

Modélisation Animation Rendu

Master 2 Informatique GL et MiTiC

Version 2014

Rémi Cozot, Fabrice Lamarche

Plan MAR 2014

- Introduction & Objectifs (rC)
- Modélisation 3D (rC)
- Rendu (rC)
- Animation (fL)

Introduction

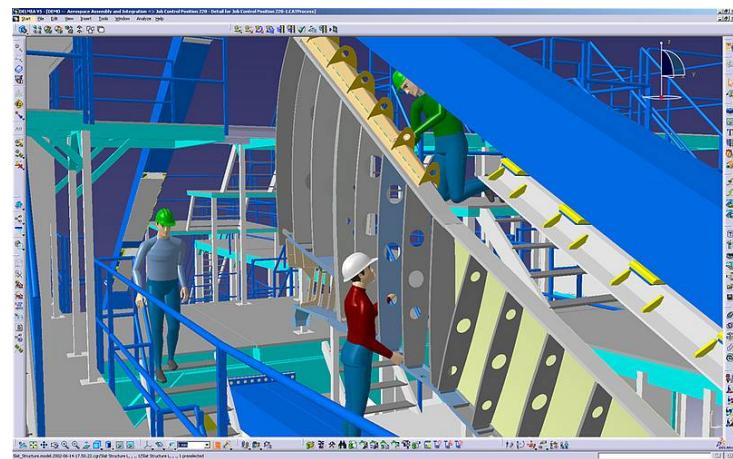
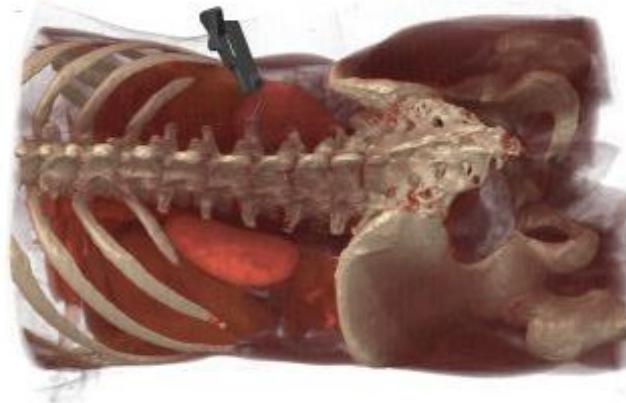
Usages et domaine

Moteur 3D

Objectifs du cours

Introduction

- Utilisations de la 3D
 - Visualisation de données
 - Aide à la conception
 - Communication
 - Divertissement



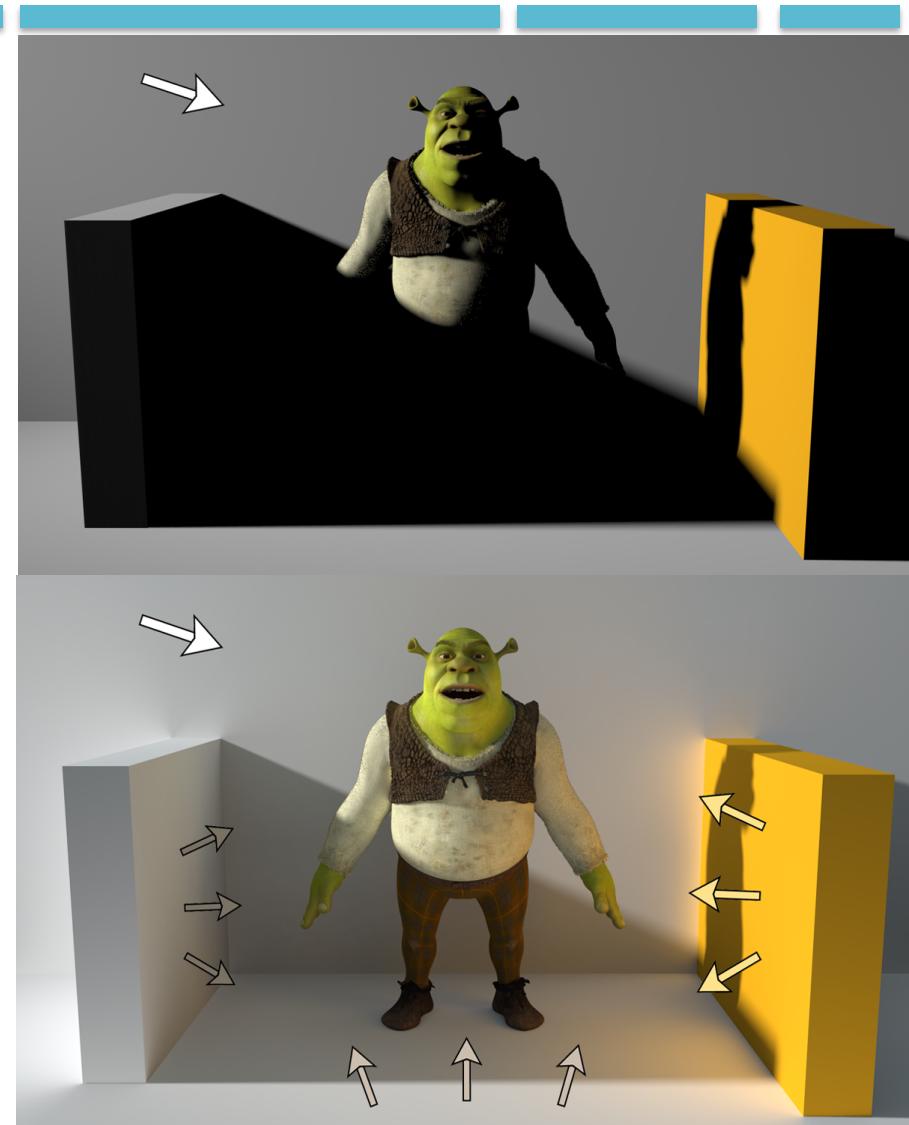
Introduction

- Cas d'utilisation
 - Réalité virtuelle (et augmentée)
 - Jeux Vidéos
 - Synthèse d'images
 - Visualisation de données



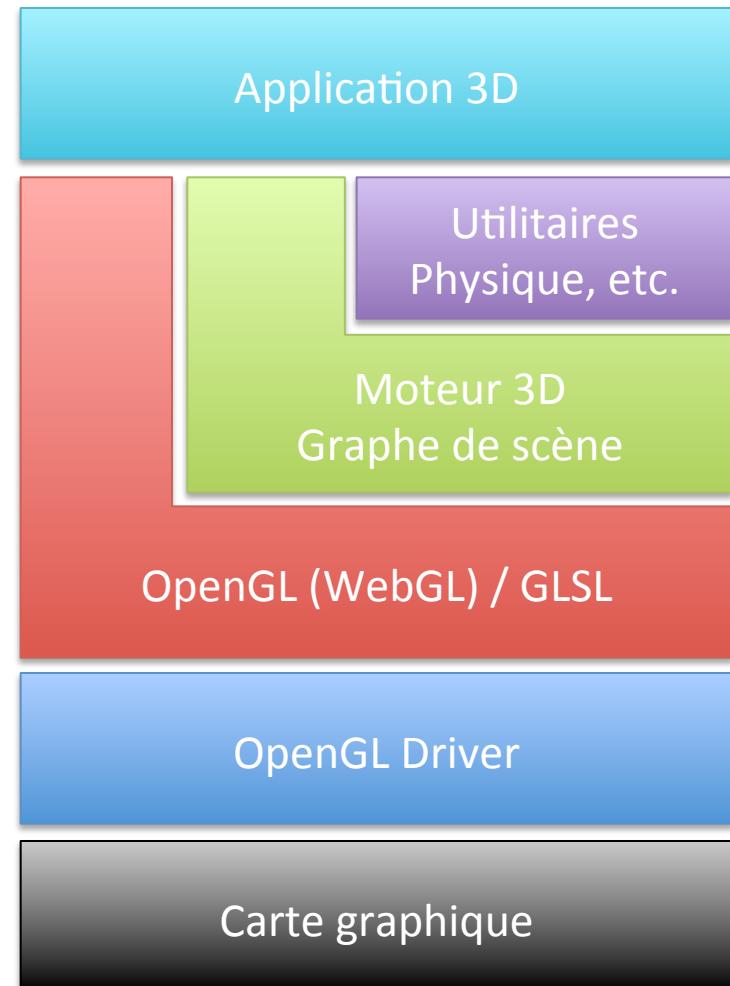
Introduction

- Modélisation
 - Définir la scène 3D
 - Choix du modèle de représentation
- Animation
 - Définir les évolutions temporelles des paramètres de la scène 3D
- Rendu
 - Calcul de la séquence d'images
 - Effets pris en compte
 - Exemple : ombre portée, illumination globale, etc.



Introduction

- Architecture application 3D
 - Différentes couches
 - Bibliothèque graphique de base
 - OpenGL
 - DirectX
 - Moteur 3D
 - ThreeJS (pour les TPs)
 - Ogre3D (C++)
- Applications all-in-one
 - Unity3D
 - Langage de script (Unity3D : javascript ou C#)



Objectifs

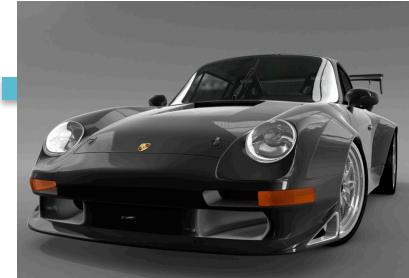
- Concepts fondamentaux
 - Modélisation, animation et rendu
- Utilisation d'un moteur 3D : ThreeJS
 - Programmation au dessus d'un moteur 3D
- Ce n'est pas
 - Programmation OpenGL d'un moteur 3D
 - Utilisation d'outils de modélisation, animation et rendu

Modélisation (3D)

Concepts fondamentaux
Utilisation ThreeJS

Modélisation

- Trouver une représentation/ un modèle
 - Permettant de calculer des images
 - Facilement éditable et manipulable
- « Objet » du Monde réel
 - Multiplicité des matières
 - Solide, gaz, granuleux, liquide, fourrure, etc.

