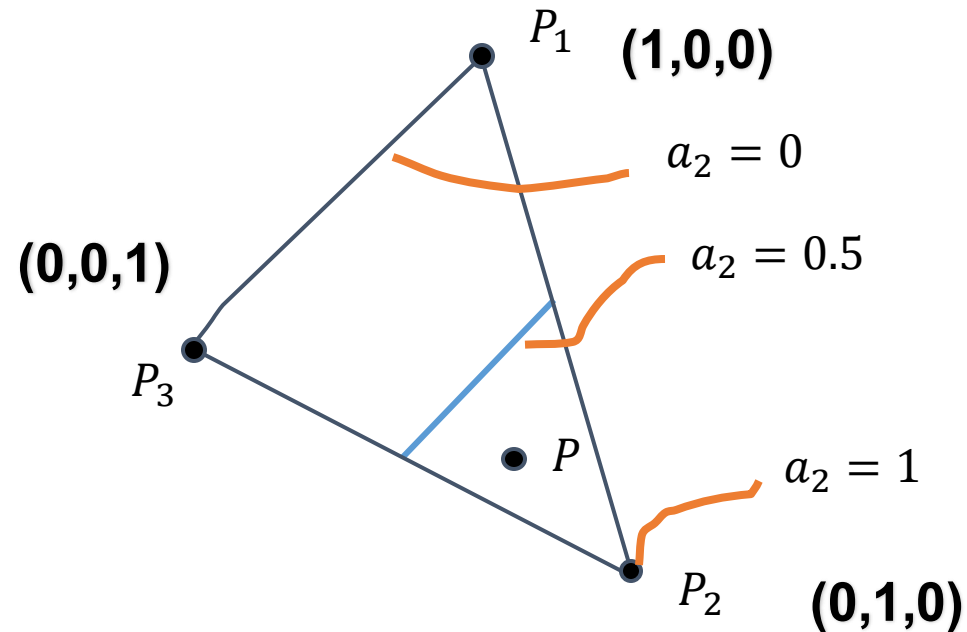
CHAQUE POINT À L'INTÉRIEUR DU TRIANGLE EST

- Une combinaison pondérée (affine) des sommets

$$P = a_1 \cdot P_1 + a_2 \cdot P_2 + a_3 \cdot P_3$$

$$a_1 + a_2 + a_3 = 1$$
$$0 \leq a_1, a_2, a_3 \leq 1$$

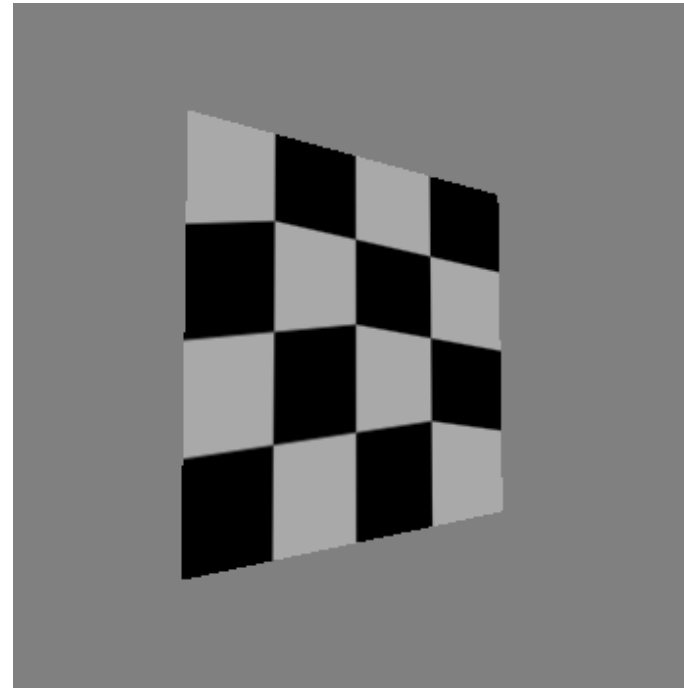
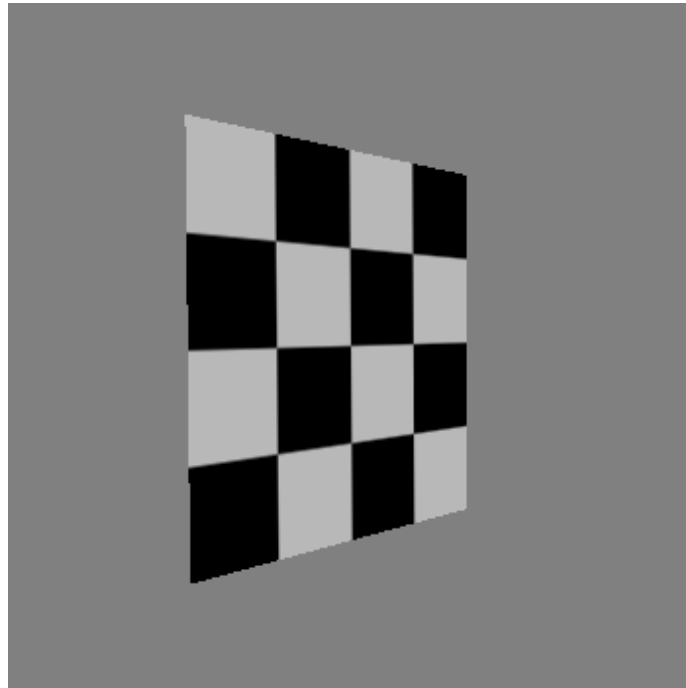
Sans cette équation
c'est le plan entier!



TEXTURE MAPPING

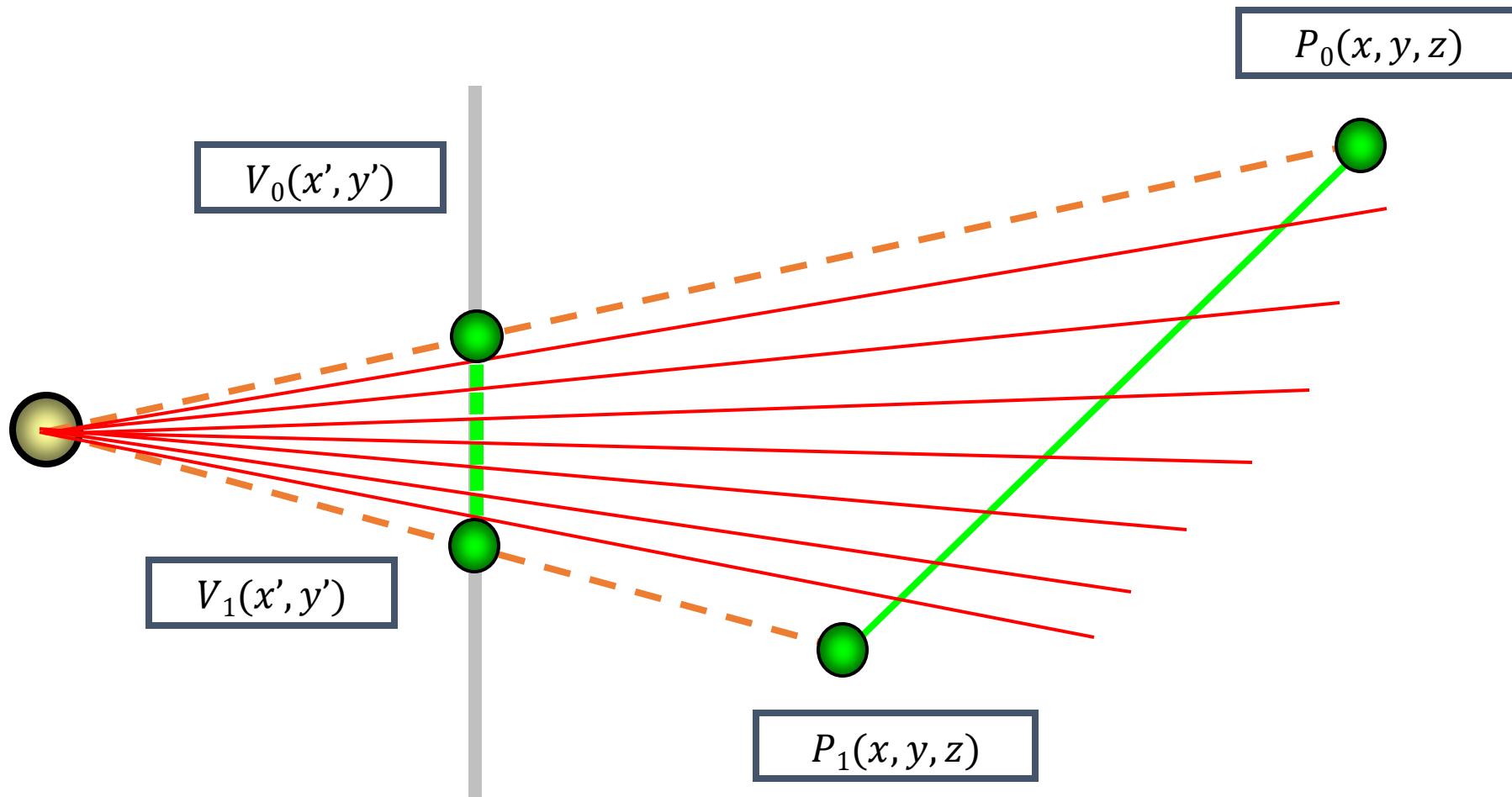
Après la projection perspective, l'interpolation linéaire dans l'espace de l'affichage est incorrecte

- Pour les textures, aussi bien que pour les couleurs, shading, etc.



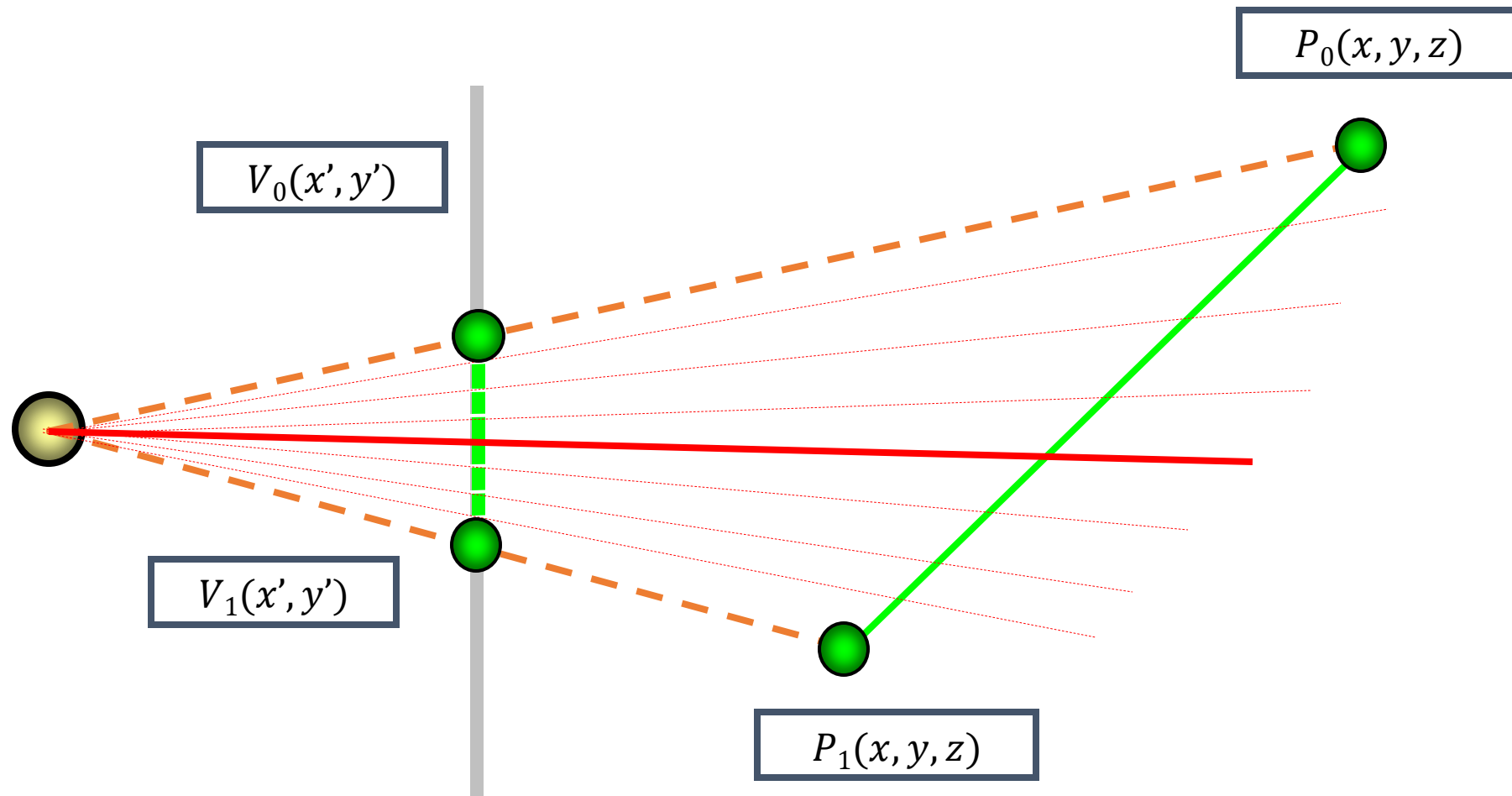
INTERPOLATION: L'ESPACE DE L'AFFICHAGE VS DU MONDE

- Normalement, on ignore le problème pour le *shading*, mais pour les textures les artefacts sont évidents



