Synthèse d'images

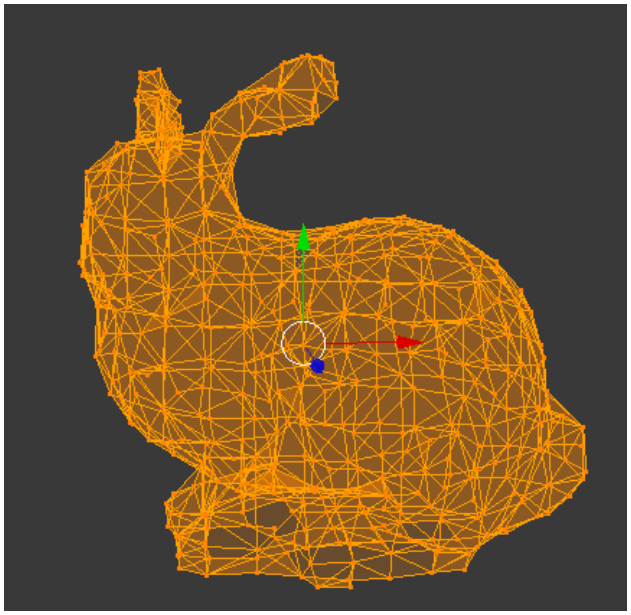
TD3

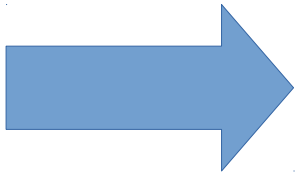
Robert Schuman
UNIVERSITÉ DE STRASBOURG