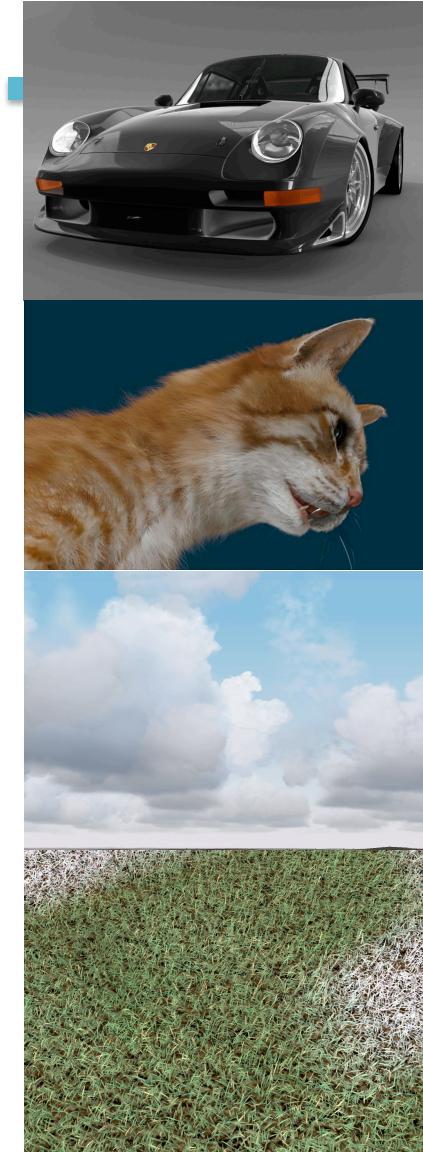


Modélisation

- Trouver une représentation/ un modèle
 - Permettre
 - Faciliter
- « Objet »
 - Multiples formes
 - Solide
 - etc.

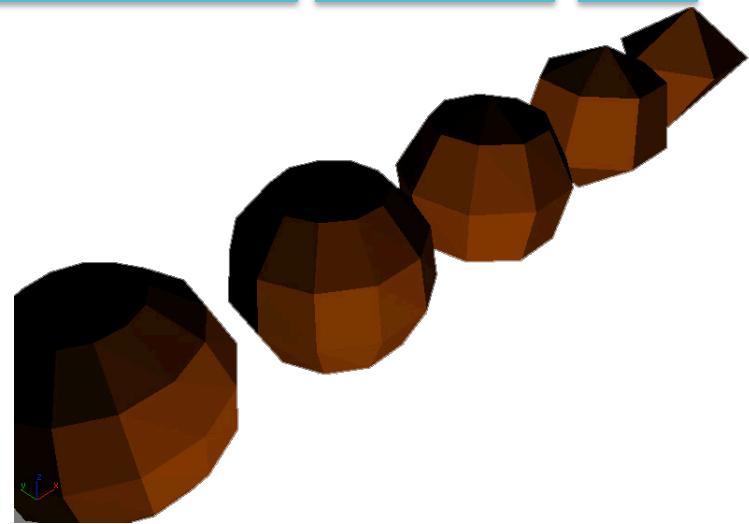
G. Bachelar :
Un modèle est toujours un compromis entre deux exigences : réalité et abstraction.

- Réalisme : trop de complexité pour être manipuler
- Abstraction : simple à manipuler, manque de réalité



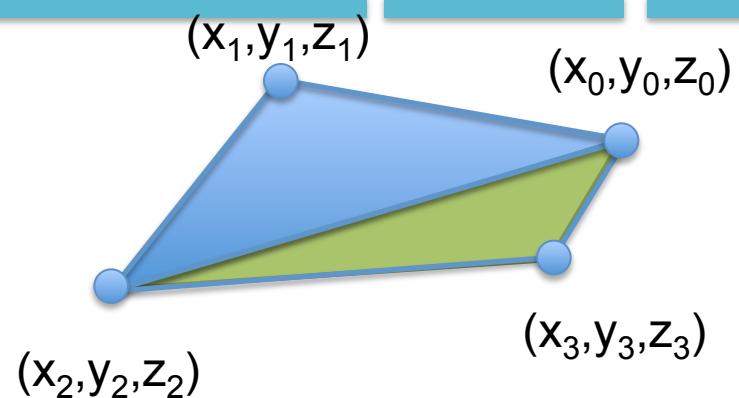
Modélisation

- Hypothèse et approche
 - Objet 3D défini par sa surface
{exclu les gaz}
 - On perçoit la surface des objets
 - Surface est approchée par une ensemble de surfaces simples
{approximation}
 - Surfaces simples : triangles
{modèle manipulable facilement}
- Conséquence
 - La précision dépend du nombre de triangles



Modélisation

- Modèle 3D
 - Ensemble de triangles
- Représentation de la géométrie
 - Ensemble de sommets
 - Liste de triangles
 - Par indirection
 - Évite la duplication de données

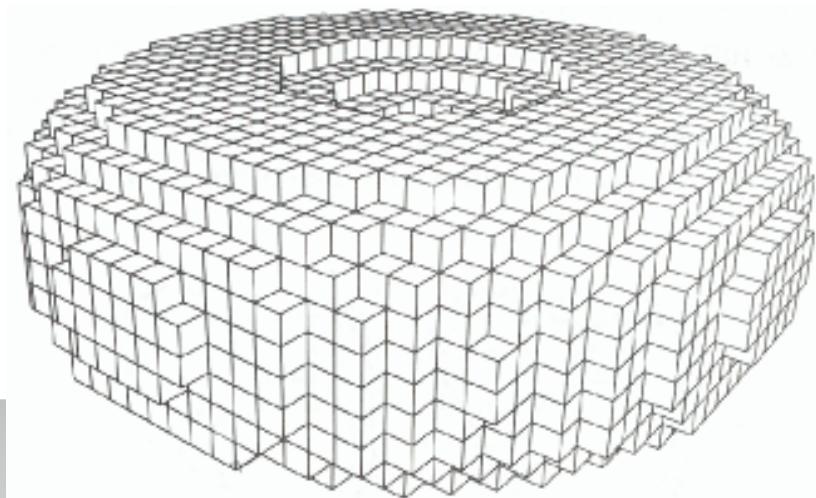
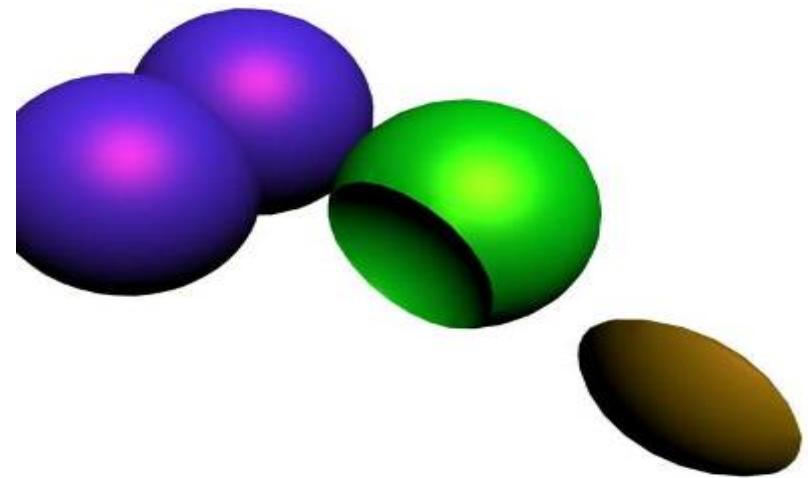
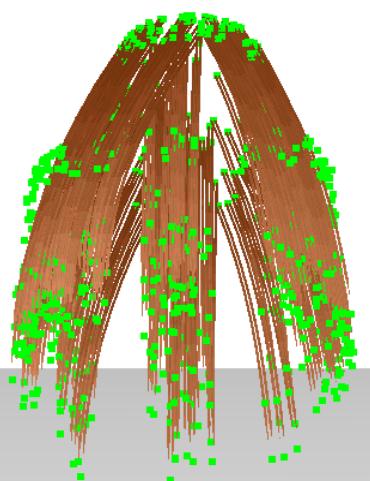
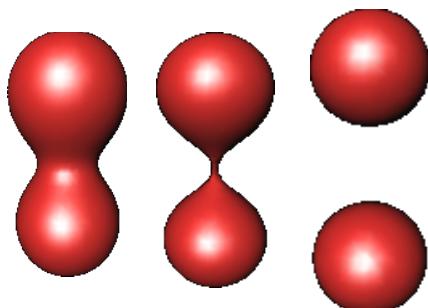


