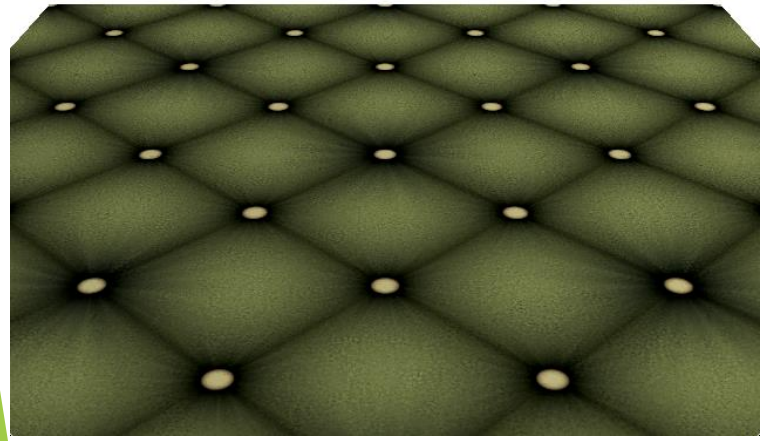
Rendu 3D – Normal mapping

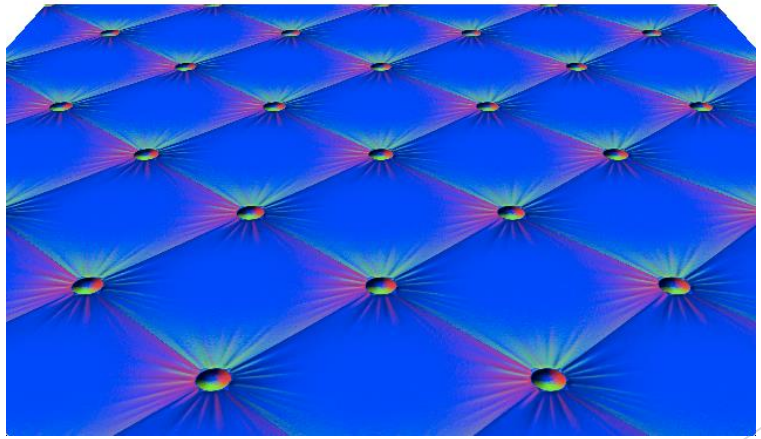
David Murray

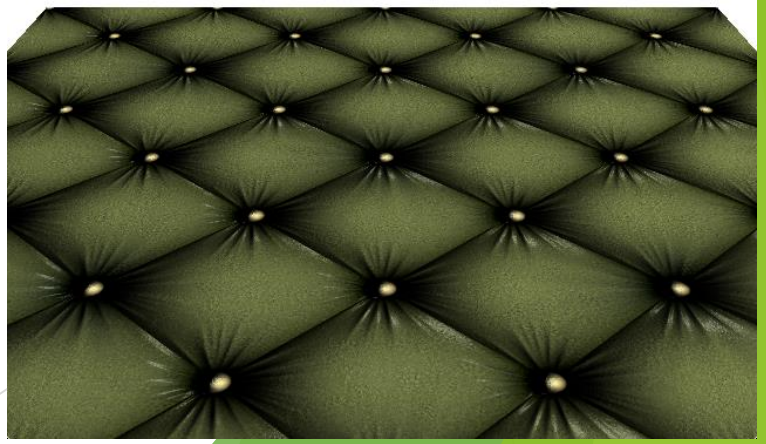
Source: Gael Guennebaud, Pierre Bénard

Idée générale

- Très couteux d'utiliser des modèles très détaillés pour représenter des petits détails
  - Avoir des normales détaillées suffisent pour ces petits détails
- Normal map : représentations des variations de normales
  - Utilisation pour le shading par fragment