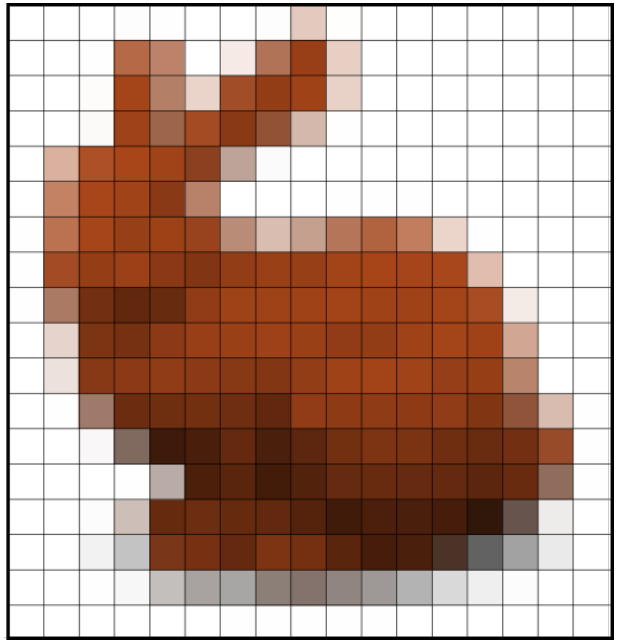
Pipeline graphique

Robert Schuman
UNIVERSITÉ DE STRASBOURG

Rappel

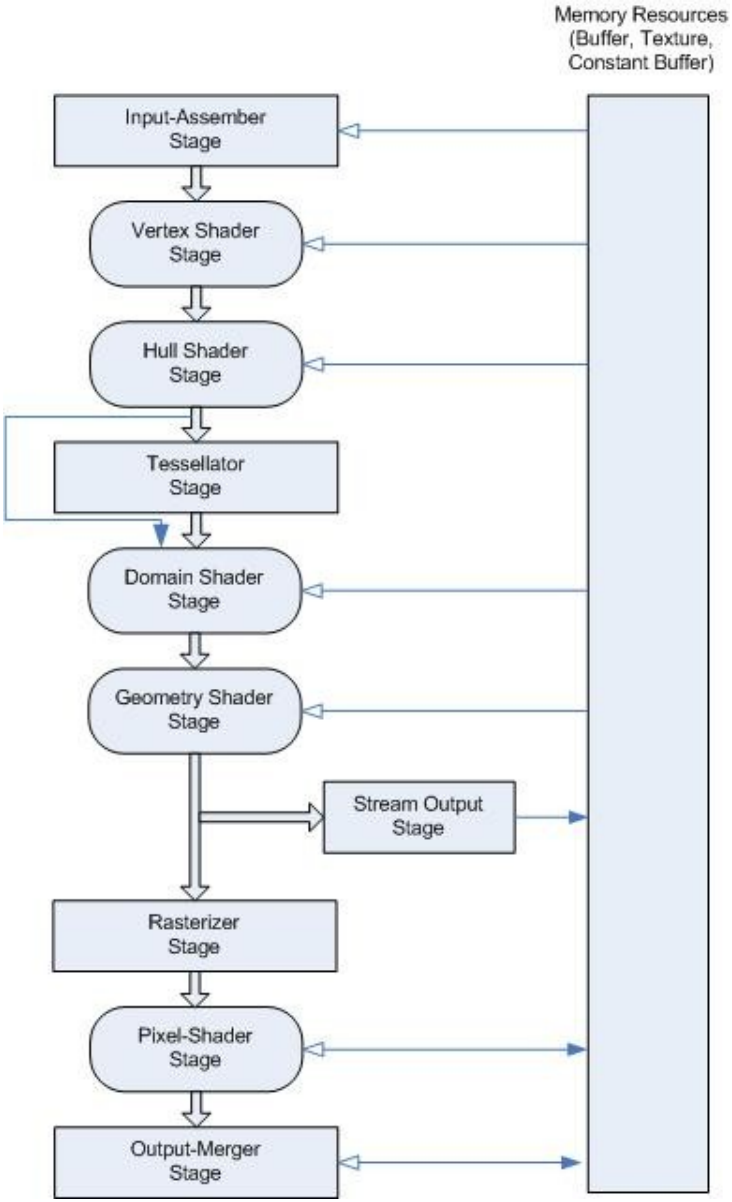




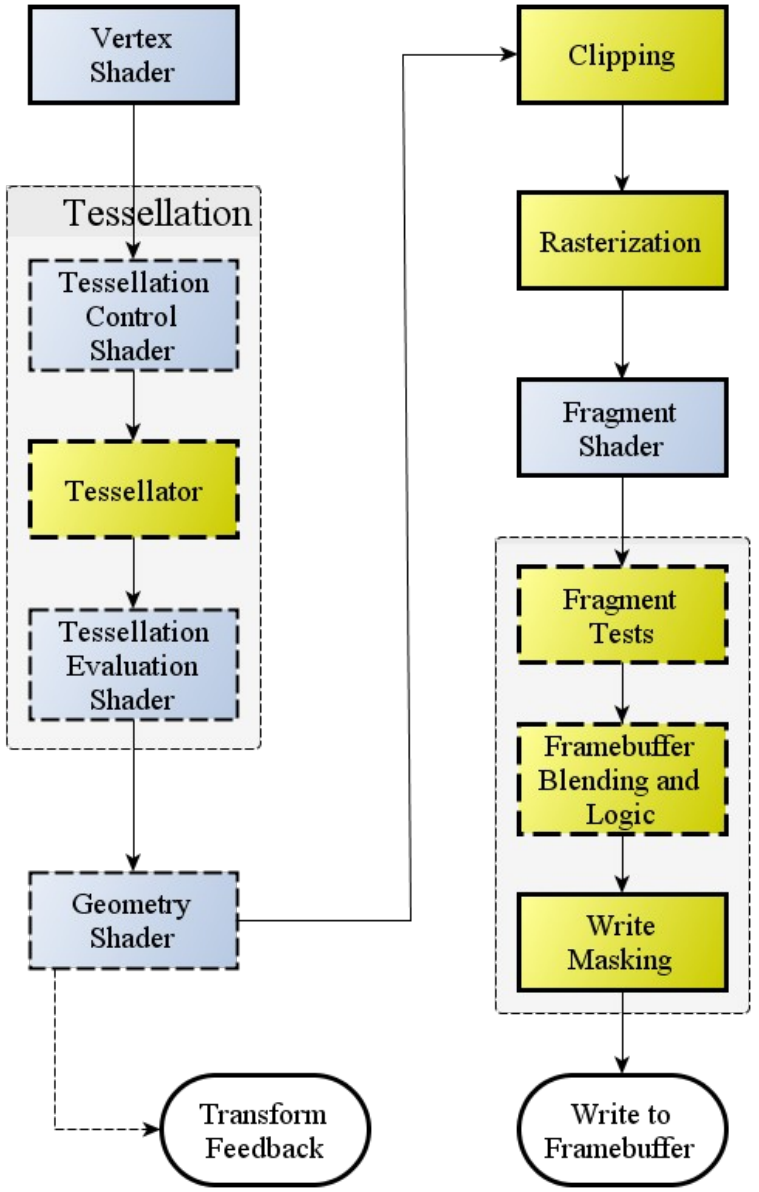