# Liste de sommets	
0	(x_0, y_0, z_0)
1	(x_1, y_1, z_1)
2	(x_2, y_2, z_2)
3	(x_3, y_3, z_3)

Liste des triangles
0,1,2
0,2,3

Modélisation

- D'autres modèles sont possibles



Modélisation

- Format d'échange simple : OBJ
 - Liste
 - Sommets
 - Coordonnées de textures*
 - Normales*
 - Triangles
 - Un sommet peut avoir plusieurs normales
- Modélisation
 - Logiciel de modélisation
 - Sketchup make, 3DS max, blender, maya, etc.
 - Export / import
 - OBJ, Collada, FBX, etc.

```

# cube.obj
g cube

# cube.obj: vertex
list
v 0.0 0.0 0.0
v 0.0 0.0 1.0
v 0.0 1.0 0.0
v 0.0 1.0 1.0
v 1.0 0.0 0.0
v 1.0 0.0 1.0
v 1.0 1.0 0.0
v 1.0 1.0 1.0

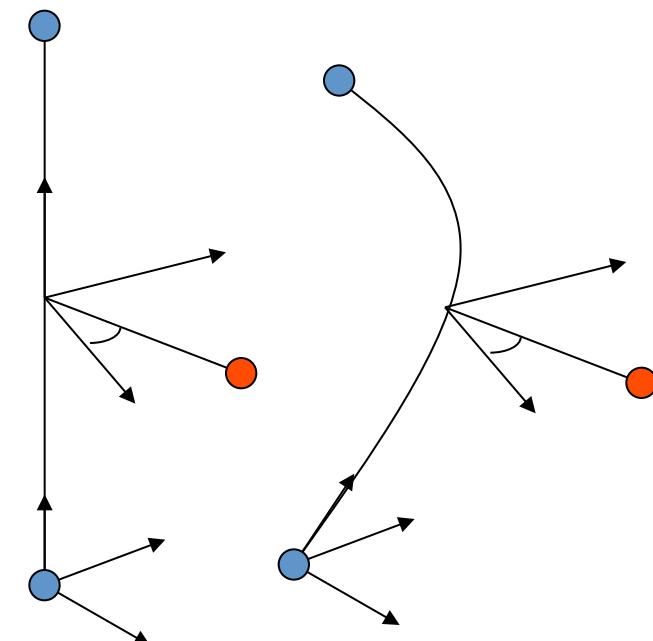
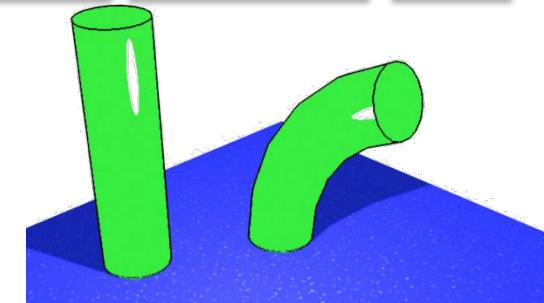
# cube.obj: face list
...
# cube.obj: normal
list
vn 0.0 0.0 1.0
vn 0.0 0.0 -1.0
vn 0.0 1.0 0.0
vn 0.0 -1.0 0.0
vn 1.0 0.0 0.0
vn -1.0 0.0 0.0

```

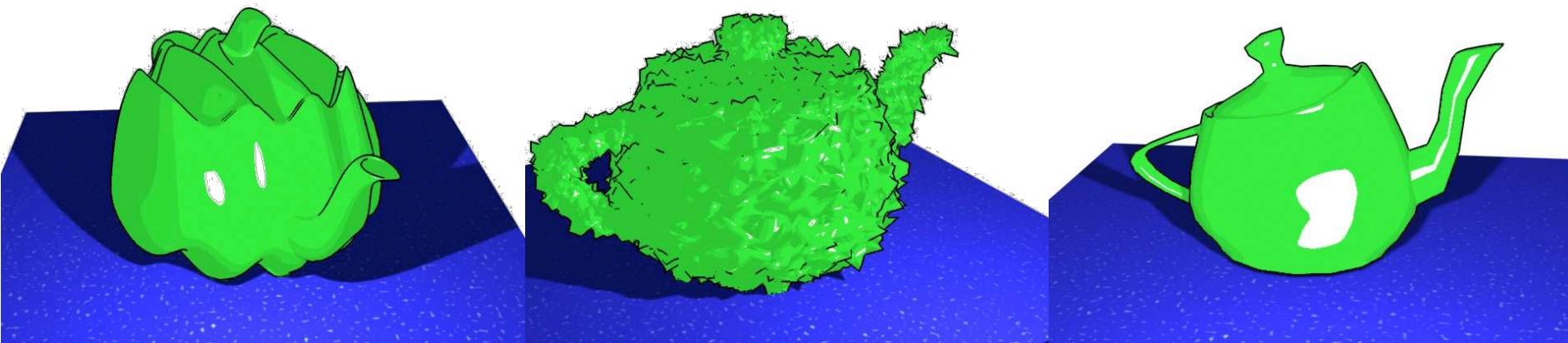
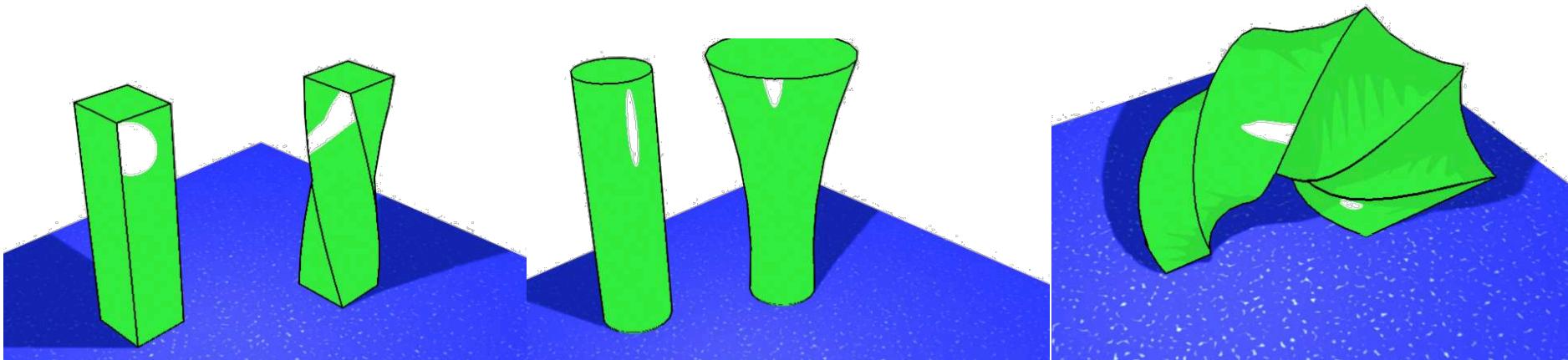
...

Modélisation

- Modélisation par listes de modifications
- Cadre général support de déformation (SDD)
- Idée : objet de base + SDD =
objet déformé
 1. Projeter les sommets dans l'espace paramétrique du SDD
 2. Modifier le SDD
 3. Recalculer les points dans l'espace cartésien