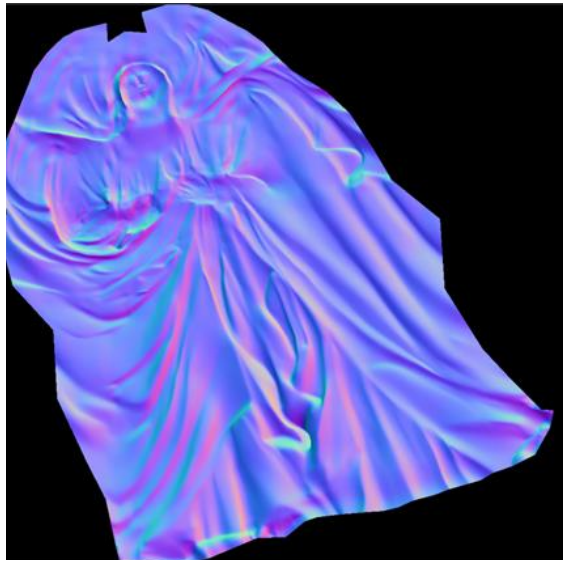


Idée générale

- Comment on stocke une normale (vecteur) dans une texture (image RGB) ?
- $R = N_x, G = N_y, B = N_z$
- Attention: $(N_x, N_y, N_z) \in [-1, 1]$ alors que $(\mathrm{R}, \mathrm{G}, \mathrm{B}) \in [0, 1]$
  - Besoin de faire un mapping pour stocker la normale
  - Besoin de faire l'inverse pour y acceder

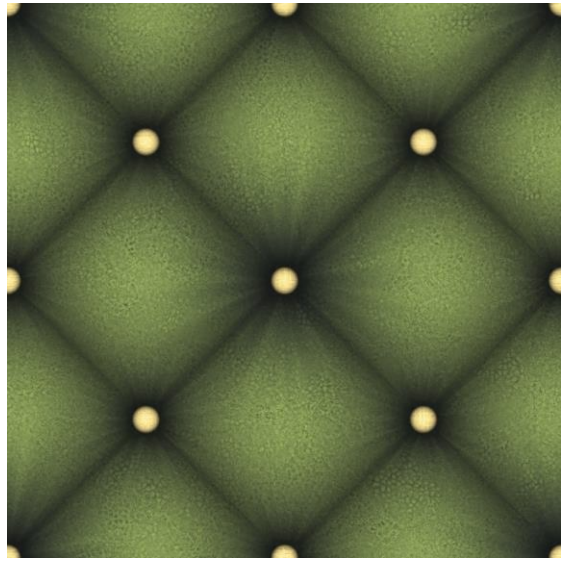
Génération d’un carte de normale: Maillage

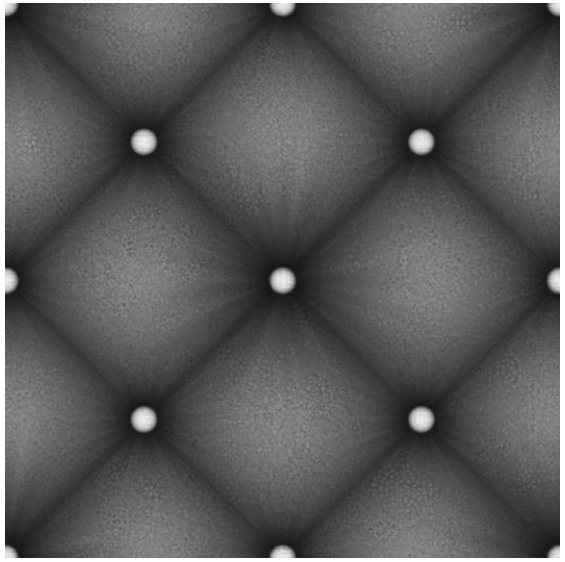
- Point de départ: un maillage très détaillé
  - Rastérisation du maillage
  - Calcul et stockage des normales
    - Attention: Il faut les stocker en espace objet ou tangent !

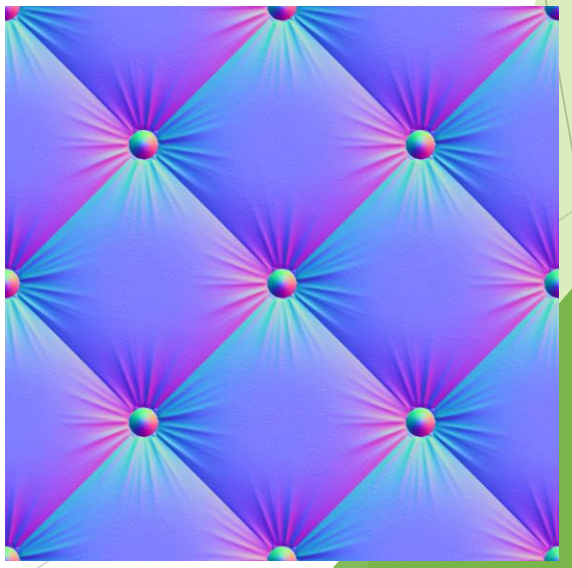


Génération d’un carte de normale: Image

- Point de départ: une image
  - Conversion en champ d’élévation (niveaux de gris)
  - Calcul des normales (gradient du champ d’élévation)
    - Attention : les normales seront forcément obtenu dans l’espace tangent à l’image