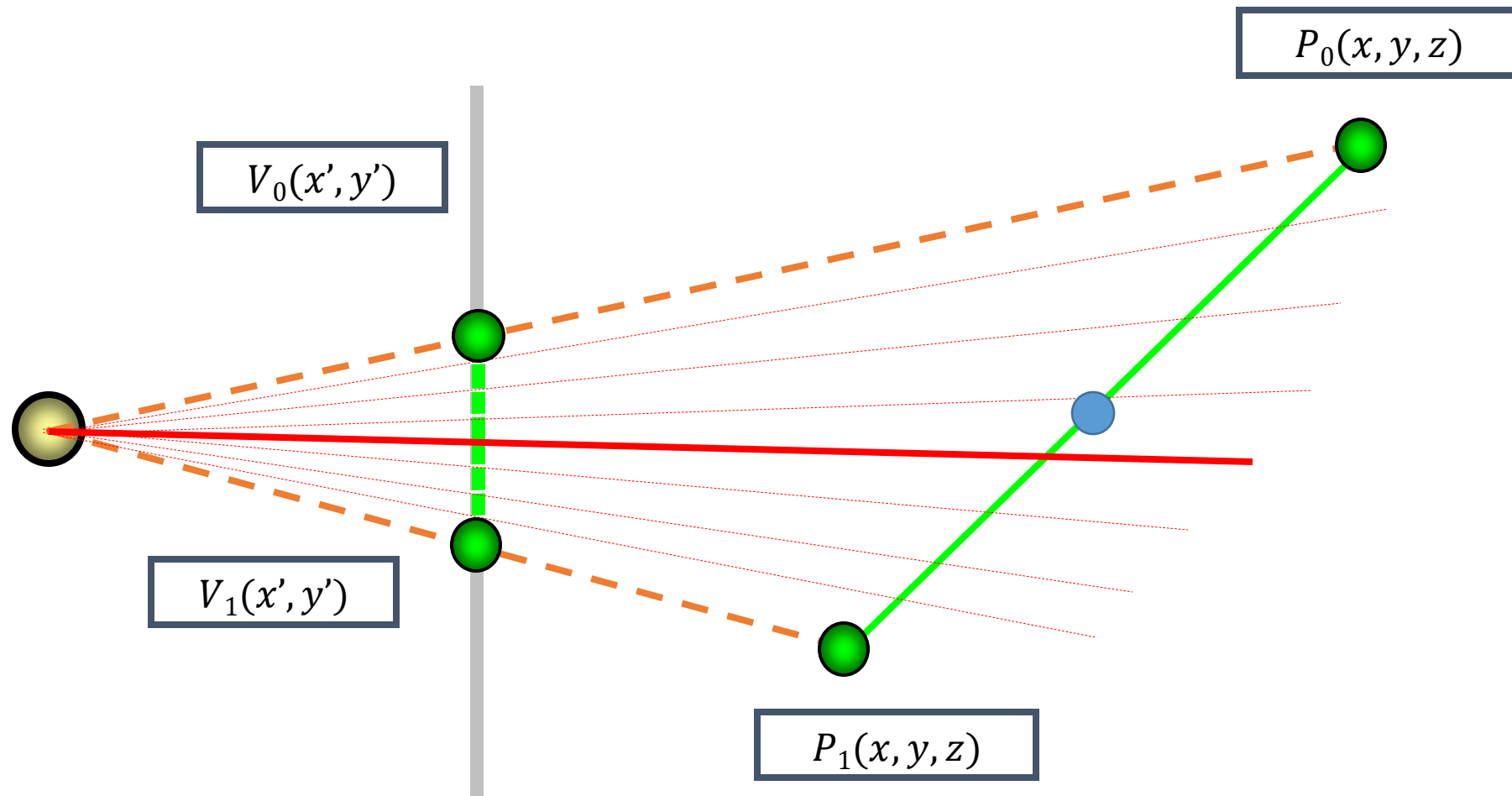
INTERPOLATION: L'ESPACE DE L'AFFICHAGE VS DU MONDE

- Normalement, on ignore le problème pour le *shading*, mais pour les textures les artefacts sont évidents



INTERPOLATION: L'ESPACE DE L'AFFICHAGE VS DU MONDE

- Normalement, on ignore le problème pour le *shading*, mais pour les textures les artefacts sont évidents



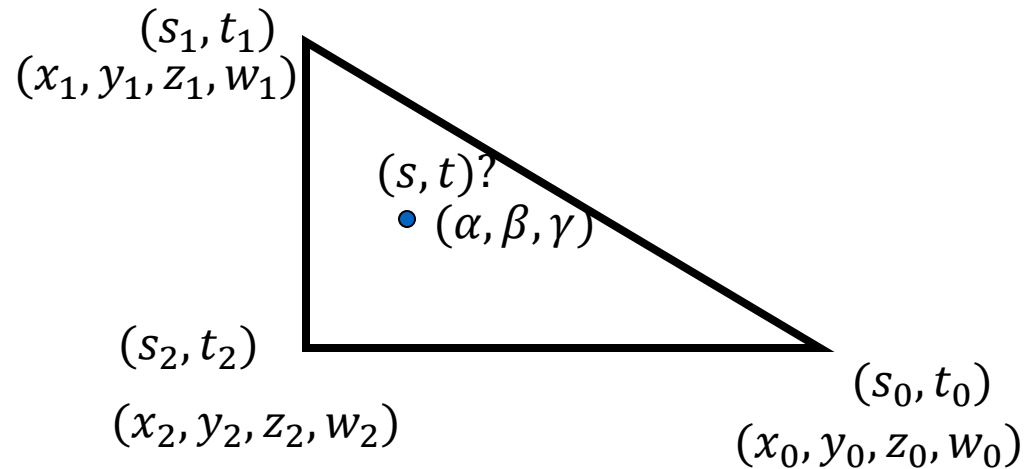
LA PERSPECTIVE – UN RAPPEL

$$T \begin{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad z_{NDC} = \frac{a \cdot z_{eye} + b}{z_{eye}} = a + \frac{b}{z_{eye}}$$

- Préserve l'ordre des profondeurs
 - Mais déforme les distances

L'INTERPOLATION APRÈS LA PERSPECTIVE

- α, β, γ : les coordonnées barycentriques (2D) du point P
- s_0, s_1, s_2 : les coordonnées dans l'espace de la texture
- w_0, w_1, w_2 : les coordonnées homogènes des vecteurs



$$s = \frac{\alpha \cdot s_0/w_0 + \beta \cdot s_1/w_1 + \gamma \cdot s_2/w_2}{\alpha/w_0 + \beta/w_1 + \gamma/w_2}$$

- La même chose pour t

La dérivation (les triangles similaires):

https://www.comp.nus.edu.sg/~lowkl/publications/lowk_persp_interp_techrep.pdf

LES AUTRES UTILISATIONS POUR LES TEXTURES

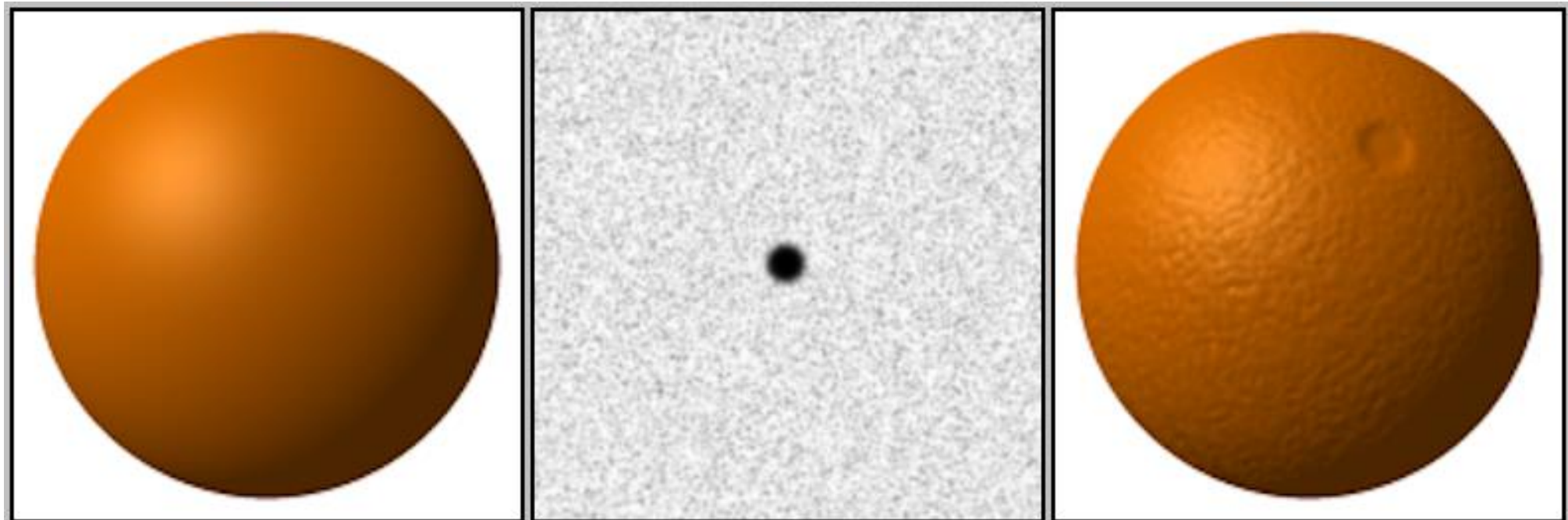
LES AUTRES UTILISATIONS POUR LES TEXTURES

- Normalement, elles donnent la couleur, mais...
- Elles peuvent être utilisées pour contrôler les autres propriétés de l'objet ou du matériel
 - La normale (*bump mapping + normal mapping*)
 - La couleur de la réflexion (*environment mapping*)

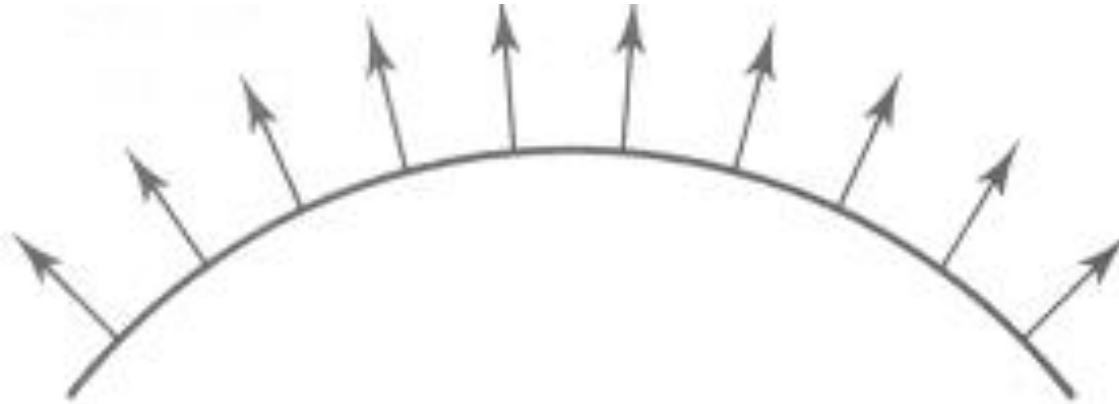


BUMP MAPPING ET NORMAL MAPPING

- La surface de l'objet est souvent rugueuse
 - On peut créer une géométrie plus complexe...
- Ou on peut la simuler en perturbant la normale seulement
 - Soit aléatoire
 - Soit spécifiée par une texture