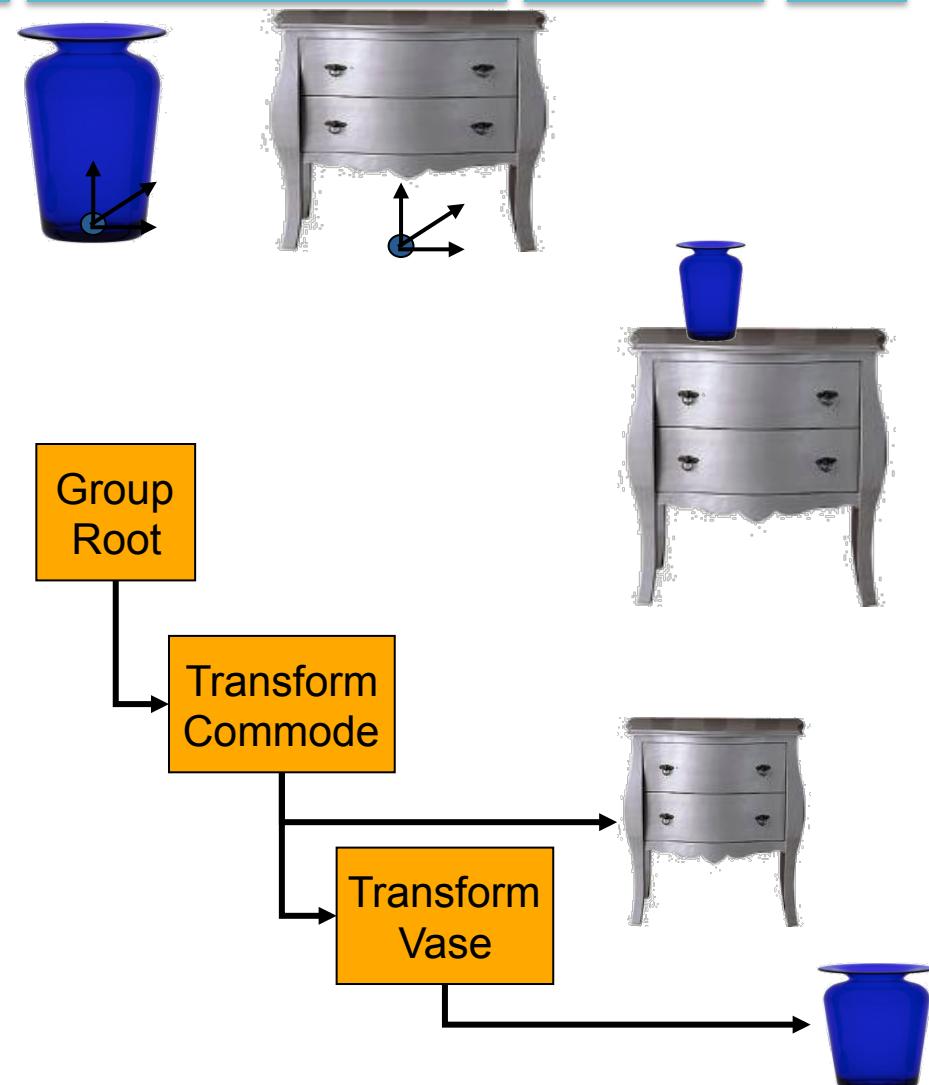


Modélisation



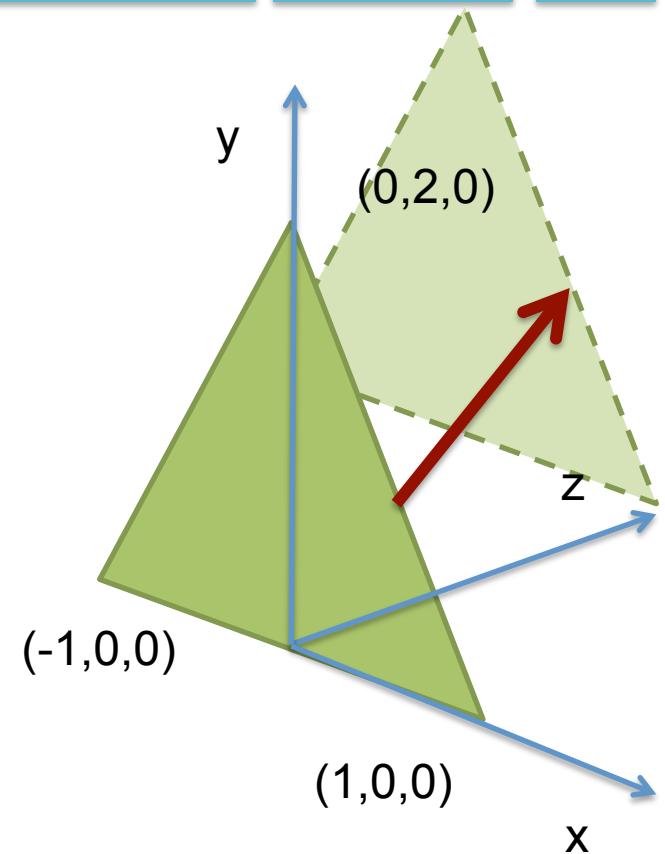
Modélisation

- Positionnement des objets les uns par rapport aux autres
- Transformations géométriques
 - Translation
 - Rotation
 - Mise à l'échelle
- Graphe de transformation : « graphe de scène »



Modélisation

- Objet = coordonnées de ses sommets
- Transformation géométriques
 - Matrice de transformations
 - Coordonnées homogènes
 - Toutes les transformations deviennent matricielles(4x4)
 - Composition de transformations
 - Produits de matrices



Modélisation

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Leftrightarrow \begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \\ \frac{z}{w} \end{bmatrix}$$

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

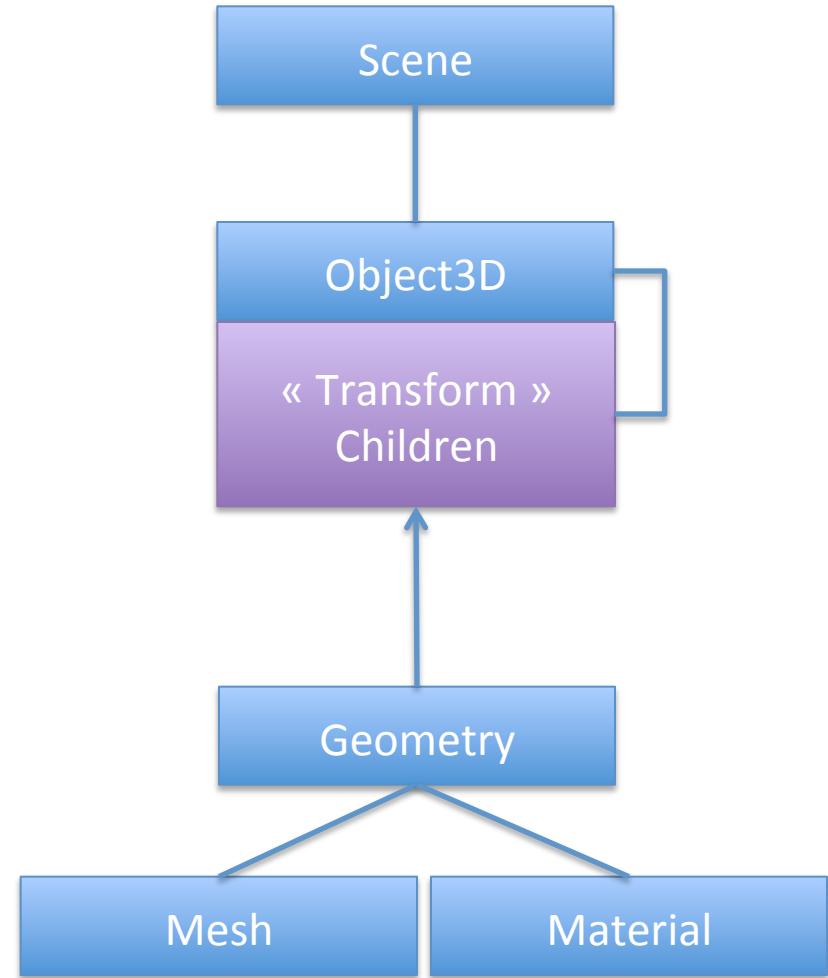
ThreeJS : squelette

```
<html>
<head>    <title>MAR 2014 - example 0 by RC&FL</title>
            <style> body { margin: 0; } canvas { width: 100%; height: 100% } </style> </head>
<body><script src="threejs/build/three.min.js"></script>
<script>
var scene = new THREE.Scene();                                // scene
var camera = new THREE.PerspectiveCamera(                    // camera
    75, window.innerWidth/window.innerHeight, 0.1, 1000 );
var renderer = new THREE.WebGLRenderer();                     // rendering engine
renderer.setSize( window.innerWidth, window.innerHeight );   // render size
document.body.appendChild( renderer.domElement );           // add a canvas

var geometry = new THREE.BoxGeometry( 10, 10, 10 );          // cube
var material = new THREE.MeshBasicMaterial( { color: 0x1010a0 } ); // color
var cube = new THREE.Mesh( geometry, material );             // mesh
scene.add( cube );                                         // add the mesh to the scene
camera.position.z = 20;                                     // camera position
var render = function () {                                    // rendering and animation
    requestAnimationFrame( render );
    cube.rotation.x += 0.01; cube.rotation.y += 0.01;
    renderer.render(scene, camera); };
render();                                                 // start
</script>
</body></html>
```

ThreeJS : éléments clés

- Objets indispensables
 - Scene : racine du graphe de scène
 - Camera
 - Moteur de rendu
- Graphe de scène
 - Mesh
 - geometry + material
 - Object3D
 - nœud « Transform »



ThreeJS : Object3D

- Object3D : base class for scene graph
 - Constructor
 - Object3D() : The constructor takes no arguments.
 - Properties ie Attributes
 - **id** : Unique number for this object instance
 - **name** : Optional name of the object
 - parent : Object's parent in the scene graph.
 - **children** : Array with object's children.
 - **position** : Object's local position.
 - **rotation** : Object's local rotation (Euler angles), in radians.
 - **eulerOrder** : Order of axis for Euler angles
 - **scale** : Object's local scale
 - up : Up direction
 - **matrix** : Local transform
 - quaternion : Object's local rotation as Quaternion. Only used when useQuaternion is set to true
 - useQuaternion : Use quaternion instead of Euler angles for specifying local rotation.
 - **matrixWorld** : The global transform of the object. If the Object3d has no parent, then it's identical to the local transform

ThreeJS : Object3D

- Object3D : base class for scene graph
 - Methods
 - applyMatrix (`matrix`) : This updates the position, rotation and scale with the matrix
 - translateX (`distance`) : Translates object along x axis by distance
 - translateY (`distance`) : Translates object along y axis by distance.
 - translateZ (`distance`) : Translates object along z axis by distance
 - **add (`object`)** : Adds object as child of this object.
 - remove (`object`) : Removes object as child of this object.
 - **traverse (`callback`)** : callback - An Function with as first argument an object3D object. Executes the callback on this object and all descendants
 - updateMatrix () : Updates local transform
 - updateMatrixWorld (`force`) : Updates global transform of the object and its children.
 - clone () : Creates a new clone of this object and all descendants.
 - **getObjectByName (`name, recursive`)** : Searches through the object's children and returns the first with a matching name, optionally recursive.
 - **getObjectById (`id, recursive`)** : Searches through the object's children and returns the first with a matching id, optionally recursive
 - translateOnAxis (axis, distance) : Translate an object by distance along an axis in object space. The axis is assumed to be normalized.
 - rotateOnAxis (axis, angle) : Rotate an object along an axis in object space. The axis is assumed to be normalized.

ThreeJS : graphe de scène

```
<script>
```

```
...
```

```
// cube
var cube = new THREE.Mesh(
    new THREE.BoxGeometry( 10, 10, 10 ),
    new THREE.MeshBasicMaterial( { color: 0x0000ff } )
);
scene.add( cube );

// sphere
var sphere = new THREE.Mesh(
    new THREE.SphereGeometry( 5, 32, 32 ),
    new THREE.MeshBasicMaterial( {color: 0xff0000} )
);
sphere.position.x = 20;

// sphere: child of cube
cube.add( sphere );
```

```
...
```

```
</script>
```

Rendu

Modèle d'illumination

Texture

Shader

Rendu

- Modèle d'apparence
 - La géométrie : nécessaire mais pas suffisant
 - Apparence
 - Couleur, matière et lumière
 - BRDF (Bidirectional Reflectance Distribution Function)
 - Source de lumière
 - Ombre et ombrage