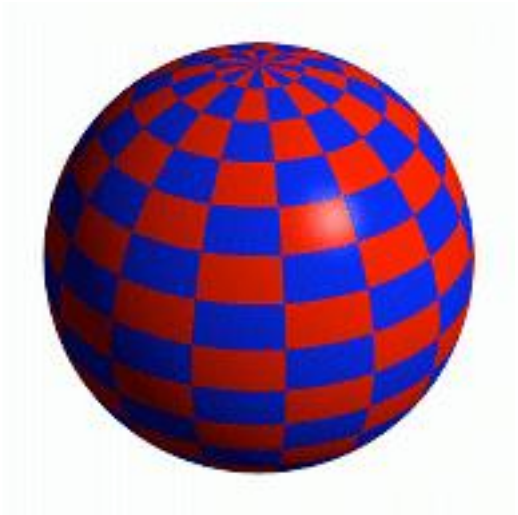
L’espace tangent

- Normal map: espace tangent
  - Pour un triangle, défini par 3 vecteurs:
    - $\vec{N}$ la normale
    - $\vec{T}$ la tangente
    - $\vec{B}$ la cotangente
    - $\vec{T}$ et $\vec{B}$ peuvent être obtenus à partir des coordonnées de textures
- Il faut repasser en espace objet pour calculer correctement l’éclairage !

Variante: le Bump Mapping

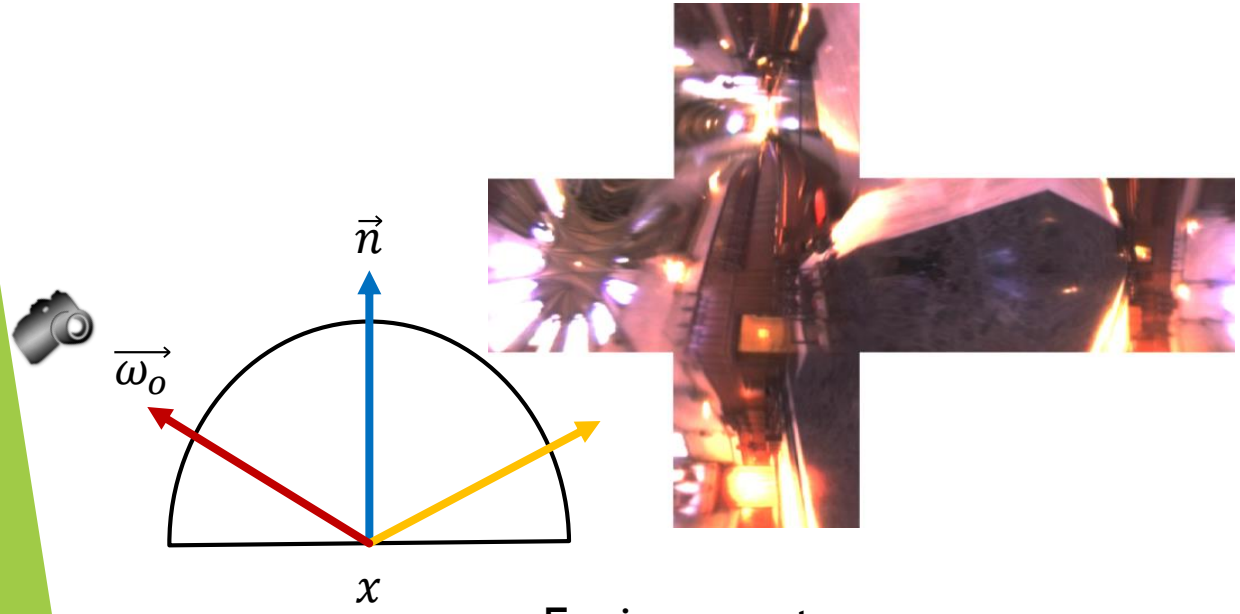
- Idée: utiliser uniquement une carte de hauteur pour perturber les normales
- Calcul de la perturbation à la volée en dérivant cette carte

- Attention: contrairement à une normal map, on stocke uniquement une perturbation !