BUMP MAPPING



$O(u)$

Original surface



$B(u)$

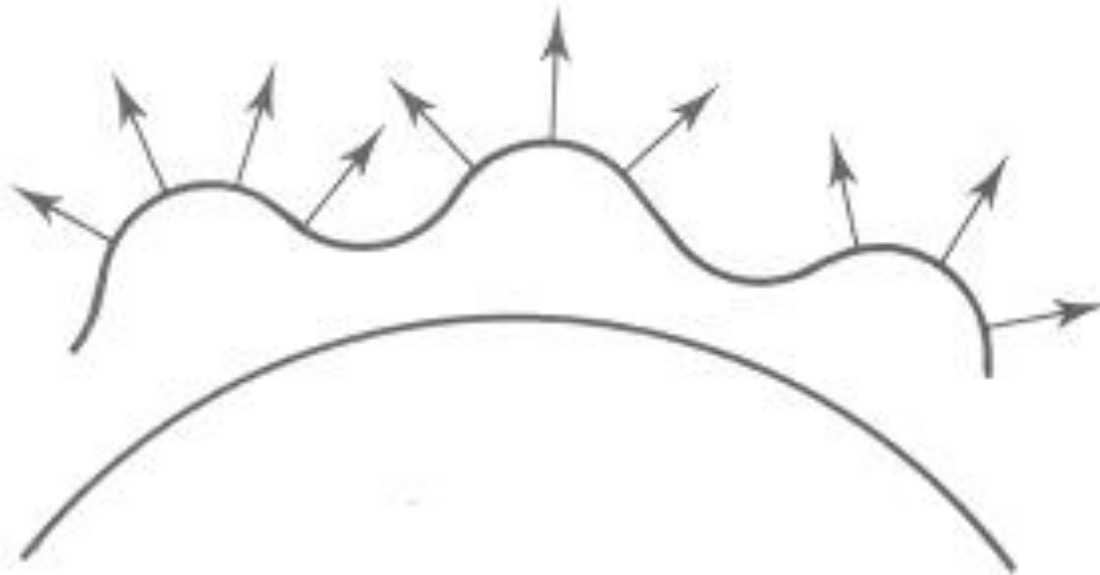
A bump map

BUMP MAPPING



$O'(u)$

Lengthening or shortening
 $O(u)$ using $B(u)$

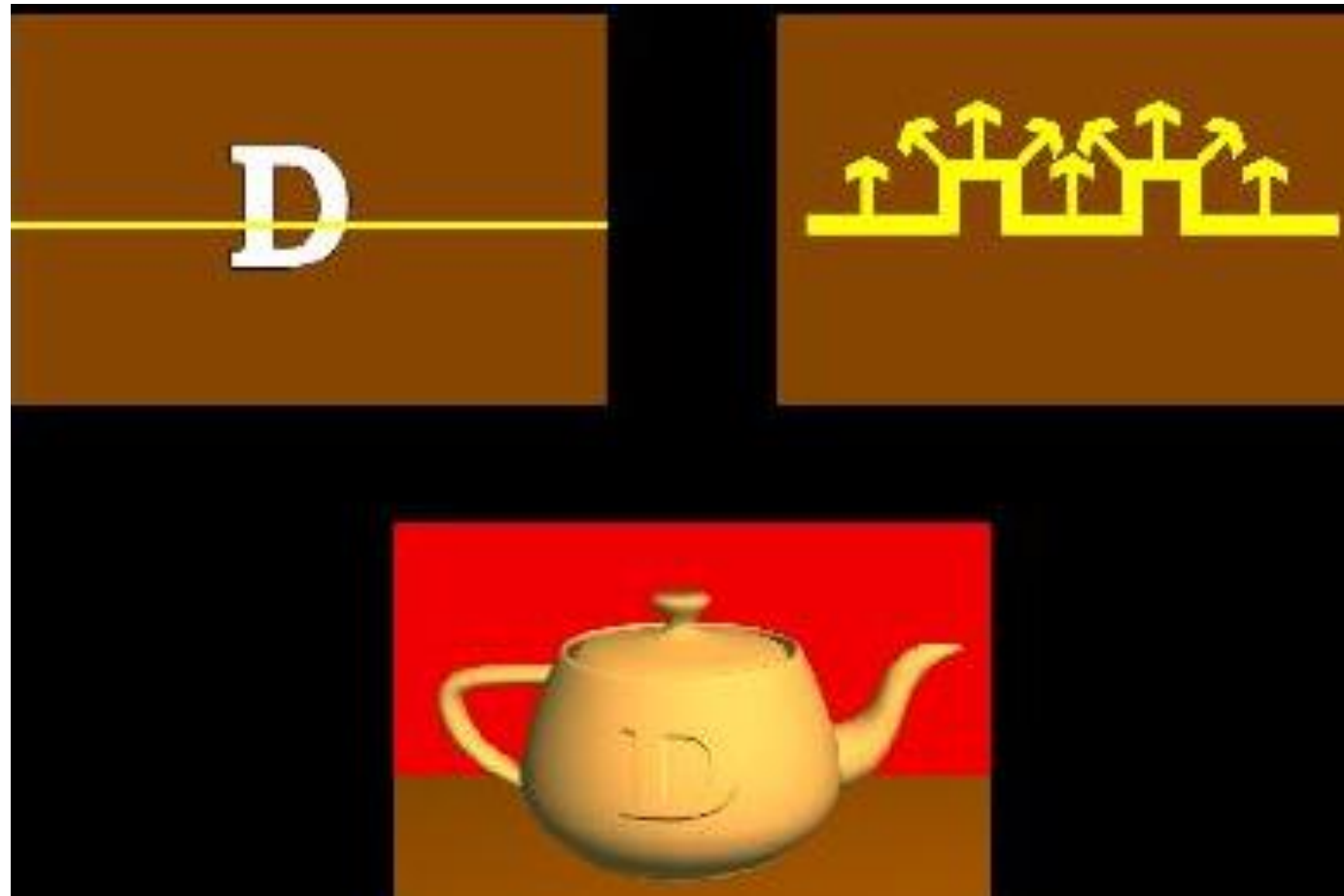


$N'(u)$

The vectors to the
'new' surface

LE RELIEF

- Aux transitions
 - Tourner la normale de ϑ ou $-\vartheta$

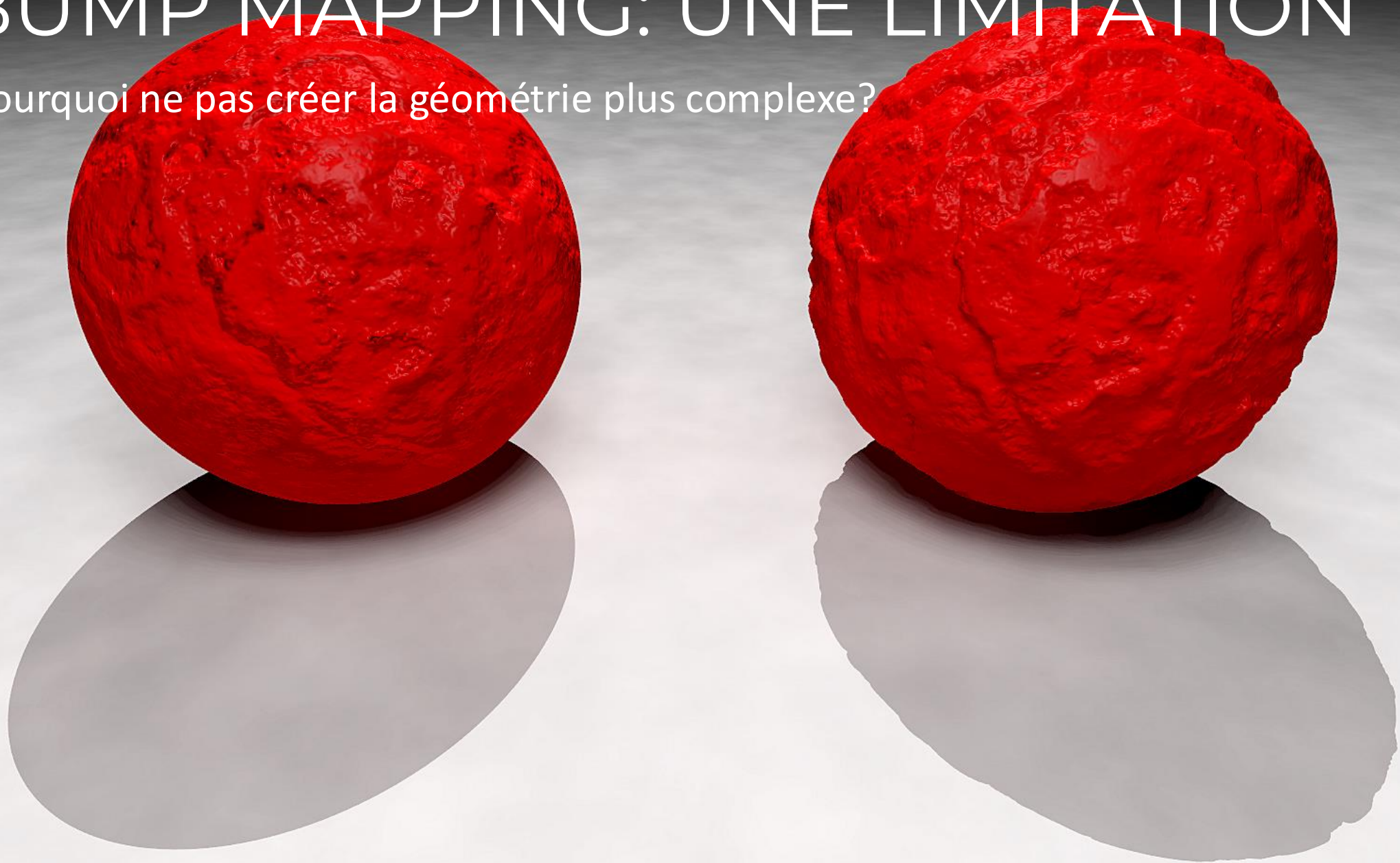


BUMP MAPPING: UNE LIMITATION



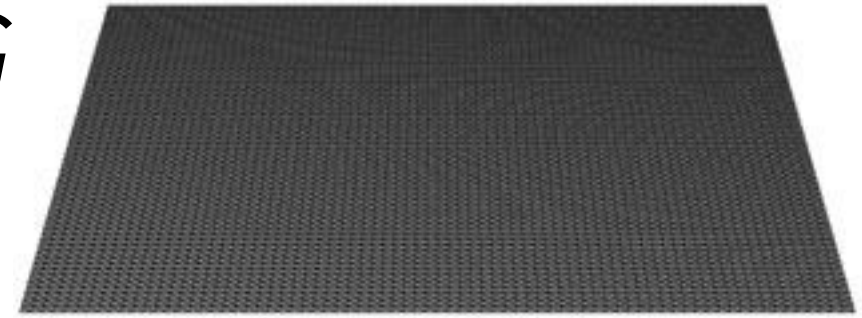
BUMP MAPPING: UNE LIMITATION

Pourquoi ne pas créer la géométrie plus complexe?



DISPLACEMENT MAPPING

- Les silhouettes sont fausses après *le bump mapping*
 - Et les ombres aussi!
- Changer la géométrie en temps réel
 - On a besoin de subdiviser la surface



ORIGINAL MESH



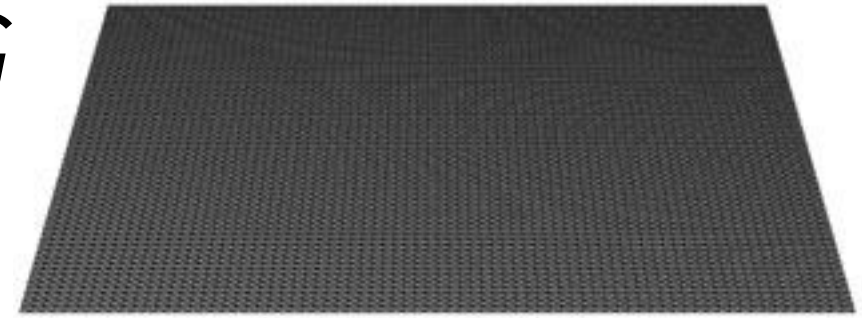
DISPLACEMENT MAP



MESH WITH DISPLACEMENT

DISPLACEMENT MAPPING

- Les silhouettes sont fausses après *le bump mapping*
 - Et les ombres aussi!
- Changer la géométrie en temps réel
 - On a besoin de subdiviser la surface
- Peut-on subdiviser la surface avant le pipeline?