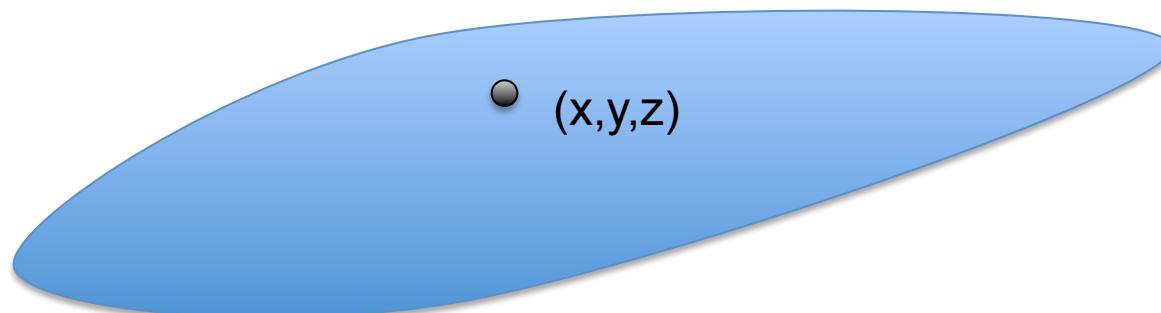
Rendu : Éclairage direct



Lampe

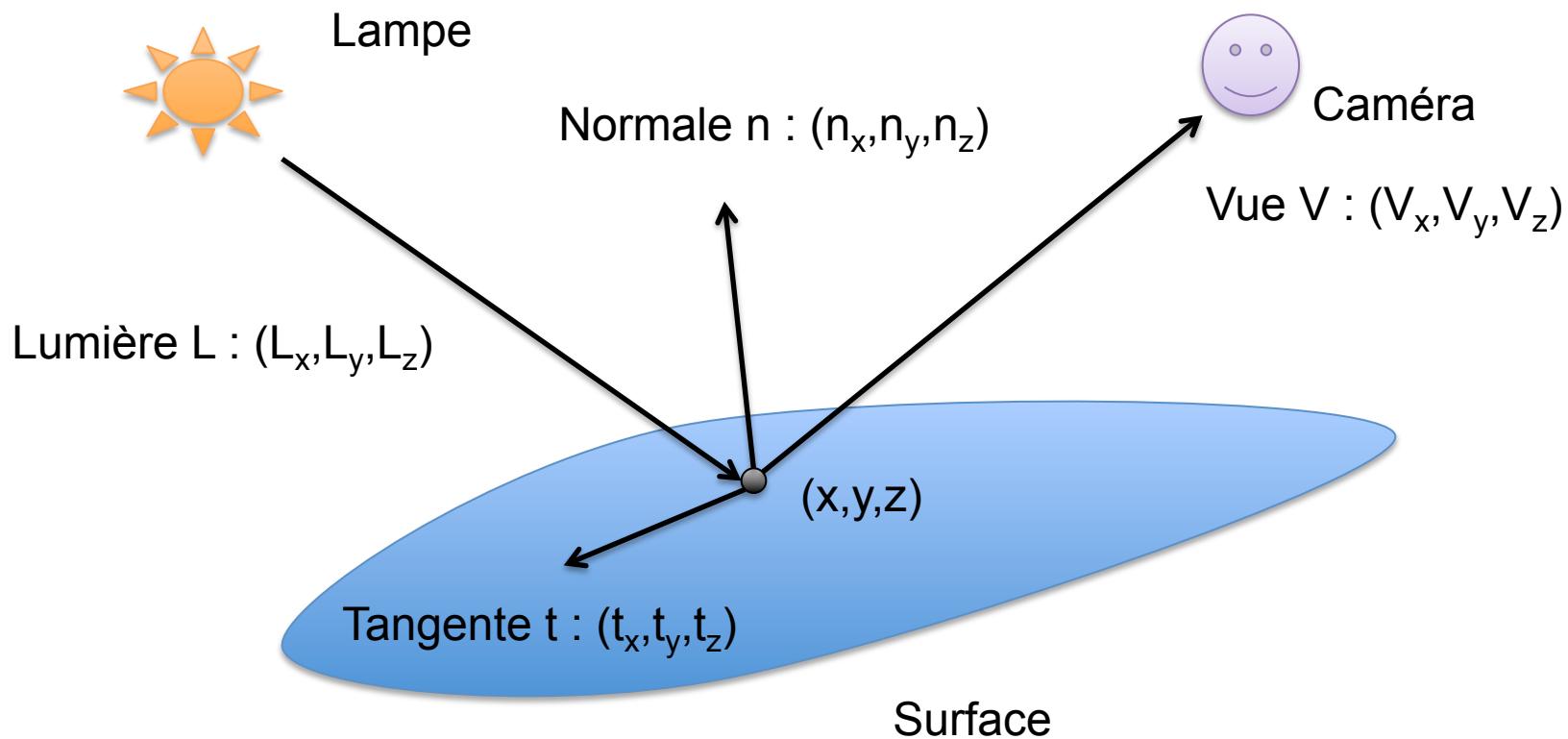


Caméra

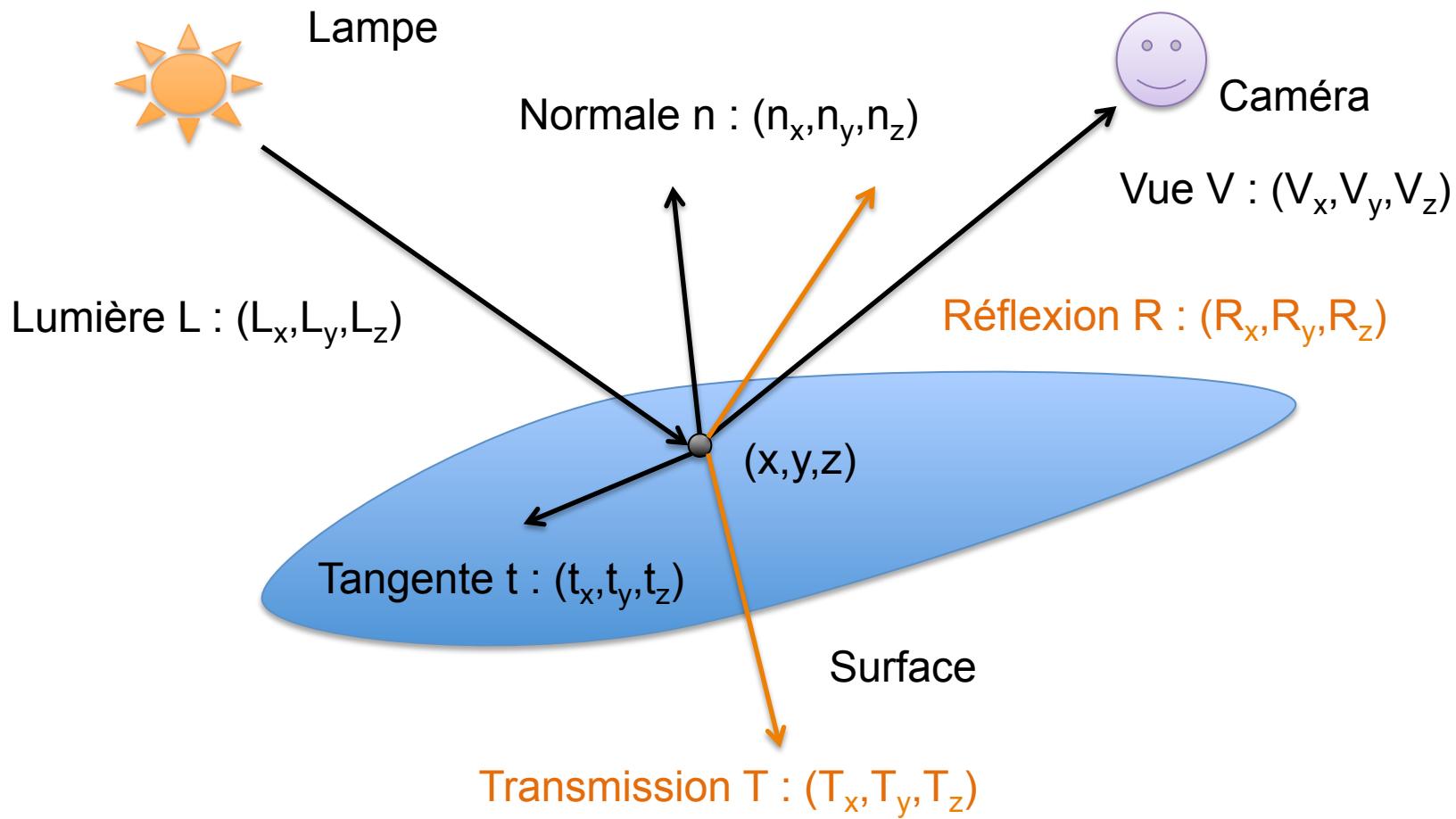


Surface

Rendu : Éclairage direct



Rendu : Éclairage direct



Rendu : Éclairage direct

- Quelles phénomènes modéliser ?
 - Matériaux :
 - Diffus, « brillant » (spéculaire)
 - Modèle de Phong

$$\left\{ \begin{array}{l} I = I_a + I_d + I_s \\ I_d = K_d I_{obj} I_{source} \cos(\text{ang}(n, L)) / d^2 \\ I_s = K_s I_{obj} I_{source} \cos^m(\text{ang}(R, V)) / d^2 \end{array} \right.$$

Rendu : ThreeJS

- Matériaux
 - Diffus : MeshLambertMaterial
 - Spéculaire : MeshPhongMaterial

```
// Lambert Material

sphereLambert = new THREE.Mesh(
  new THREE.SphereGeometry( 10, 32, 32 ),
  new THREE.MeshLambertMaterial(
    {
      color: 0xff40ff
    }
);
```

```
// Phong Material

spherePhong = new THREE.Mesh(
  new THREE.SphereGeometry( 10, 32, 32 ),
  new THREE.MeshPhongMaterial(
    {
      color: 0x40ffff,
      specular: 0xfffffff,
      shininess: 50
    }
);
```

