Rendu 3D – Rappel et compléments

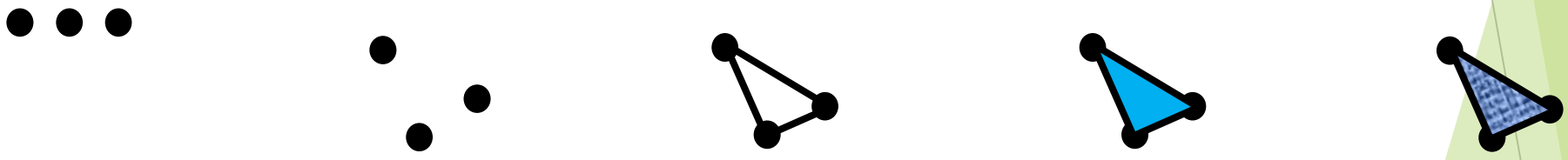
David Murray

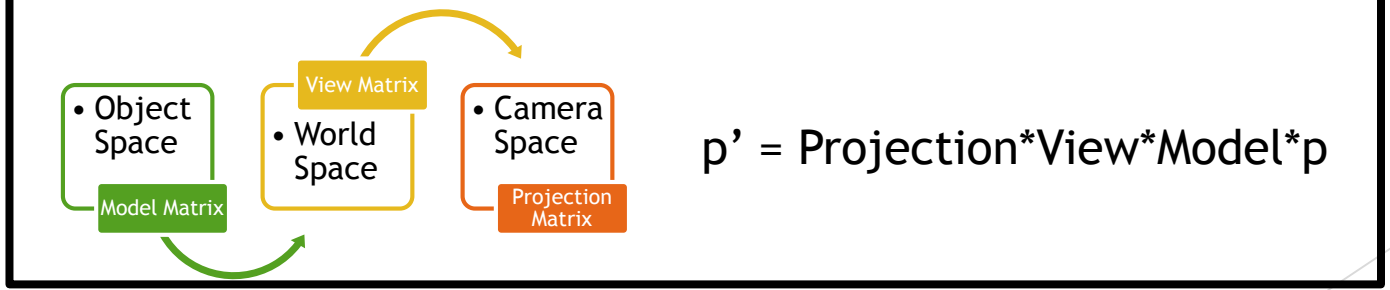
Source: Gael Guennebaud, Pierre Bénard

Pipeline graphique

+ attributs

Input → Vertex Shader (programmable) → Assemblage des primitives → Rastérisation des primitives → Fragment Shader (programmable) → Output (frame buffer)





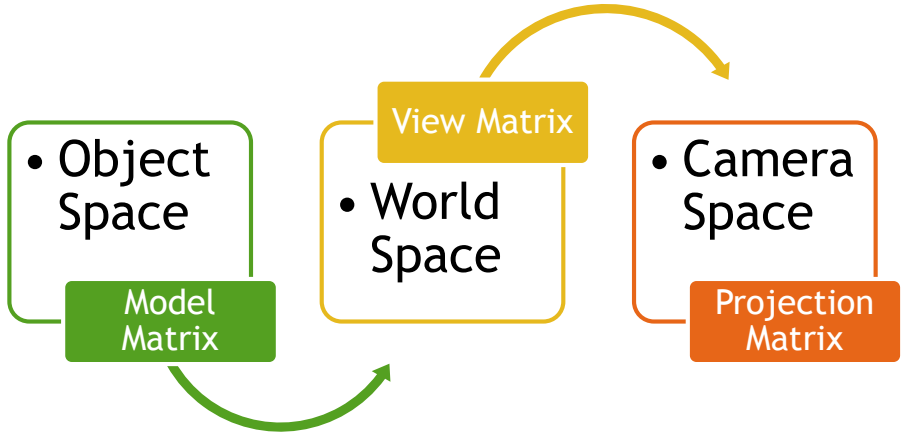
Les inputs

- 1 objet = 1 Vertex Array Object (VAO)
- 1 VAO = n Vertex Buffer Object (VBO)
  - 1 VBO = un ensemble d'information du même type
  - Type d'information:
    - Position des sommets
    - Couleurs des sommets
    - Propriété de réflexion des sommets
    - …
- 1 VAO = 1 draw call -> envoi des vertex au GPU -> pipeline graphique