ORIGINAL MESH



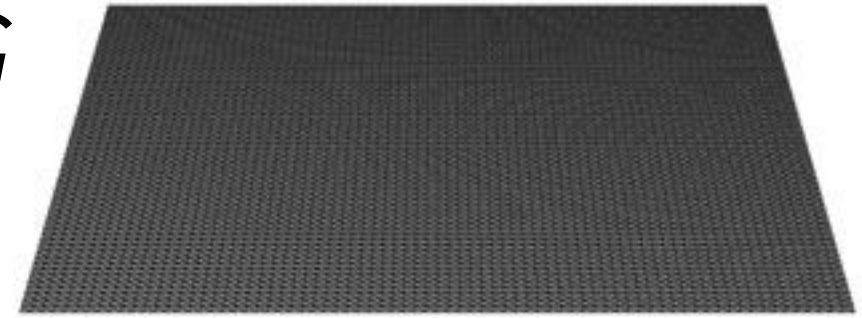
DISPLACEMENT MAP



MESH WITH DISPLACEMENT

DISPLACEMENT MAPPING

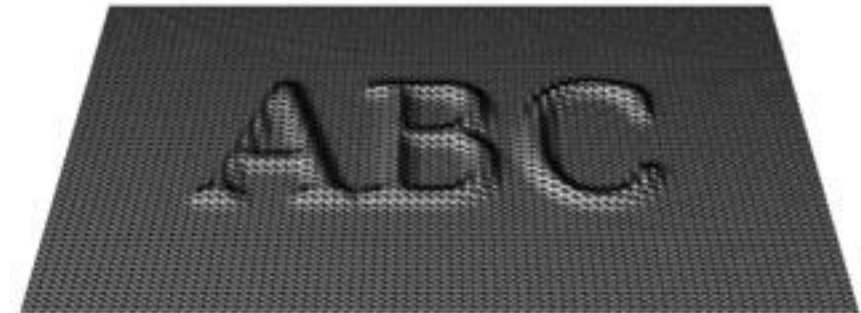
- Les silhouettes sont fausses après *le bump mapping*
 - Et les ombres aussi!
- Changer la géométrie en temps réel
 - On a besoin de subdiviser la surface
- Peut-on subdiviser la surface avant le pipeline?
 - Beaucoup de mémoire!
- Ou l'appliquer? Vertex shader? Fragment shader?



ORIGINAL MESH



DISPLACEMENT MAP



MESH WITH DISPLACEMENT

ENVIRONMENT MAPPING

Une méthode peu coûteuse pour obtenir un effet de réflexion complexe

- Si toutes les lumières sont loin, la réflexion dépend seulement de la direction
- Générer une image de l'environnement
- Mapper à l'objet, utiliser pour les réflexions



ENVIRONMENT MAPPING

The background image is a close-up of a Terminator robot from the movie Terminator 2. The robot's head and upper torso are visible, showing its intricate mechanical structure and highly reflective, metallic skin. The lighting is dramatic, with strong highlights and deep shadows, emphasizing the robot's form. The background is dark and out of focus, suggesting an industrial or urban setting.

- On l'utilise pour modéliser l'objet qui reflète
- Les réflexions réelles sont précalculées pour un objet simple
- Puis, la direction de réflexion pour un objet complexe indique la couleur
- Différentes formes de proxy: une sphère (rarement), un cube

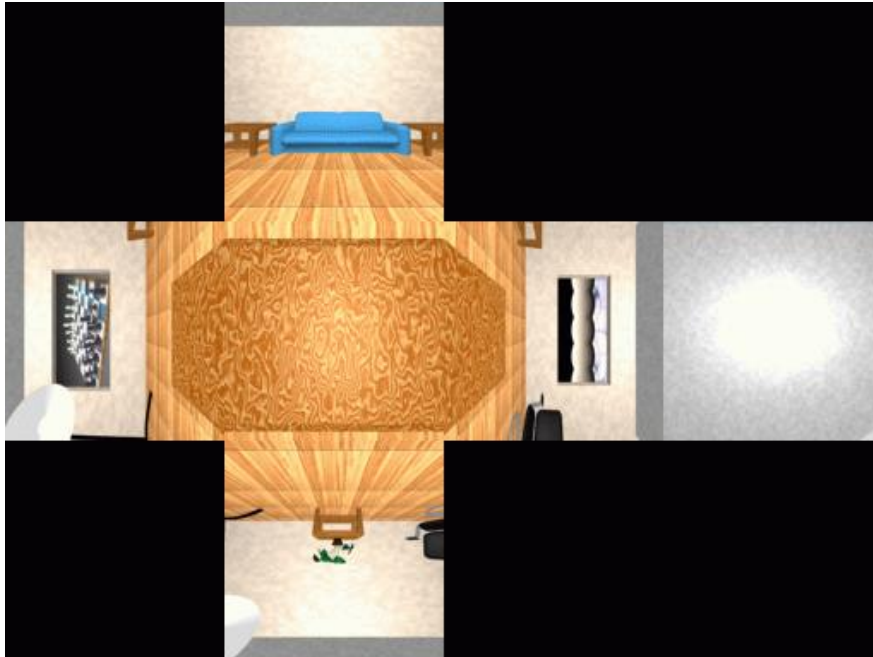
SPHERE MAPPING

- Calculer les réflexions pour la sphère
- Remplacer la sphère par l'objet, utiliser les nouvelles normales

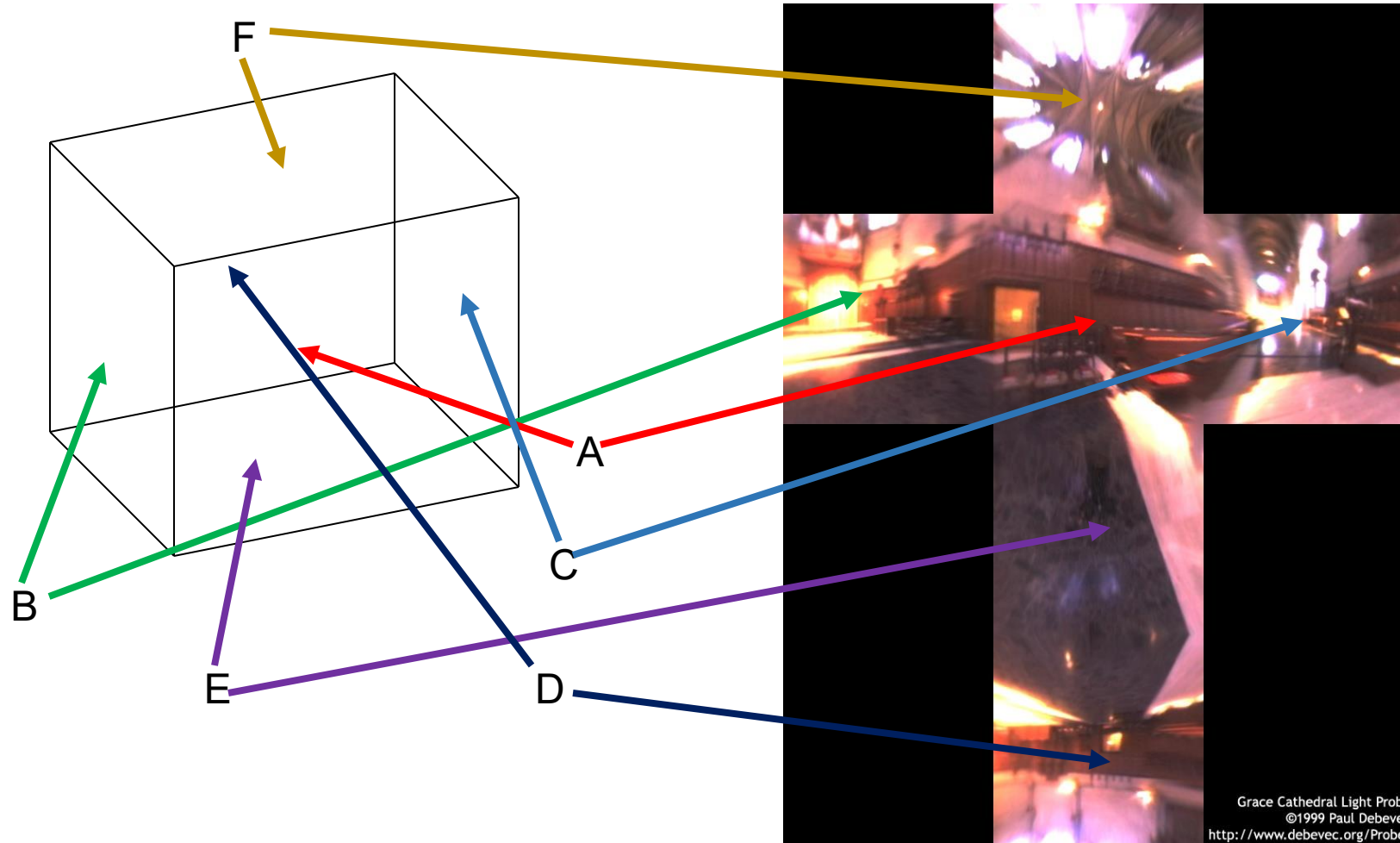


CUBE MAPPING

- 6 textures planaires, les faces du cube
 - Pointe la caméra depuis l'origine dans 6 directions



CUBE MAPPING



CUBE MAPPING

- La direction du vecteur r de la réflexion indique la face du cube
 - La coordonnée avec la plus grande valeur absolue
 - e.g., le vecteur $(-0.2, 0.5, -0.84)$ indique la face $-Z$
 - Les autres deux coordonnées définissent les coordonnées uv
- Il est difficile d'interpoler entre les faces

CUBE MAPPING

Comment
calculer?

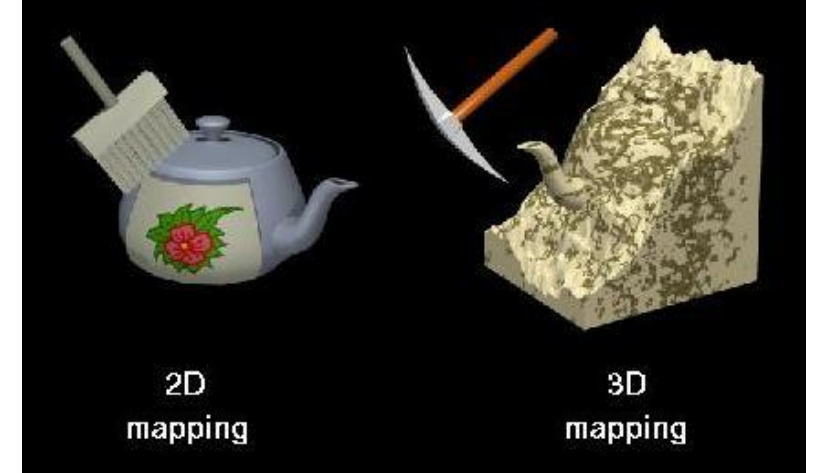
- La direction du vecteur r de la réflexion indique la face du cube
 - La coordonnée avec la plus grande valeur
 - e.g., le vecteur $(-0.2, 0.5, -0.84)$ indique la face $-Z$
 - Les autres deux coordonnées définissent les coordonnées uv
- Il est difficile d'interpoler entre les faces

ENVIRONMENT MAPS (EM)

- En théorie, chaque objet doit avoir une EM différente
- En théorie, chaque fois quelque chose bouge, on doit recalculer la EM
- “Vous serez surpris de ce que vous pourrez faire avec”

TEXTURE VOLUMÉTRIQUE

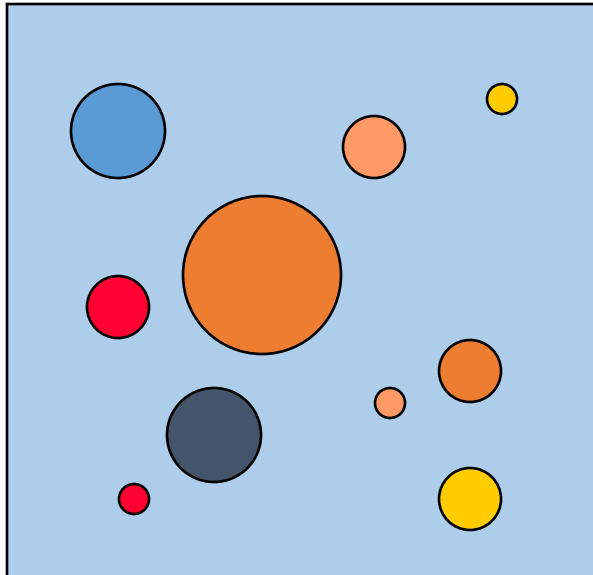
- Définir la texture dans l'espace 3D
- La fonction de la texture peut être stockée ou **procédurale**
- La couleur est définie par la position 3D
- e.g., ShaderToy
- Le calcul est rapide, mais l'accès à la mémoire ne l'est pas