Rendu : ThreeJS

- Source de lumières
 - PointLight, SpotLight
 - DirectionnalLight

```
// DirectionnalLight

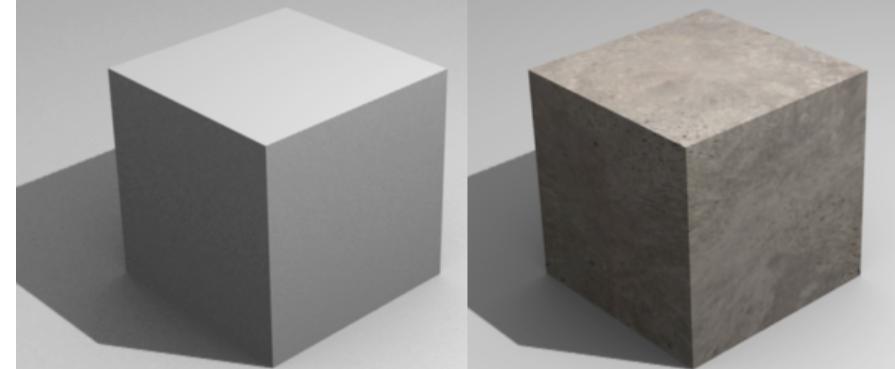
var directionalLight = new THREE.DirectionalLight( 0xffffff, 0.5 );
directionalLight.position.set( 0, 1, 0 ).normalize();
// add the light to the scene
scene.add( directionalLight );
```

```
// PointLight

var light = new THREE.PointLight( 0xff0000, 1, 100 );
light.position.set( 50, 50, 50 );
// add the light to the scene
scene.add( light );
```

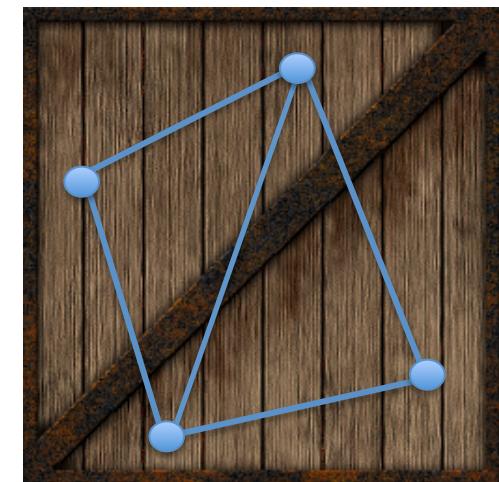
Rendu : Textures

- Définition du type de matériau ne suffit pas
- Plaquer une image sur une surface
 - image numérisée ou fabriquée ou calculée
 - chaque élément de l' image est appelé « texel »
- Texture
 - Image rectangulaire
 - propre système de coordonnées
- Lors du le rendu d'un pixel
 - les texels sont utilisés pour modifier une ou plusieurs propriété d' un matériau
 - la couleur diffuse par exemple



Rendu : Textures

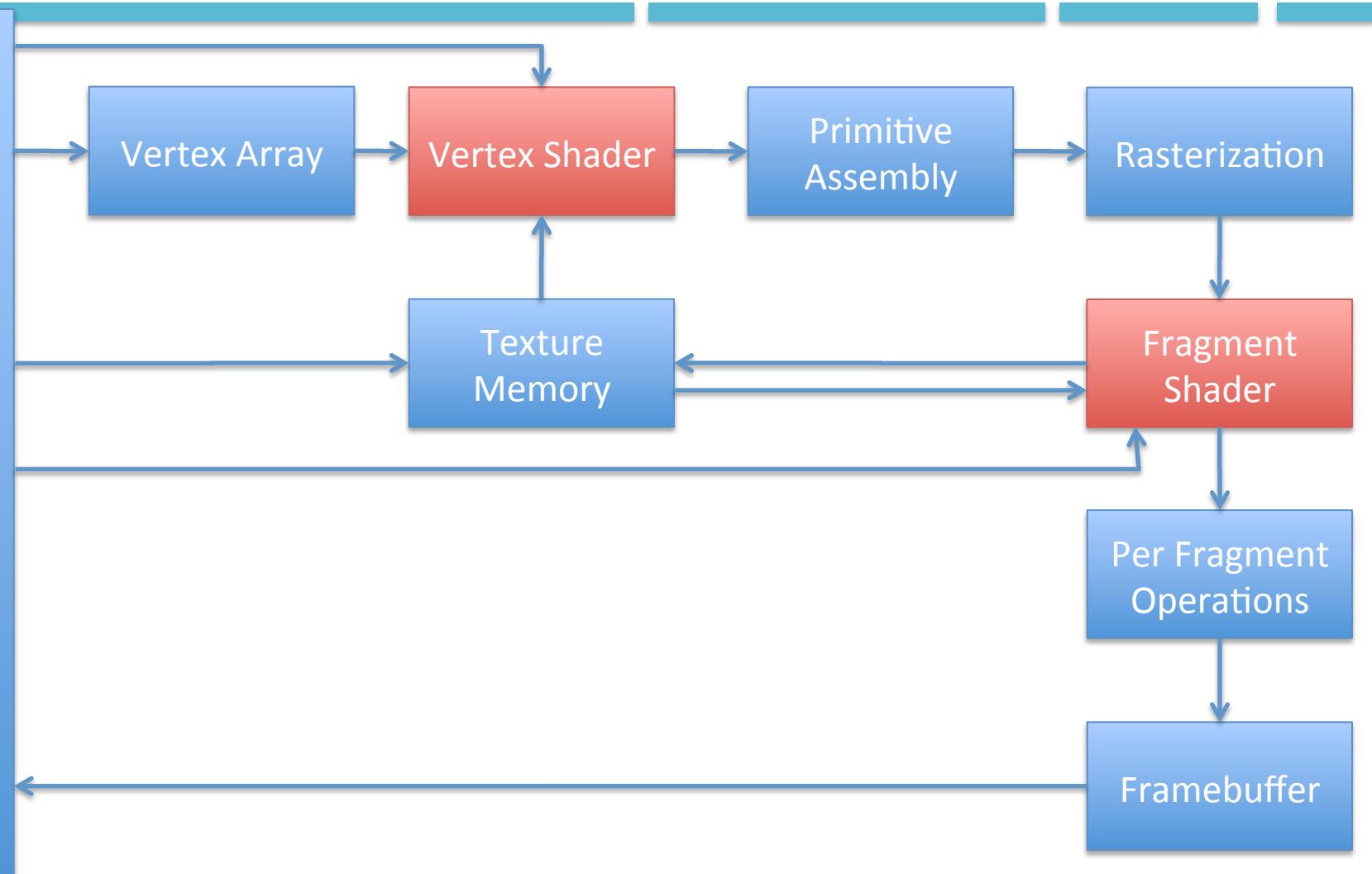
- Image plaquée sur une géométrie
 - Coordonnées de texture
 - Espace texture $[0,1] \times [0,1]$
 - **Coordonnées du sommet dans la texture**
 - Sommet (x, y, z)
 - coordonnées textures (u, v)
 - Normales (n_x, n_y, n_z)
 - Attention taille de l'image texture
 - taille 2^n



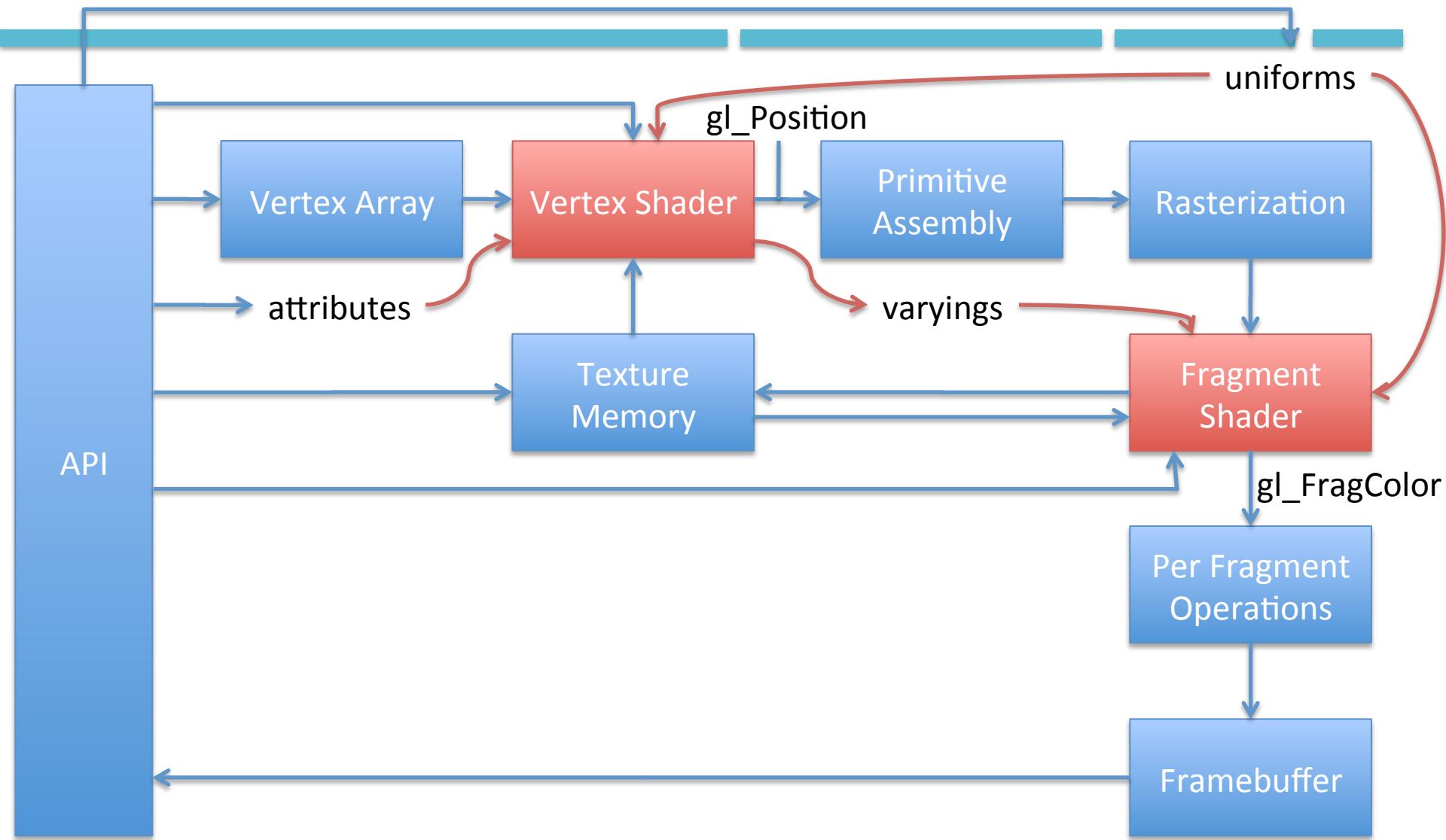
Rendu : Shader

- ThreeJS : matériau par défaut
 - Lambert, Spéculaire
- Autres matériaux
 - Programmation shader (`THREE.ShaderMaterial`)
- Principe
 - Entrées
 - Géométrie local : position, normale
 - Lampe : position, direction, couleur
 - Sortie
 - Couleur du pixel (fragment)
 - Code du shader : GLSL