Robert Schuman
UNIVERSITÉ DE STRASBOURG

Pipeline Graphique



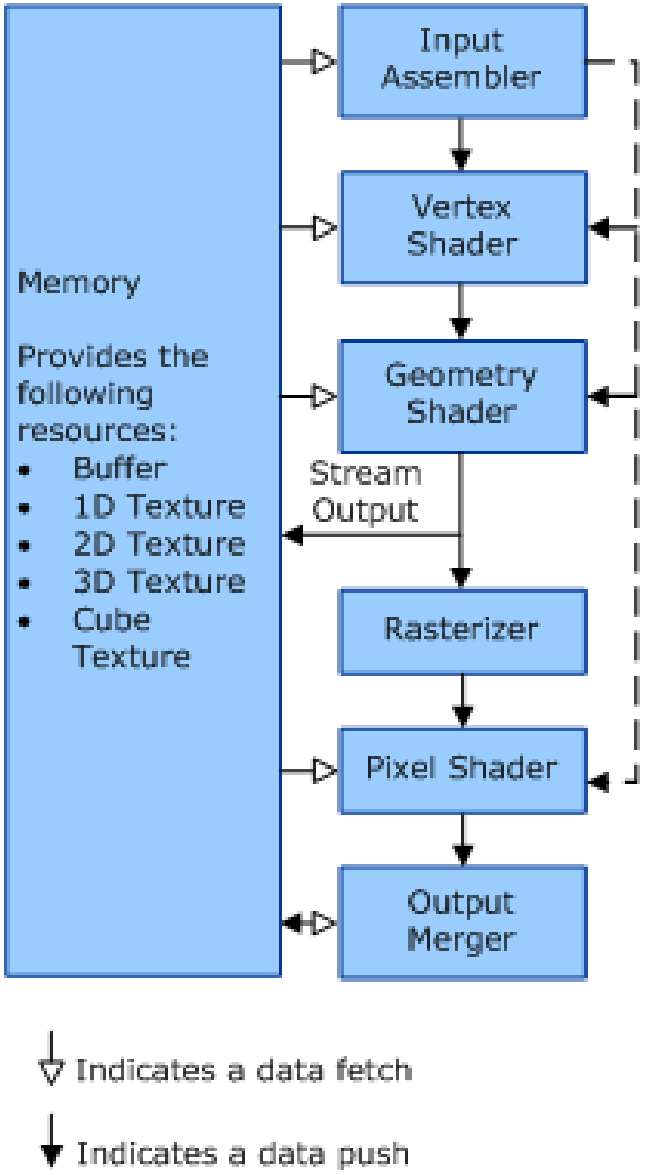
(source : msdn)



(source : opengl.org/wiki)