EFFETS DE TEXTURE PROCÉDURALE: BOMBING

Larguer des 'bombes' par hasard (stocker l'historique dans un tableau)

- Pour chaque point P , rechercher dans le tableau
 - Si P est à l'intérieur d'une bombe, on utilise sa couleur

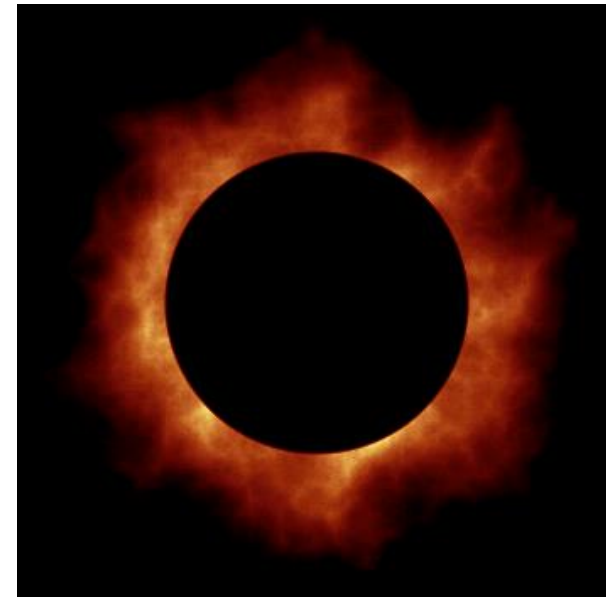
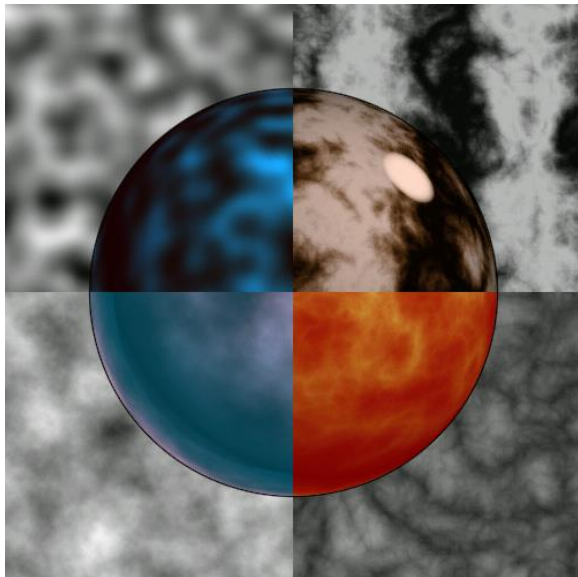


Larguer des 'bombes' par hasard (stocker l'historique dans un tableau)

- Pour chaque point P , rechercher dans le tableau
 - Si P est à l'intérieur d'une bombes, on utilise sa couleur

BRUIT DE PERLIN: TEXTURES PROCÉDURALES

- Quelques bonnes explications:
 - <http://www.noisemachine.com/talk1>
 - http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
 - <http://www.robo-murito.net/code/perlin-noise-math-faq.html>



<http://mrl.nyu.edu/~perlin/planet/>

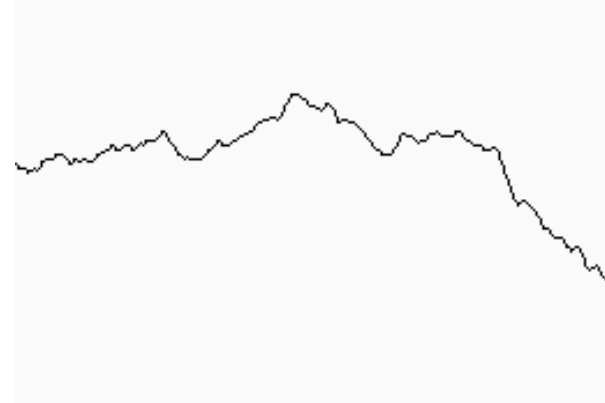
DEMOSCENE

- Une sous-culture de l'art numérique
- Créer une petite démo (e.g. <64k)
- Montrer les compétences en programmation
- Typiquement les démos dépendent *fortement* des algorithmes de bruit et des textures procédurales

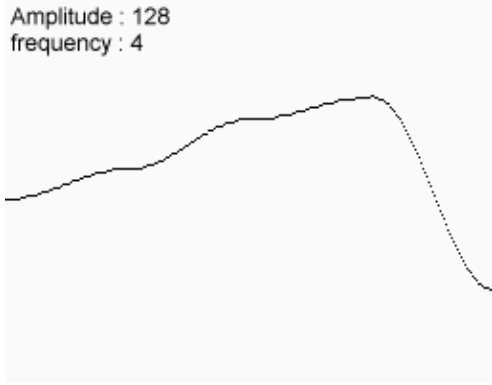
BRUIT DE PERLIN: LA TURBULENCE

- Ajouter plusieurs échelles

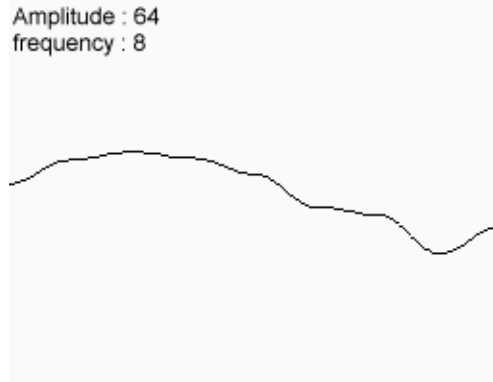
Sum of Noise Functions = (Perlin Noise)