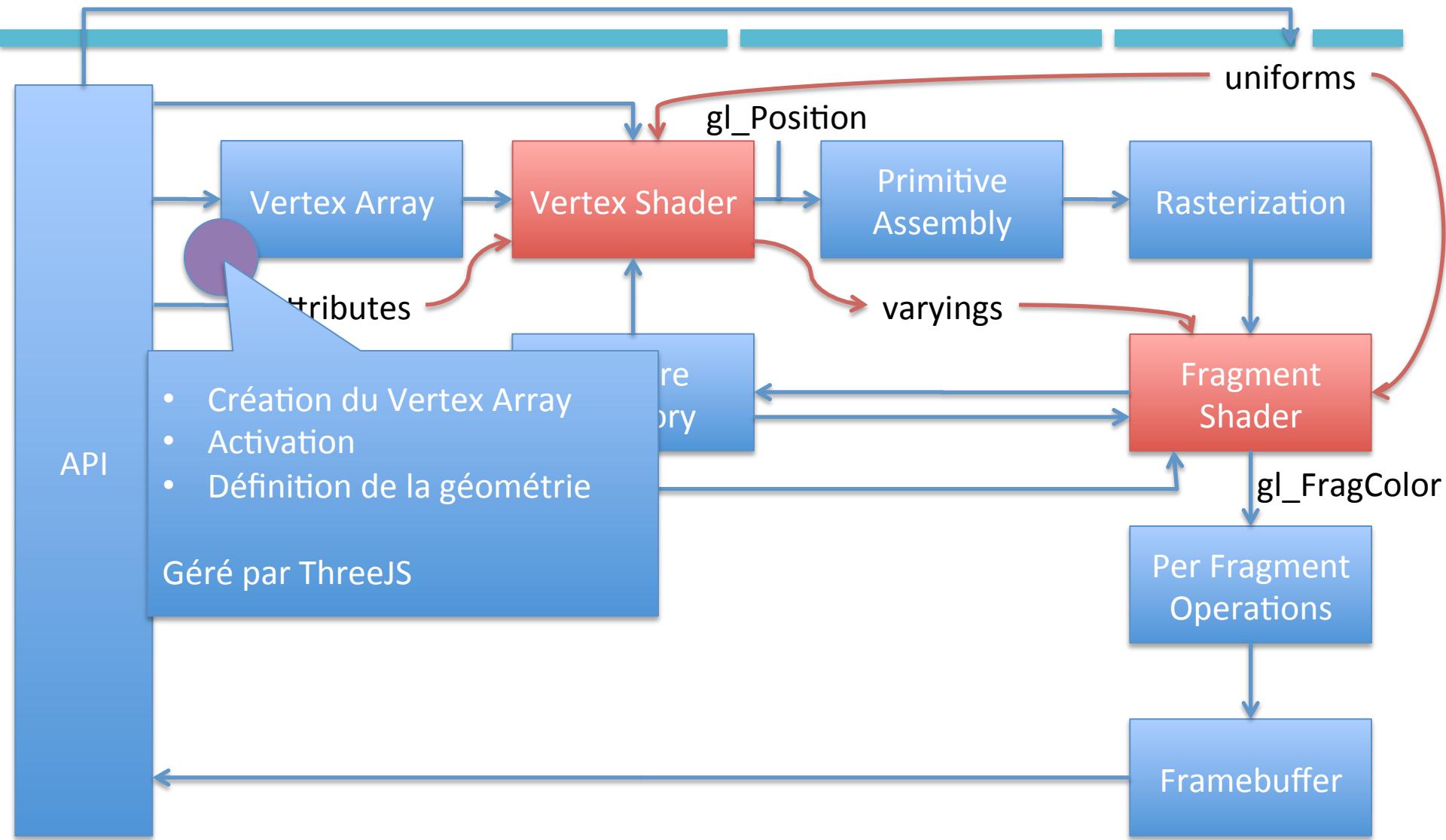
Rendu : Shader



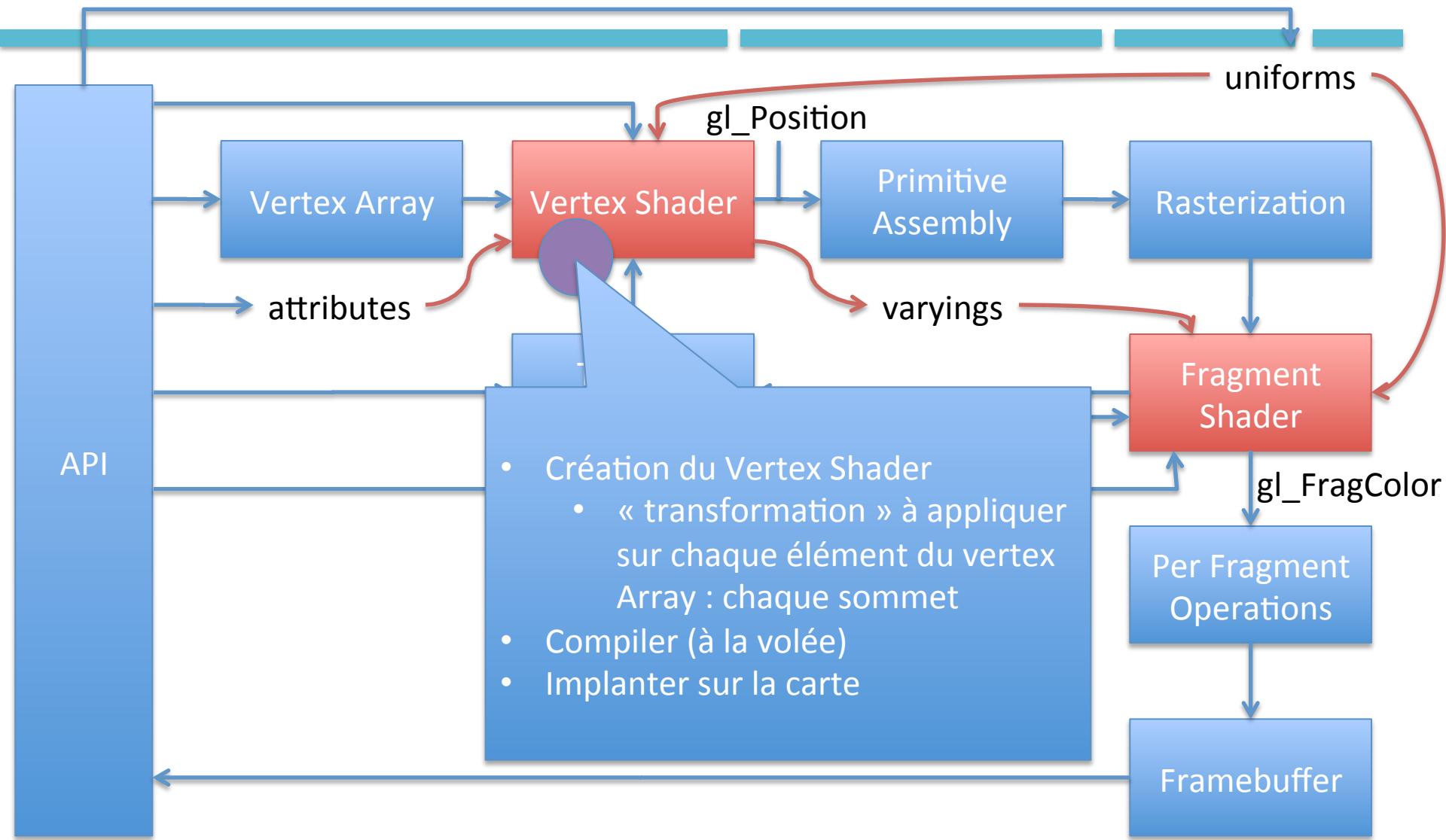
Rendu : Shader



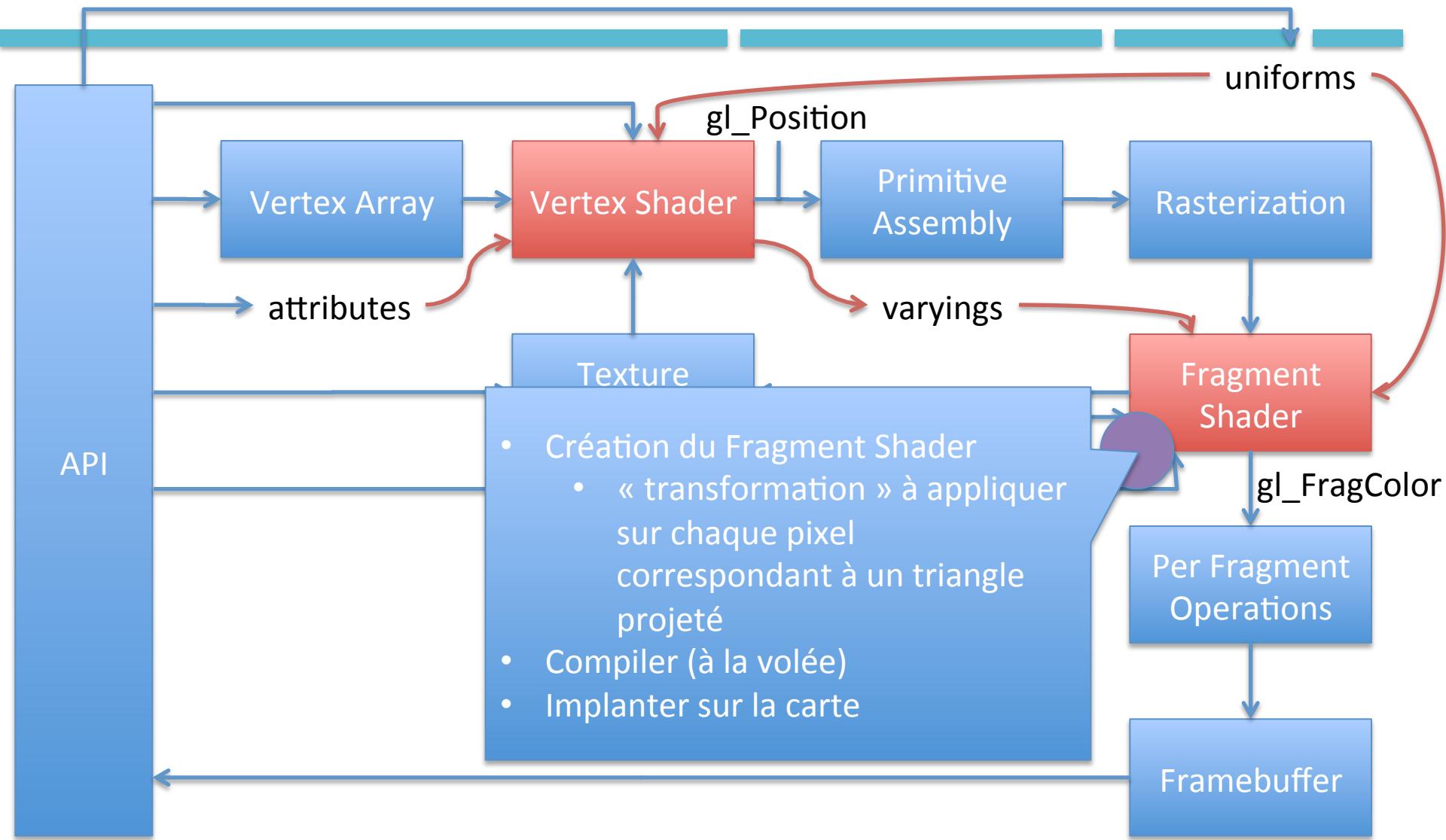
Rendu : Shader



Rendu : Shader



Rendu : Shader



ThreeJS : vertex shader

```
<script id="vertexShaderMetal" type="x-shader/x-vertex">
    // metal per fragment -vertex shader
    //-----
    // built in uniform and attributes
    // vec3 position, normal;
    // mat4 projectionMatrix, mat4 modelViewMatrix, normalMatrix;

    //-----
    // custom varying
    // normals
    varying vec3 vNormal;

    // position dans le repère de la caméra
    varying vec4 vPosition;

    // main
    void main() {
        vNormal = normalMatrix * normalize(vec3(normal));
        vPosition = modelViewMatrix * vec4(position, 1.0);
        gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    }
</script>
```

ThreeJS : fragment shader

```
<script id="fragmentShaderMetal" type="x-shader/x-fragment">
    uniform vec3 color; uniform float kd, sh ;                      //uniforms
    varying vec3 vNormal; varying vec4 vPosition;                     // varying

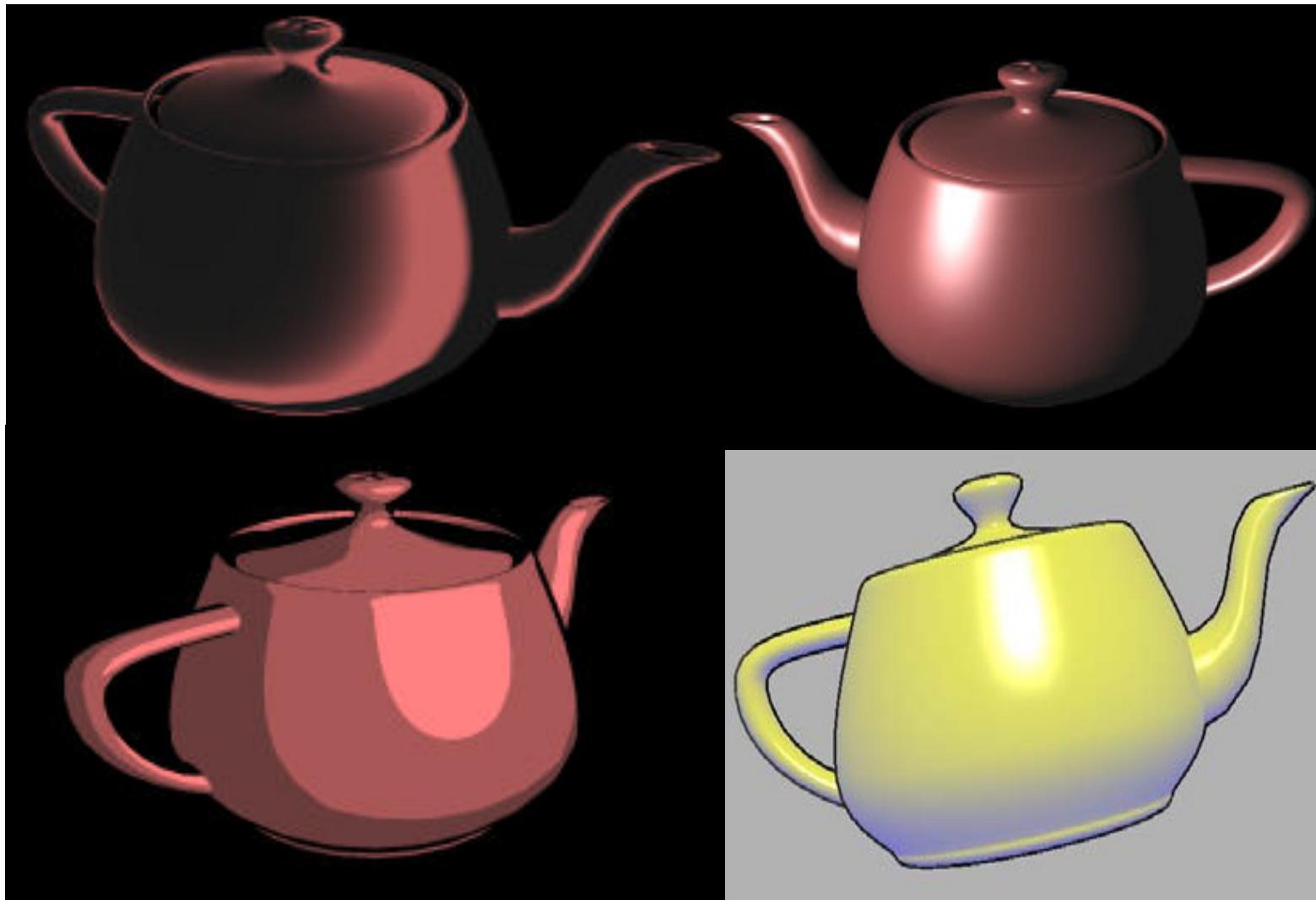
    void main(void) {                                                 // main
        float ks = 1.0 - kd;                                         // material spec
        // ambient, lights color1, light1 direction
        vec3 amb = vec3(0.0,0.0,0.25); vec3 lightC1 = vec3(1.0,1.0,0.5);
        vec3 lightDirection1 = normalize(vec3(-1.0, 1.0, 1.0));