Robert Schuman
UNIVERSITÉ DE STRASBOURG

Pipeline Graphique simplifié

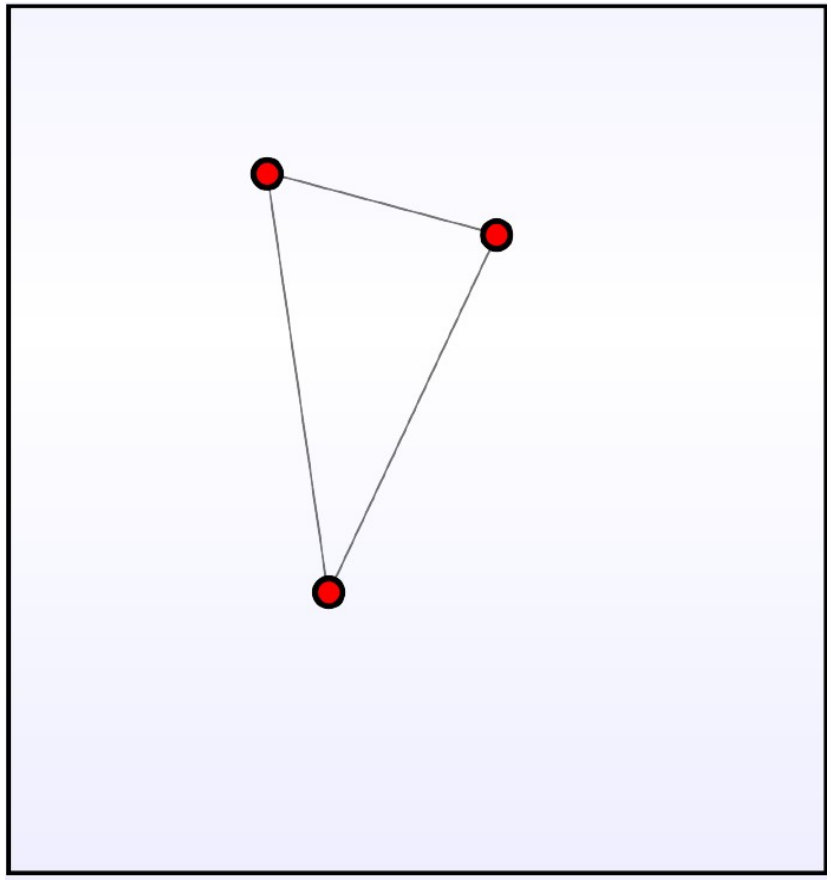


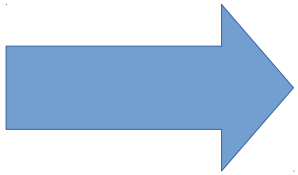
(source : msdn)