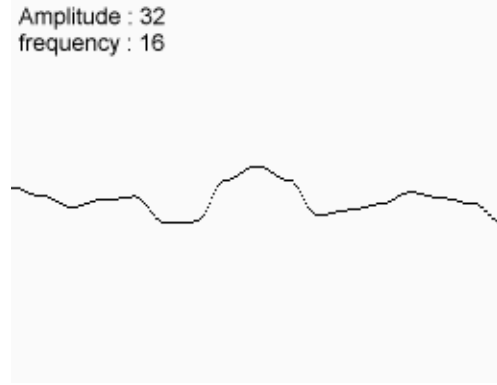
Amplitude : 128
frequency : 4



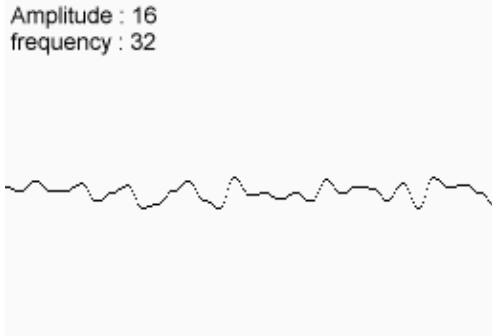
Amplitude : 64
frequency : 8



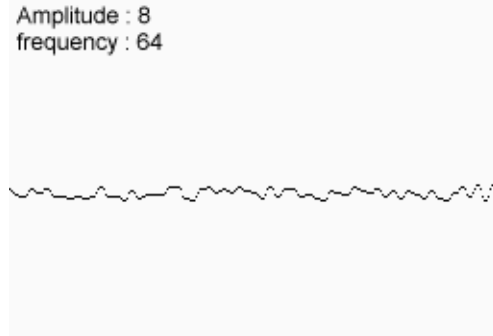
Amplitude : 32
frequency : 16



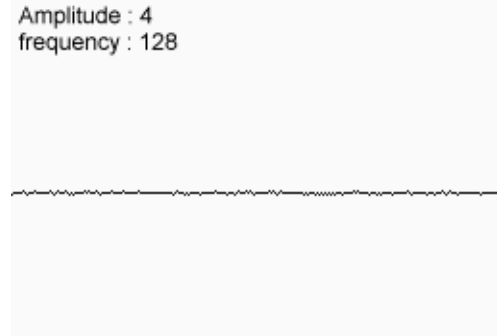
Amplitude : 16
frequency : 32



Amplitude : 8
frequency : 64

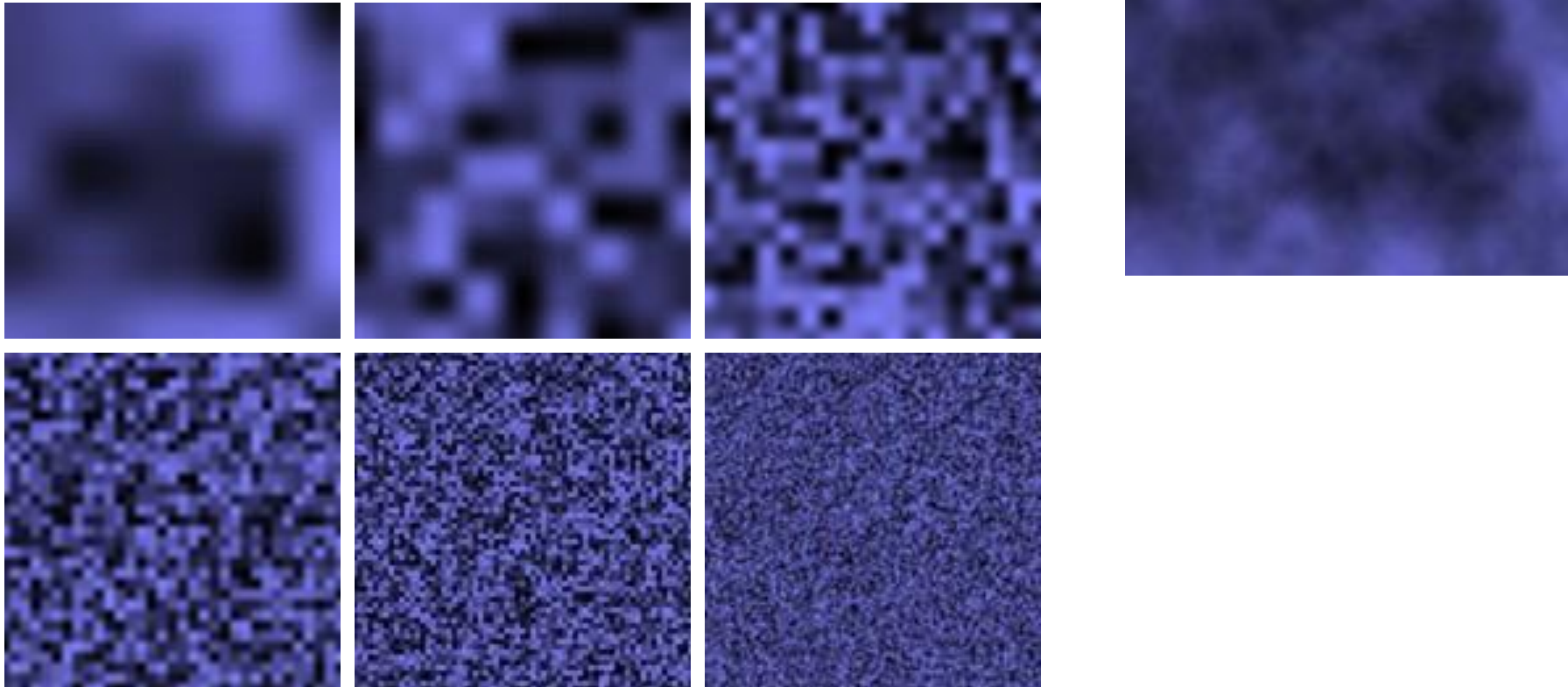


Amplitude : 4
frequency : 128

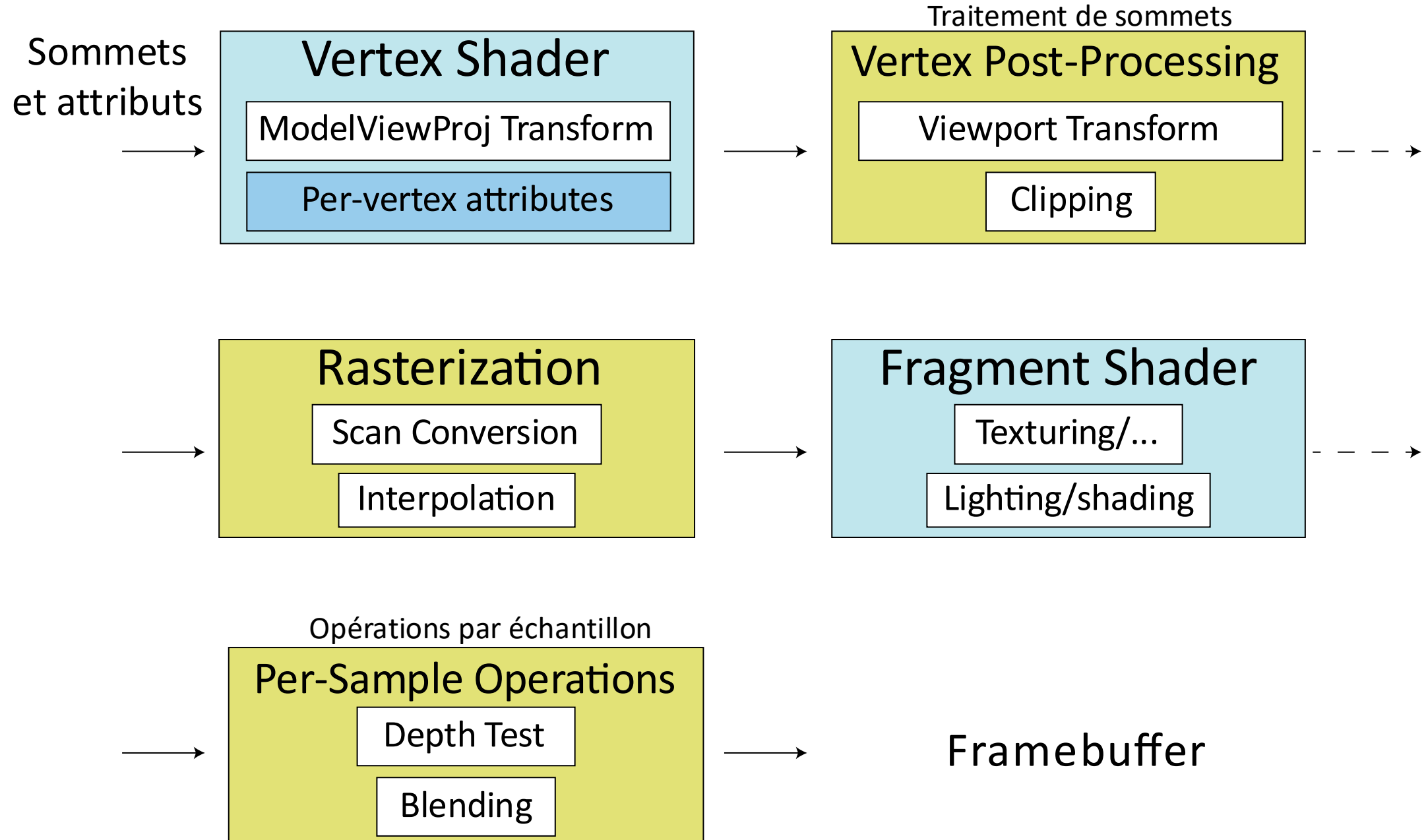


BRUIT DE PERLIN: LA TURBULENCE

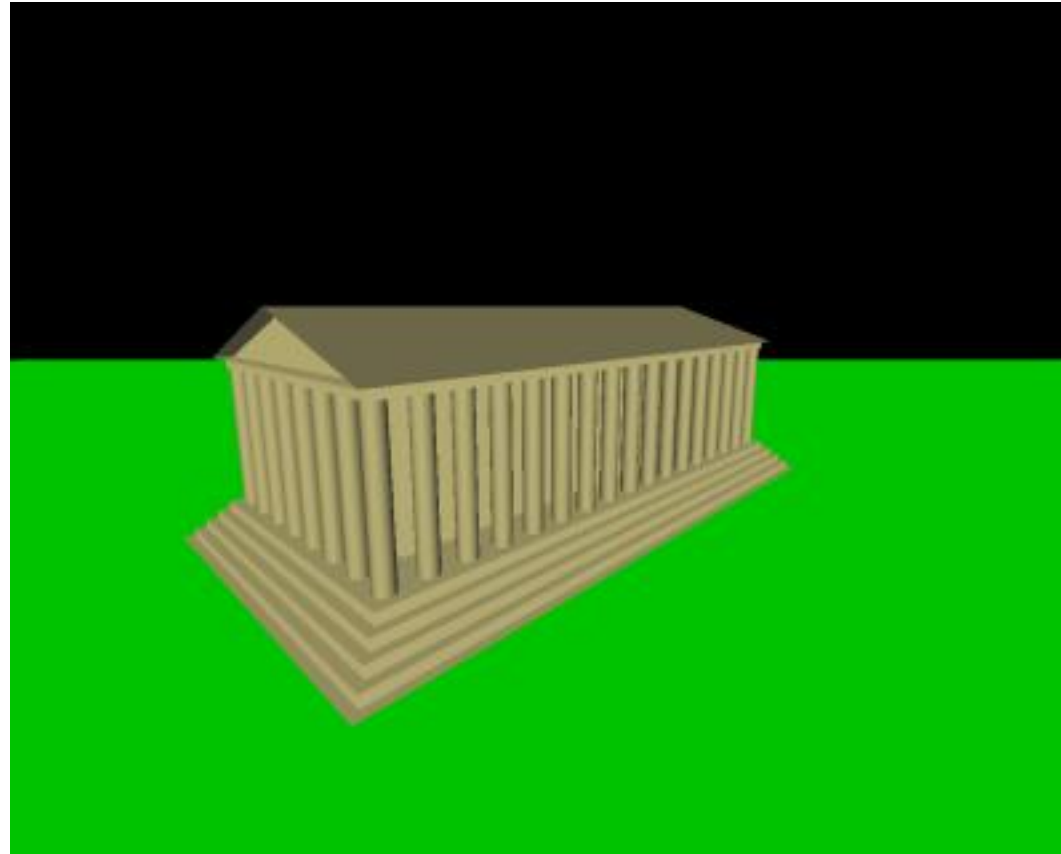
- Ajouter plusieurs échelles



PIPELINE: PLUS DE DÉTAILS



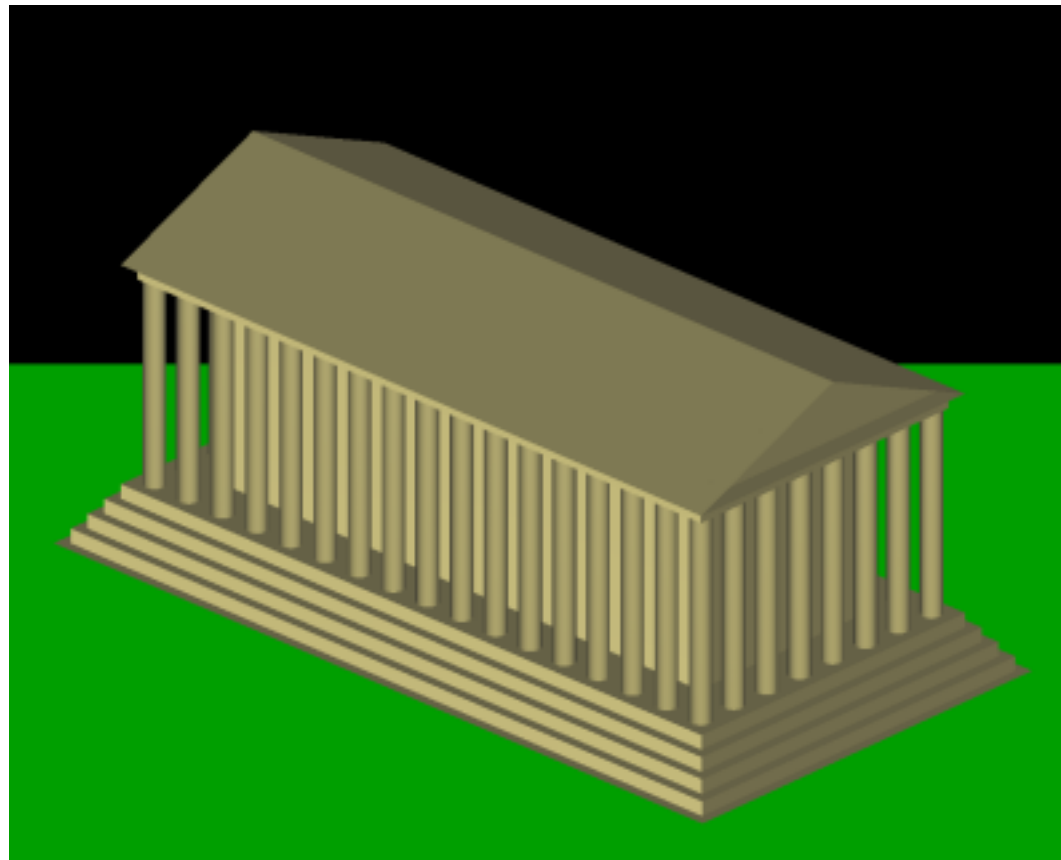
LES OMBRES



LES OMBRES

On a besoin d'au moins 2 passes de *shaders*:

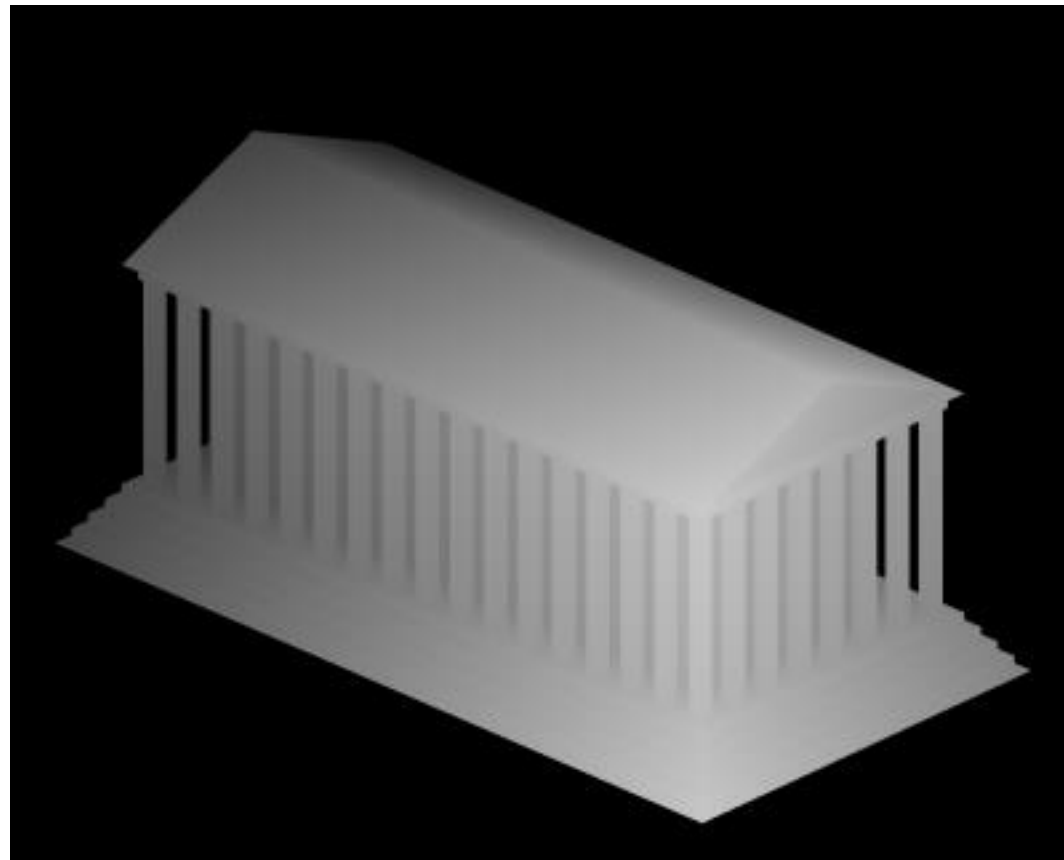
1. Rendre tout comme vu depuis **la lumière**



LES OMBRES

On a besoin d'au moins 2 passes de *shaders* :