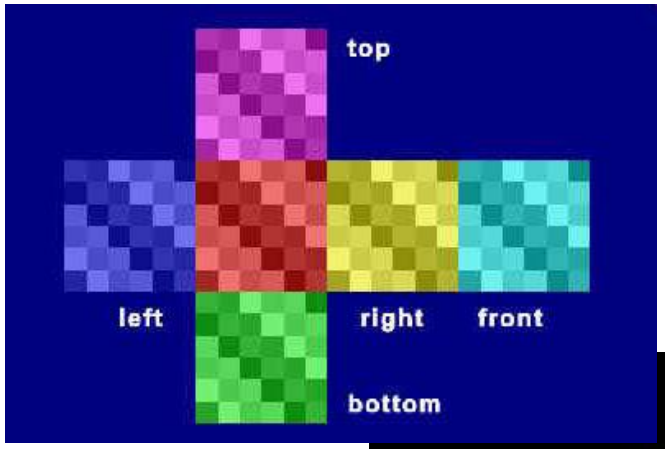
Environment Mapping

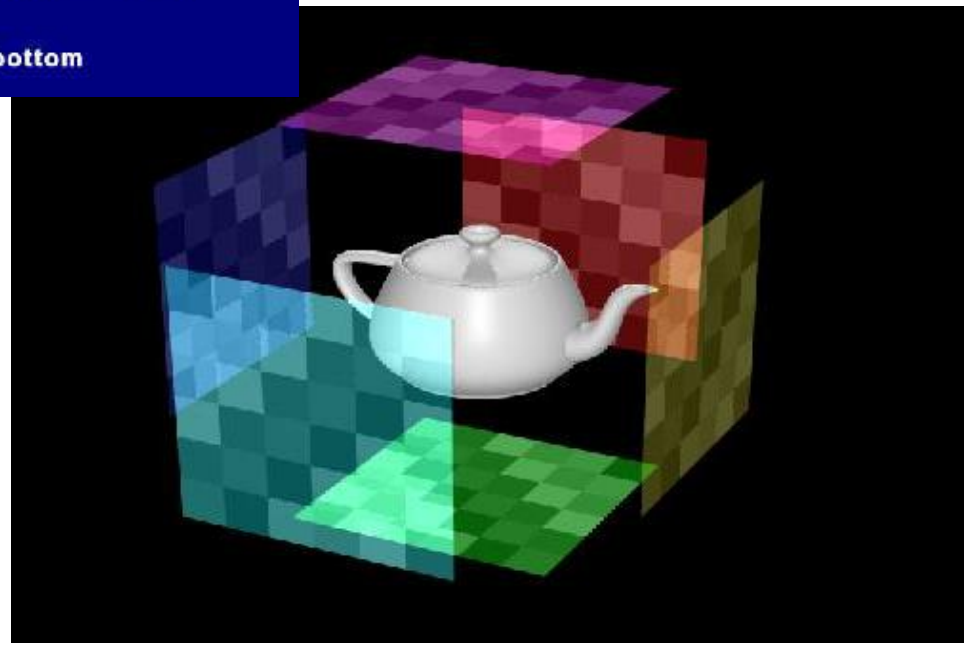
- Rappel: simuler un environnement -> une texture cubique autour de l'objet