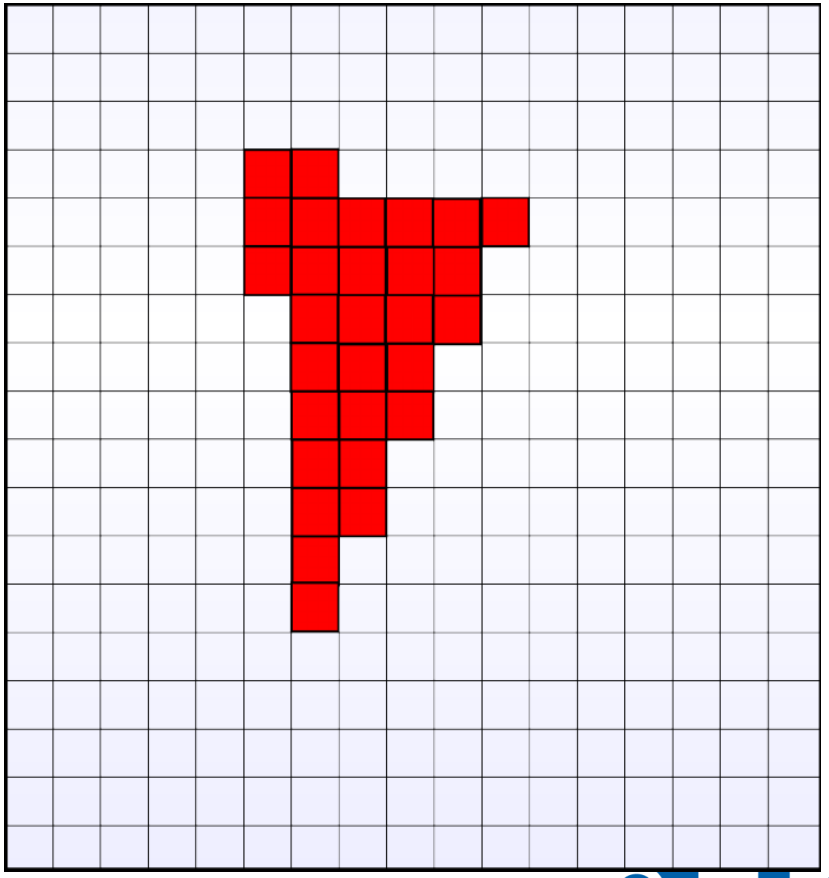
Utilisation du pipeline graphique

- Étapes contrôlables dans une certaine mesure
  - *Input assembly, rasterizer, output merger*
- Étapes entièrement programmables
  - *Vertex, pixel, geometry, domain, … shaders*
  - Fournir des programmes (code)
- Évolution vers du calcul générique
  - *GPGPU, compute shaders*

Rappel







Vertex Shader : que retenir ?

- Données géométriques en entrée
  - stockées en mémoire graphique
  - Stocké par sommet : vertex buffer
- Transformation géométrique des données
  - Dans le repère d'affichage
- Données dans le repère de projection
  - Factuellement, l'écran
  - Prêtes pour le pixel shader

Pixel Shader : que retenir ?

- Données géométriques interpolées en entrée
  - Interpolation après rastérisation
  - Par pixel à l'écran
- Calcul de la couleur finale
  - Selon un modèle
    - Physique, réaliste ou non réaliste (MeshBasicMaterial)
- Données par pixel en sortie
  - Couleur

Shaders

Robert Schuman
UNIVERSITÉ DE STRASBOURG

Exemple vertex shader

```
void main() {
	vec4 worldPos = modelMatrix * vec4(position, 1.0);
	gl_Position = projectionMatrix * viewMatrix * worldPos;
}
```

Vertex shader avec les mains

- Entrées :
  - Vertex buffer : positions, normales, uvs, …
    - Apparaît dans le shader sous la forme *attribute*
  - Uniform : paramètres supplémentaires
    - Comme des paramètres de fonctions
- Action : code du programme
- Sorties :
  - Built-ins : *gl_position*
  - Personnalisables
    - Apparaît dans le shader sous la forme *varying*
    - Pour être récupérées interpolées dans le pixel shader

Exemple pixel shader

```
void main() {
      gl_FragColor = vec4(1.0,0.0,0.0, 1.0);
}
```

Pixel shader avec les mains

- Entrées :
  - Personnalisables :
    - Apparaît dans le shader sous la forme *varying*
    - Récupérées depuis le *vertex shader*
  - Uniform : paramètres supplémentaires
    - Comme des paramètres de fonctions
- Action : code du programme
- Sorties :
  - Built-ins : *gl_FragColor*, couleur du pixel

Éléments de syntaxe GLSL

- Compilé par l'API OpenGL
  - Fournir le code source à l'API
- Scalaire C (float, bool, int, double)
- Vecteur : vec2, vec3, vec4 (ivec, dvec)
- Matrice : mat2, mat3, mat4, mat2x3,…
- uniform, varying, attribute

https://www.opengl.org/wiki/Data_Type_(GLSL)

Éléments de syntaxe GLSL (suite)

- gl_Position (clip space)
- gl_FragColor

Éléments Three.js

- Pour utiliser des shaders personnalisés :

  ```
  var material = new THREE.ShaderMaterial(…)
  ```

  https://threejs.org/docs/#api/materials/ShaderMaterial

  https://threejs.org/docs/#api/renderers/webgl/WebGLProgram

Éléments prédéfinis Three.js - vertex

- uniform mat4 modelMatrix;
- uniform mat4 modelViewMatrix;
- uniform mat4 projectionMatrix;
- uniform mat4 viewMatrix;
- uniform mat3 normalMatrix;
- uniform vec3 cameraPosition;
- attribute vec3 position;
- attribute vec3 normal;
- attribute vec2 uv;
- attribute vec2 uv2;

Éléments prédéfinis Three.js - pixel

- uniform mat4 viewMatrix;
- uniform vec3 cameraPosition;

Surfaces et matériaux

Robert Schuman
UNIVERSITÉ DE STRASBOURG

Rappel

- Un objet se perçoit par :
  - Sa forme (TD2)
  - Son aspect
- Le matériau va définir l'aspect de la surface
  - Couleur et réaction à la lumière

Perception

- Un organe : l'œil
  - Rétine, cônes et bâtonnets
  - Réagit aux ondes électromagnétiques reçues
  - Interprétation par le cerveau
- Proposer une information à l'œil
  - Information spectrale dans la lumière visible
  - Au moyen des pixels via le moniteur
    - en variant couleur et intensité émise par chaque pixel

Quelles valeurs de pixels ?