1. Rendre tout comme vu depuis **la lumière**
Enregistrer la **profondeur** (tampon de profondeur, '*depth map*')



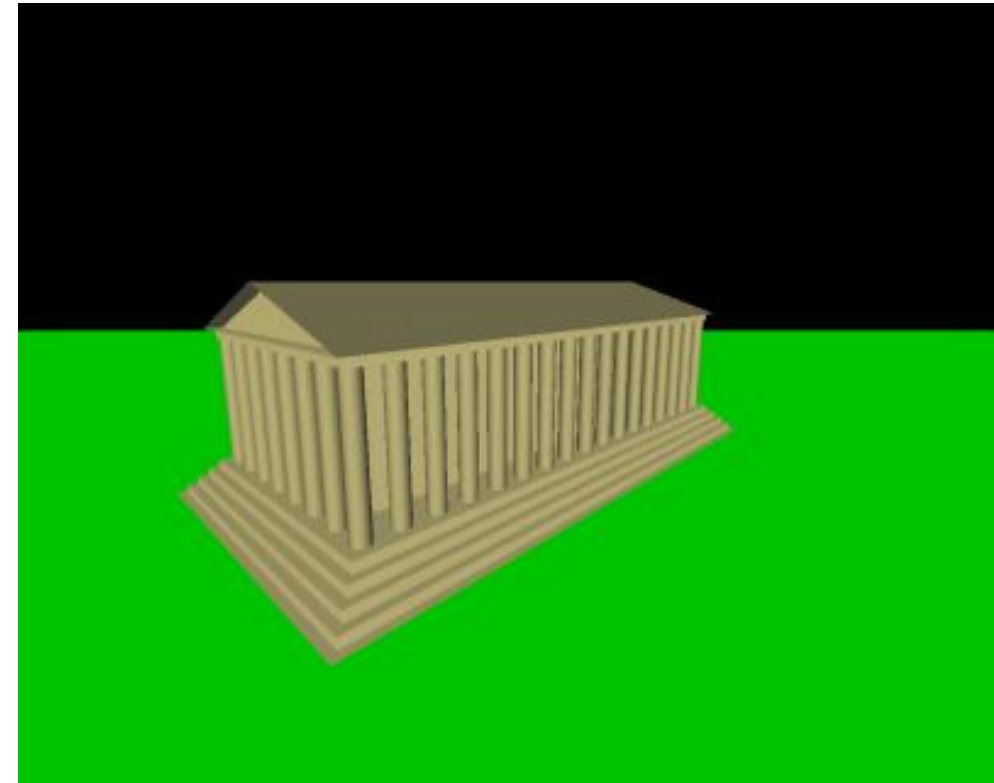
LES OMBRES (IDÉE)

On a besoin d'au moins 2 passes de *shaders* :

1. Rendre tout comme vu depuis **la lumière**
Enregistrer la **profondeur** (tampon de profondeur/ombre, '*shadow map*')
2. Maintenant, rendre tout depuis la **CAMÉRA**

Quand on calcule la couleur d'un fragment:

- Convertir les coordonnées dans l'espace de la lumière (x_l, y_l, z_l)
 - Prendre la profondeur $D(x_l, y_l)$
- Est-ce $z_l > D(x_l, y_l)$?
 - Oui: je suis dans l'ombre
 - Non: je suis éclairé



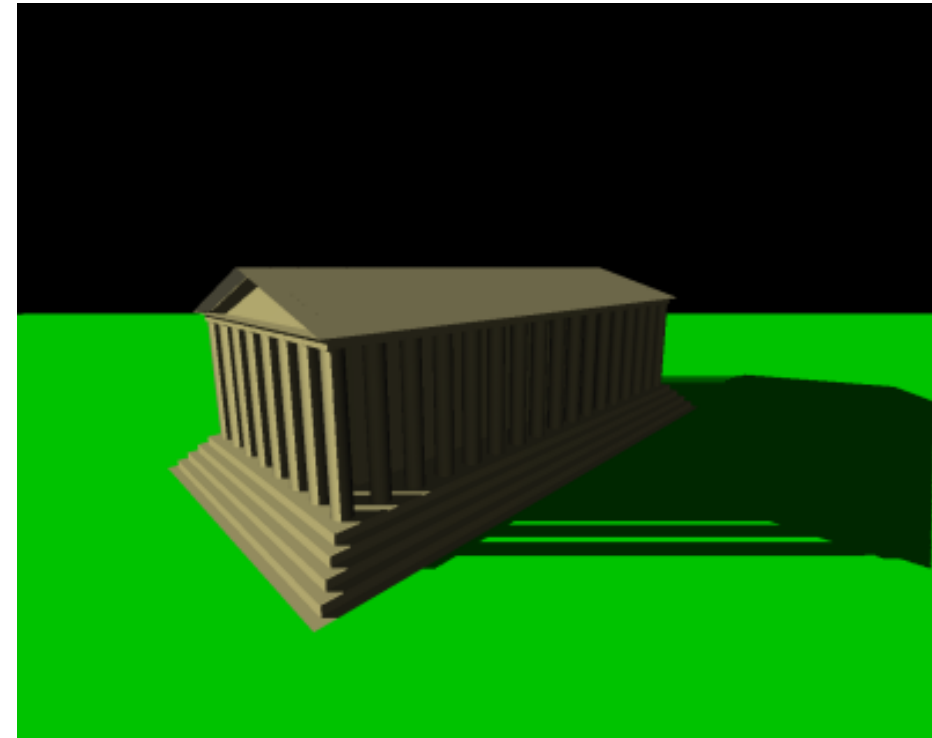
LES OMBRES (IDÉE)

On a besoin d'au moins 2 passes de *shaders* :

1. Rendre tout comme vu depuis **la lumière**
Enregistrer la **profondeur** (tampon de profondeur/ombre, '*shadow map*')
2. Maintenant, rendre tout depuis la **CAMÉRA**

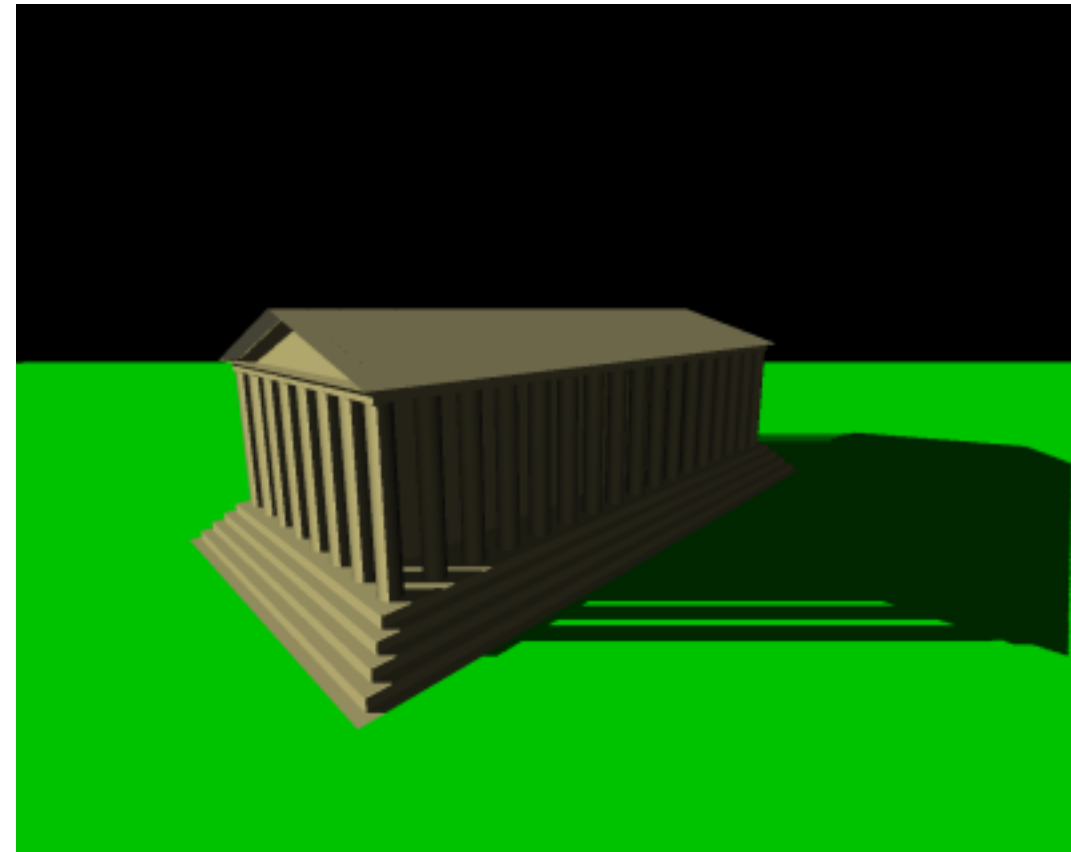
Quand on calcule la couleur d'un fragment:

- Convertir les coordonnées dans l'espace de la lumière (x_l, y_l, z_l)
 - Prendre la profondeur $D(x_l, y_l)$
- Est-ce $z_l > D(x_l, y_l)$?
 - Oui: je suis dans l'ombre
 - Non: je suis éclairé



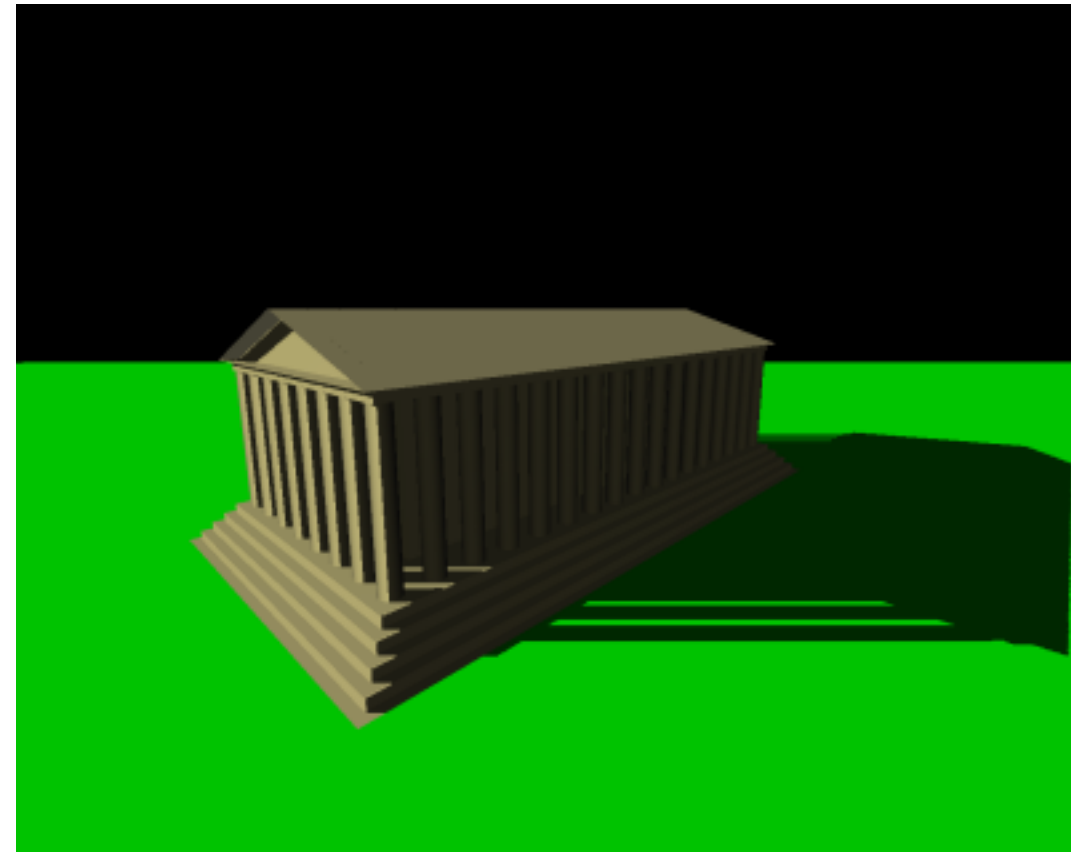
LES PROBLÈMES DE SHADOW MAPPING

- Bords d'ombre durs
 - Peut être résolu par plusieurs accès dans le shadow mapping