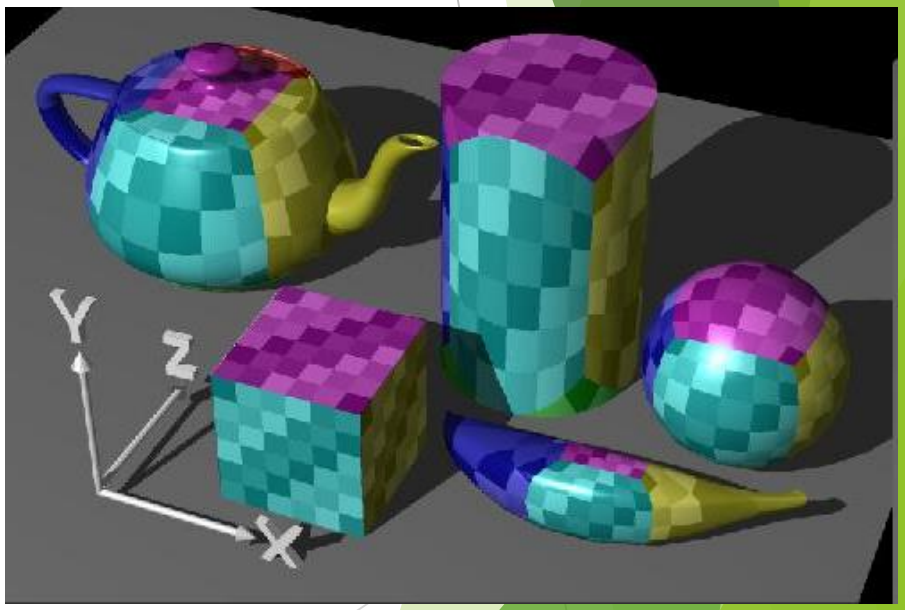


Environment map

Principe en détail: la Cube Map







Principe en détail: la Cube Map

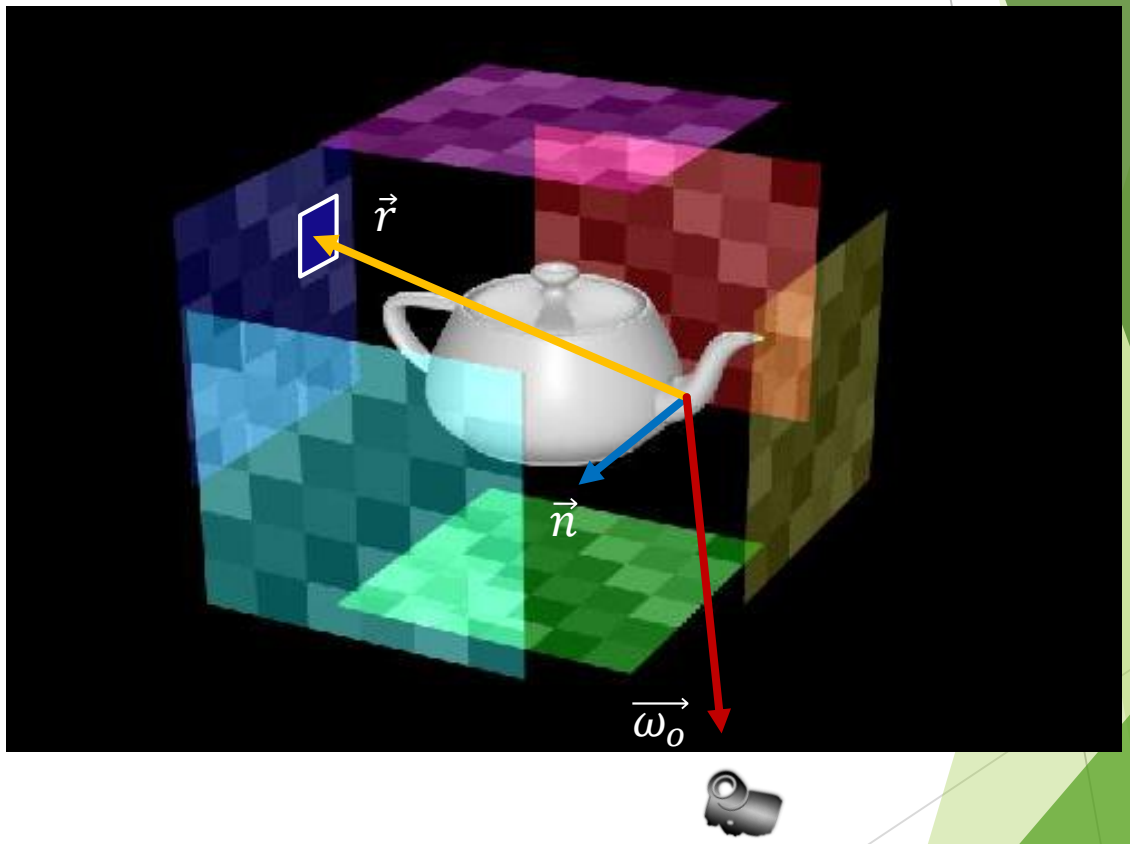
- Equivalent à 6 textures 2D représentant un cube centré sur l'origine
- Accès avec des coordonnées 3D (r_x, r_y, r_z)
- La transformation 3D -> 2D se fait automatiquement pour accéder au bon élément de la bonne face
  - $\text{Face} = \max(r_x, r_y, r_z)$, on nommera sc et st les deux autres
  - $s = \frac{\left(\frac{sc}{abs(Face)}+1\right)}{2}$
  - $t = \frac{\left(\frac{tc}{abs(Face)}+1\right)}{2}$

Principe en détail: la Cube Map

- En OpenGL :
  - Créer 6 textures 2D avec le flag:
    - GL_TEXTURE_CUBE_MAP_{POSITIVE,NEGATIVE}_{X,Y,Z}
  - Pour chaque face, l'image correspondante
- En GLSL :
  - samplerCube
  - Accès avec un vec3

Environment Mapping

- Accès avec le vecteur réflexion
- Même calcul que pour Phong ☺



A vous !