- Arbitraire ?
  - MeshBasicMaterial !
  - Coloriage, dessin, ...
- Modèle/Algorithme ?
  - Modèle « physique »
    - Optique géométrique
    - Électromagnétisme
    - Approximations
    - MeshLambertMaterial, MeshPhongMaterial, etc.
  - Modèle non réaliste → Arbitraire !
- Personnalisable ?
  - MeshShaderMaterial !

Équation de rendu

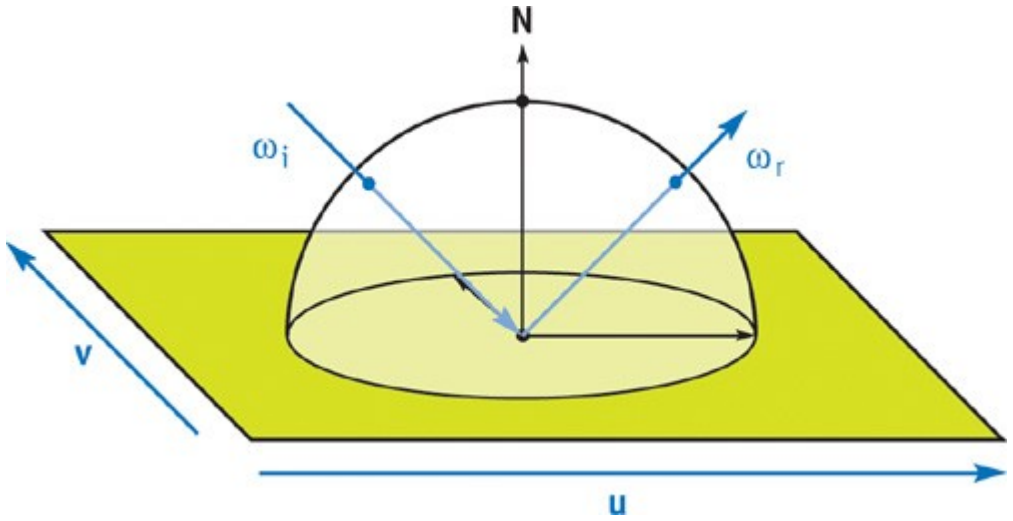
- Kajiya (84)
- Formalisation mathématique du transport de la lumière

$$L_o(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_\Omega f_r(\mathbf{x},\omega_i,\omega_o) L_i(\mathbf{x},\omega_i)(\omega_i \cdot \mathbf{n}) d\omega_i$$

Modèle qui marche plutôt bien

Fonction de réflectance

- BRDF : *Bidirectional reflectance distribution function*
- fonction à 4 dimensions



(source : nvidia.com/GPUGems)

BRDF

- Ce sont des « modèles » au sens physique
  - Formule mathématique essayant de reproduire la réalité
- Les plus connus :

  Lambert, Phong, Blinn, Cook-Torrance (micro-facettes)

Surface diffuse : Lambert

- Fonction de réflectance constante
  - Vu en TP1 (MeshLambertMaterial)
- Radiance constante dans toutes les directions
- Lobe en cosinus (cf. équation de rendu)