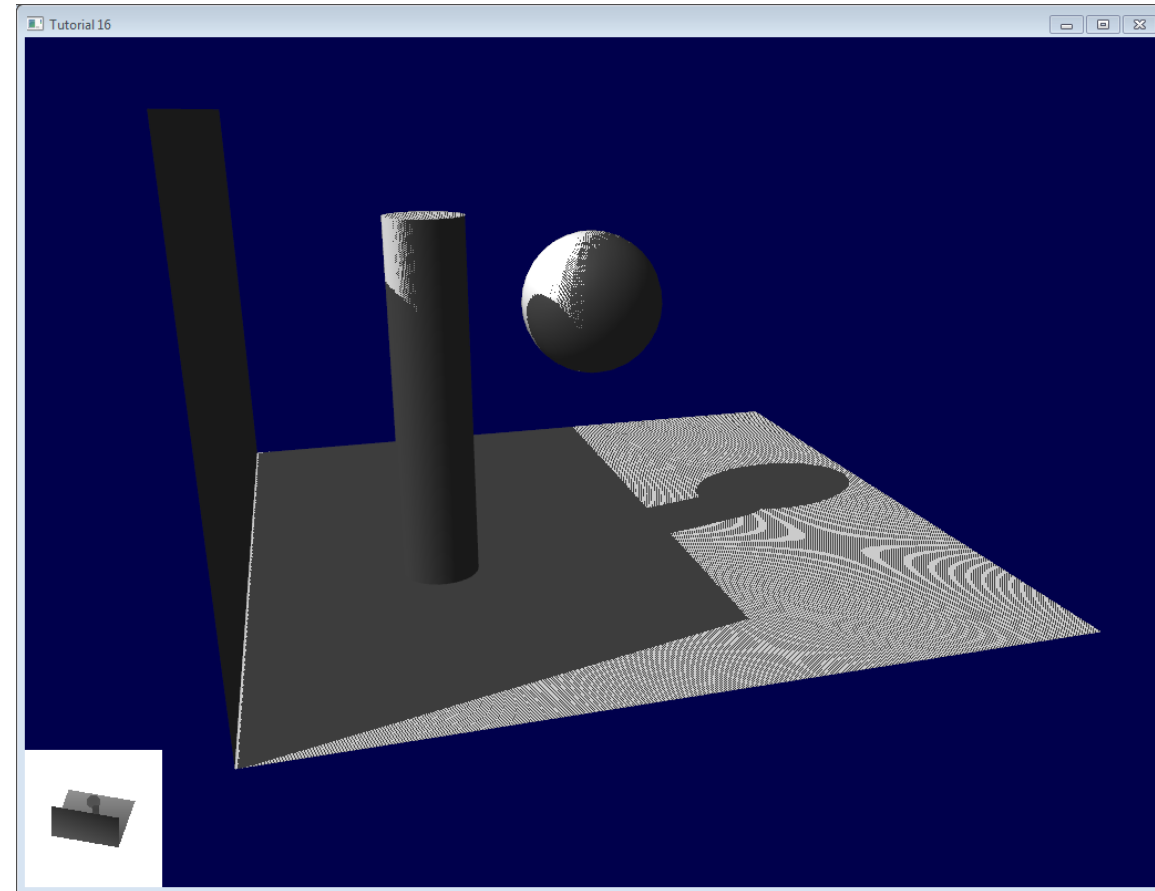
LES PROBLÈMES DE SHADOW MAPPING

- Bords d'ombre durs
 - Plusieurs recherches dans le *shadow mapping*
- L'aliasage d'ombre
 - Augmenter la résolution
- Acné/biais
 - Une profondeur per texel de *shadow map* + epsilon
- Plusieurs variations de *shadow mapping* essayent de résoudre ces problèmes



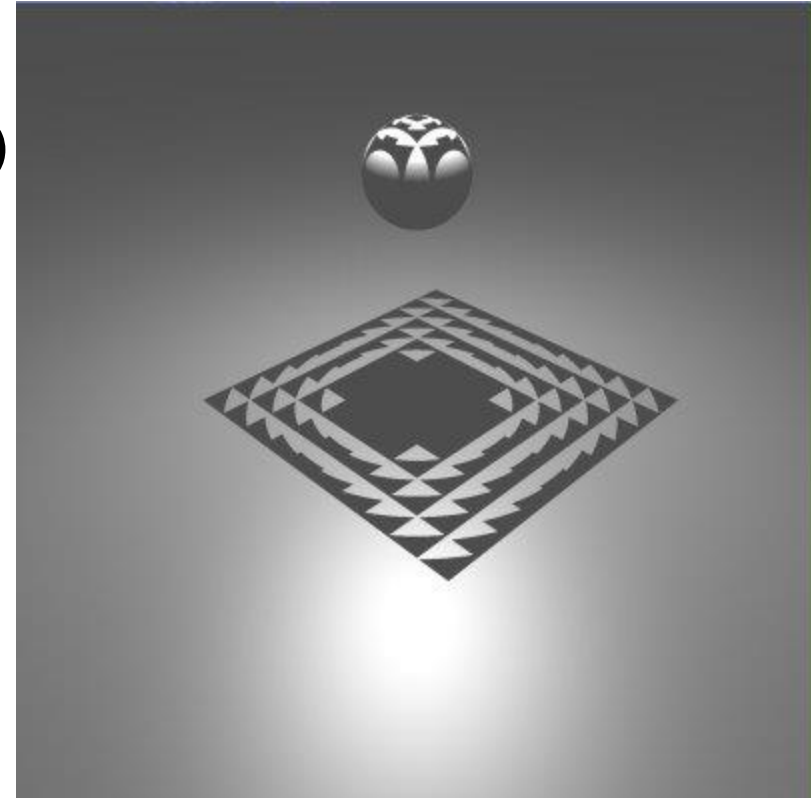
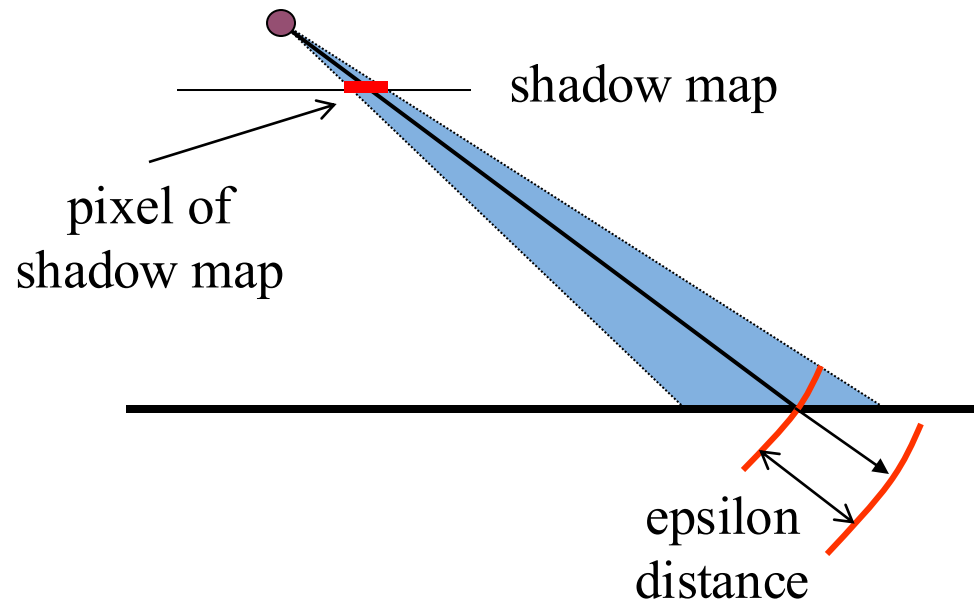
LES PROBLÈMES DE SHADOW MAPPING

- Bords d'ombre durs
 - Plusieurs recherches dans le *shadow mapping*
- L'aliasage d'ombre
 - Augmenter la résolution
- **Acné/biais**
 - Une profondeur per texel de *shadow map* + epsilon



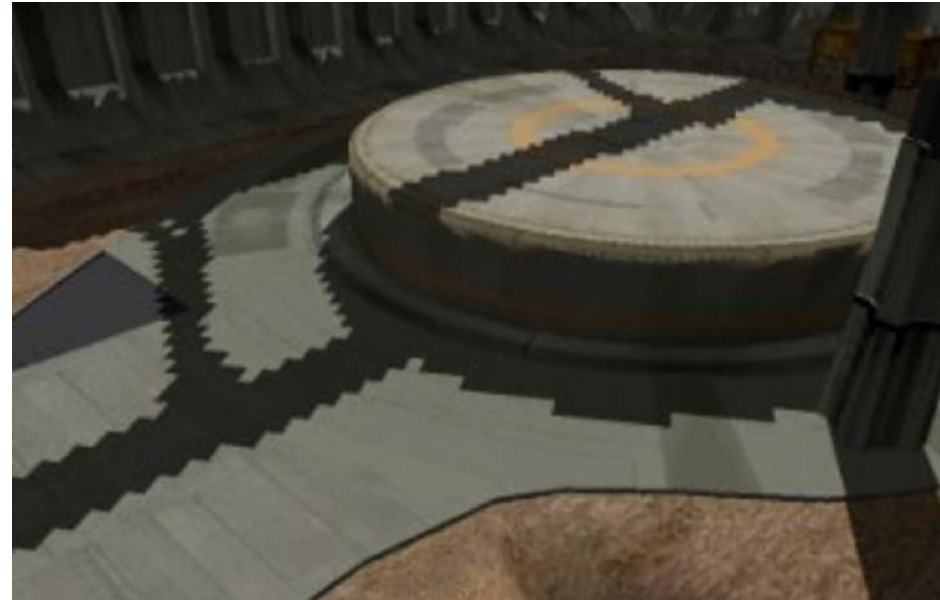
TAMPON DE PROFONDEUR - *SHADOW MAP*

- effet d'escaliers aux silhouettes des ombres
- *acné de surface* (ajoute epsilon à la distance du point)

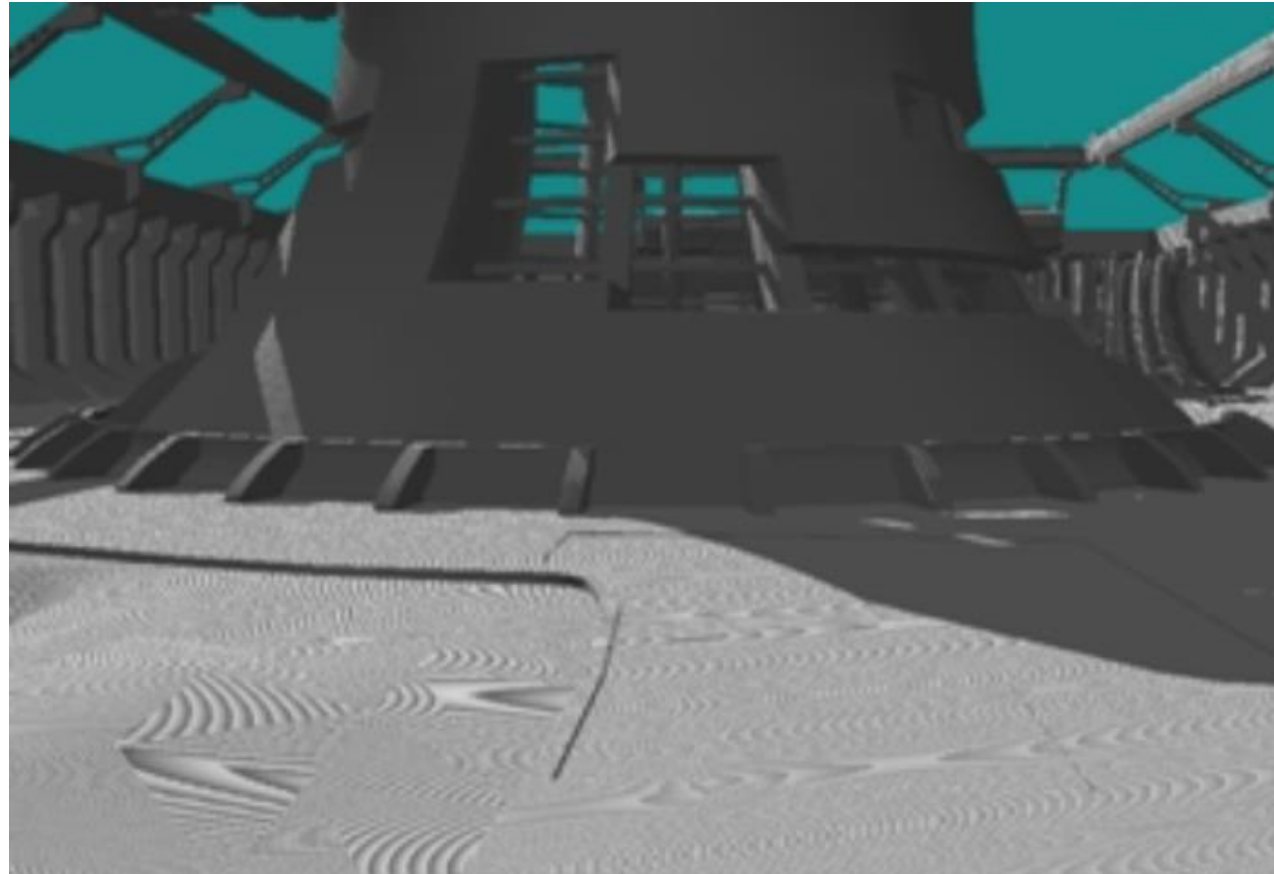


TAMPON DE PROFONDEUR - *SHADOW MAP*

Aliassage



TAMPON DE PROFONDEUR - *SHADOW MAP*



Acné de surface