        vec3 normal = normalize(vNormal);                             // normalize normal
        vec3 eyeDirection = normalize(-vPosition.xyz);               // eye direction
        vec3 reflectionDirection1 =
            normalize(2.0*normal*dot(normal,lightDirection1)-(lightDirection1));
        float diffu1 = max(dot(normal, lightDirection1), 0.0); // diffus
        // speculars
        float spec1 = pow(max(dot(reflectionDirection1, eyeDirection), 0.0), sh);
        // metals
        float metal1 = pow(max(dot(reflectionDirection1, eyeDirection), 0.0), sh*1.5);
        vec3 l1 = kd*lightC1 *color * diffu1 +                      // sum of terms
            2.0*ks*lightC1 *clamp(5.0*(spec1-metal1),0.0,1.0);
        vec3 L = l1 + amb*color;
        gl_FragColor = vec4(L,1.0);
    }
</script>
```

ThreeJS : Shader

```
//my shader
var metalShader ;

...
metalShader = new THREE.ShaderMaterial( {
    uniforms: {
        color: { type: "v3", value: new THREE.Vector3(1.0,1.0,1.0) },
        kd: { type: "f", value: 0.33 },
        sh: { type: "f", value: 50.0 }
    },
    vertexShader: document.getElementById( 'vertexShaderMetal' ).textContent,
    fragmentShader: document.getElementById( 'fragmentShaderMetal' ).textContent
} );
...
// cube2
cube2 = new THREE.Mesh(
    new THREE.BoxGeometry( 20, 20, 20 ),
    metalShader
);
```

Rendu : Shader



Rendu

- Améliorer la qualité de l'éclairage
 - Illumination globale
 - Milieux participatifs
- Rendu non photoréaliste (NPR)