Surface diffuse

- Raccourci Diffuse = Lambert
- Autre modèle : Oren-Nayar

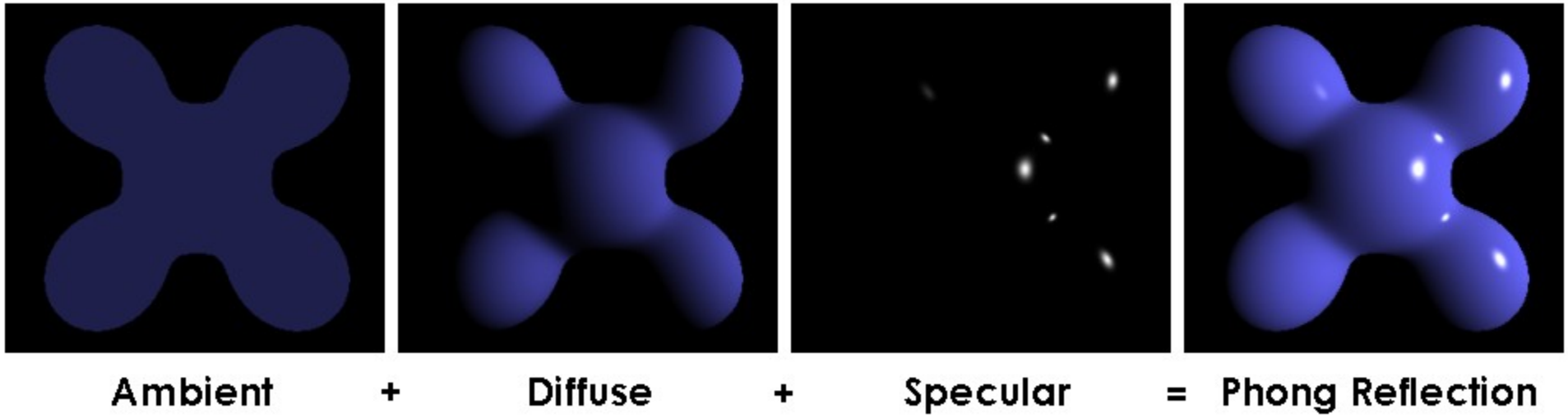
(source : wikipedia)

Réflexion spéculaire

- Diffuse = surface matte
- Surface type plastique ? Métaux ?
- Composante spéculaire
  - Agit comme un miroir plus ou moins parfait

Modèle de réfléctance de Phong

- Composante spéculaire



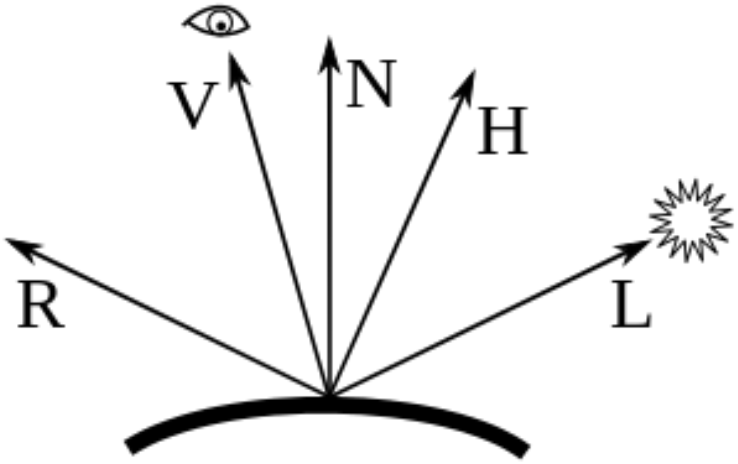
$$I_p = k_a i_a + \sum_{\mathrm{m} \in \mathrm{lights}} (k_d(\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s(\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,s}).$$

Robert Schuman
UNIVERSITÉ DE STRASBOURG

(source : wikipedia)

Modèle de réfléctance de Phong (suite)

- Three.js : MeshPhongMaterial
- Variante : Blinn-Phong
  - Terme spéculaire calculé différemment
  - dot(R,V) « remplacé » par dot(N,H)



(source : wikipedia)

Aparté : pipeline artistique

- Diffuse + specular (Phong)
  - Difficile à contrôler et appréhender par des graphistes
- Metalness + roughness (micro-facettes + GGX)
  - Plus intuitif, on se préoccupe juste de la matière
  - Three.js : MeshStandardMaterial
  - → PBR (p*hysically based rendering*)
    - https://www.marmoset.co/posts/basic-theory-of-physically-based-rendering/

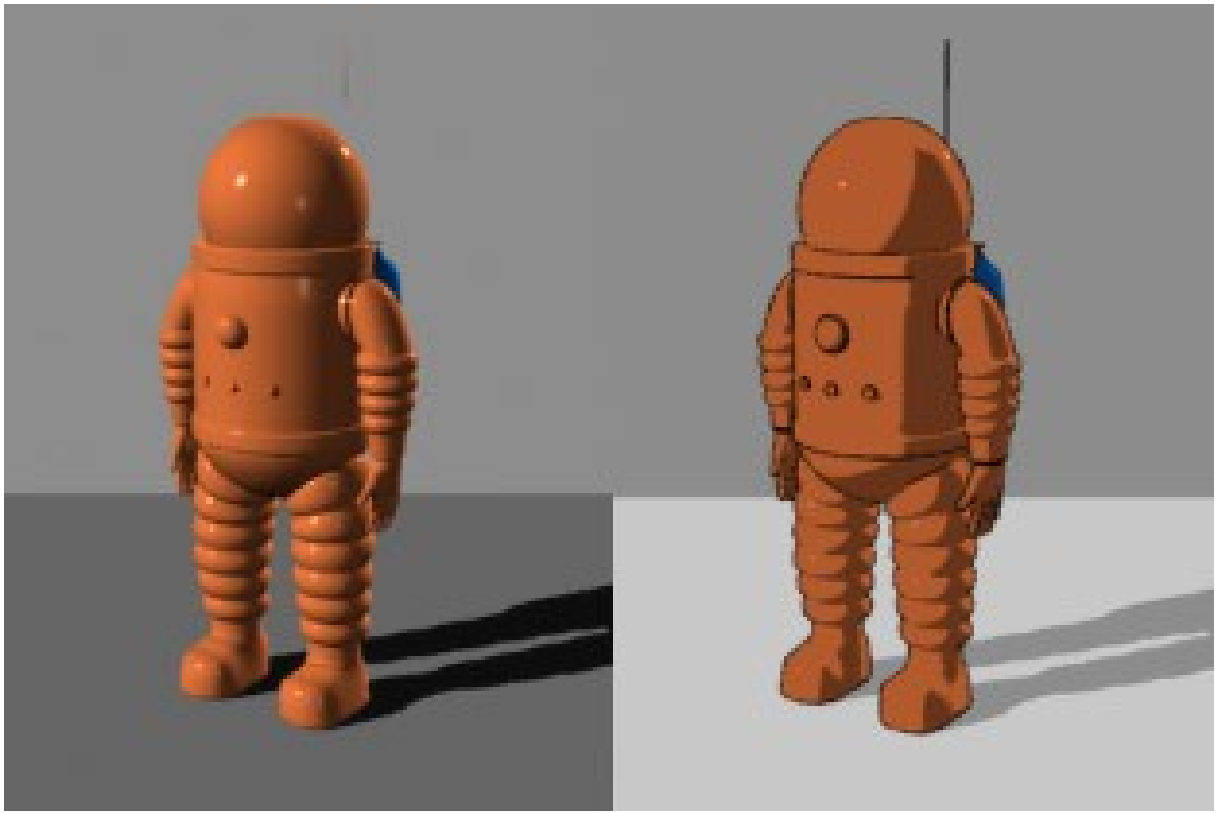
Aparté : Modèles non réalistes

- NPR (Non-photorealistic rendering)
- Les modèles proposés :
  - Pas de réalité physique
  - Une approche artistique intentionnelle
    - Effets coup de pinceau/crayonné
    - Mangas
    - Illustration technique/industrielle

Modèles non réalistes

- Cel / Toon Shading (manga)

  MeshToonMaterial



(source : wikipedia)

Un peu de maths

Robert Schuman
UNIVERSITÉ DE STRASBOURG

Coordonnées homogènes

- Manipulation vectorielle dans un espace projectif (géométrie projective)
- Algèbre géométrique, algèbre de Clifford
- Dimension n+1 par rapport à l'espace affine correspondant
- À manipuler et comprendre avec les mains

Matrices

- Dimension 4
- Définit une transformation de l'espace
- Multiplication associative

Matrices (suite)

- World/Model/Object
- View/Camera
- Proj/Projection