Le vertex shaders

- Un vertex = un appel au Vertex Shader
- gl_Position
  - Un vec4 permettant de spécifier la position finale d'un sommet

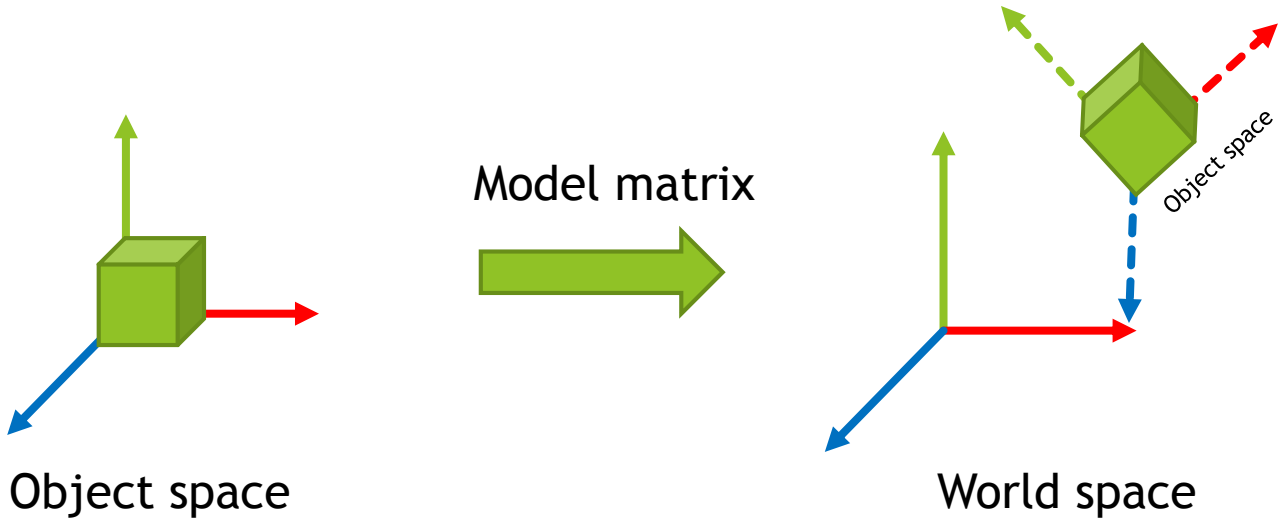
- Le vertex shader place correctement chaque vertex dans la scene selon:



p' = Projection*View*Model*p

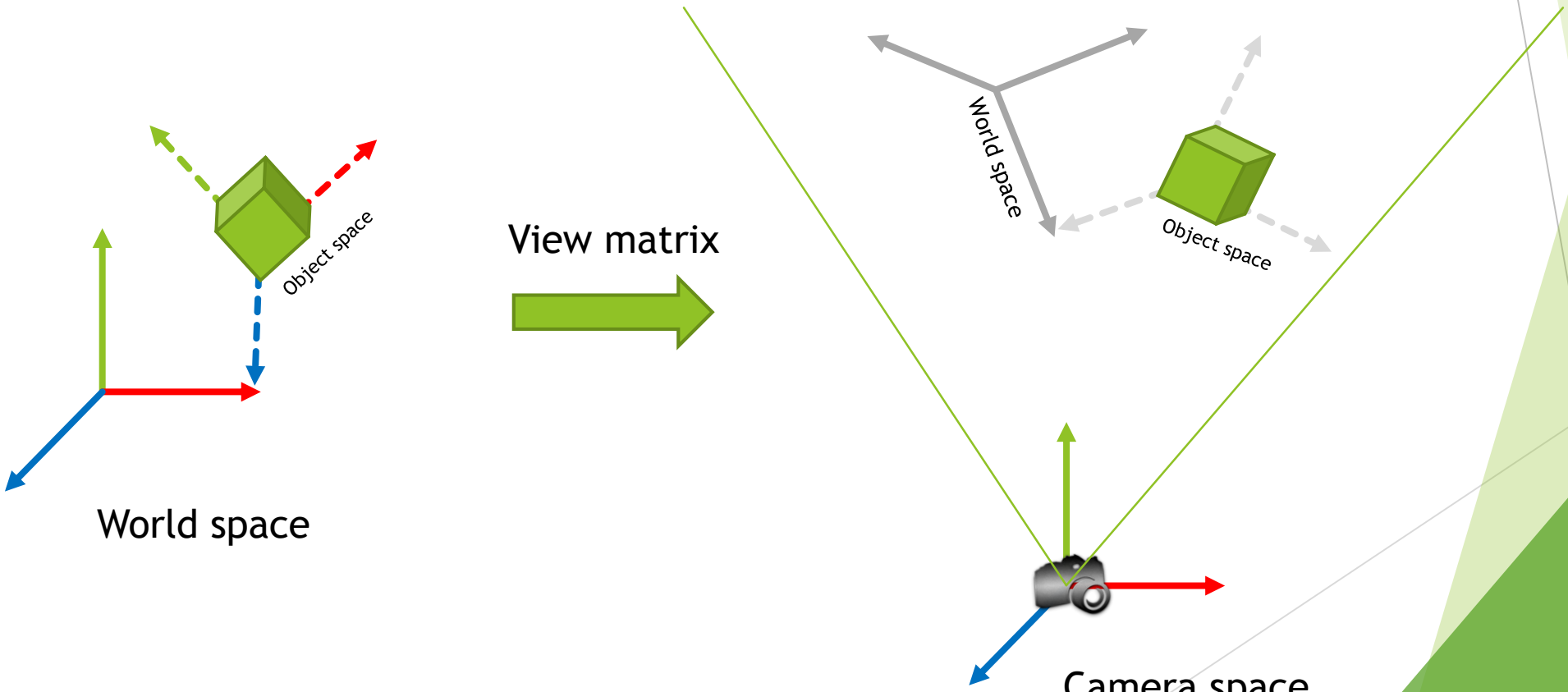
Bien placer un vertex en 3 étapes

- Etape 1: Passer en espace monde -> le repère globale de la scène



Bien placer un vertex en 3 étapes

- Etape 2: Passer en espace caméra -> Repère centré sur la caméra

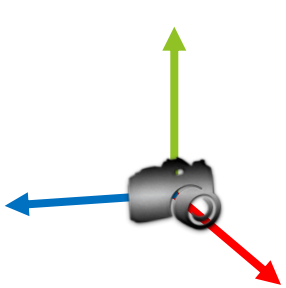


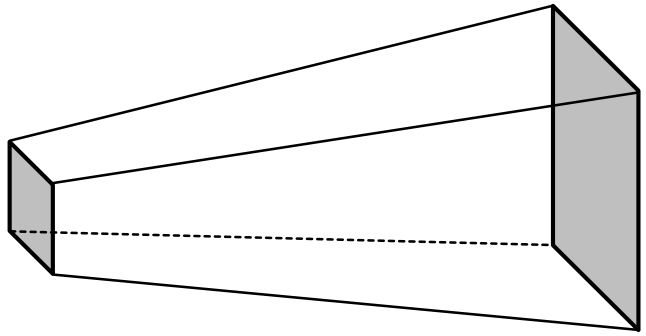
Etape 2: View Matrix

- Rappel: la view matrix est l'inverse de la « model » de la caméra
- Pour la calculer directement : la fonction « lookAt »
- Camera::lookAt(position, target, upVector)
  - position: position de la caméra DANS LE REPERE MONDE
  - target: point de visée DANS LE REPERE MONDE
  - upVector: vecteur indiquant l'orientation de la caméra

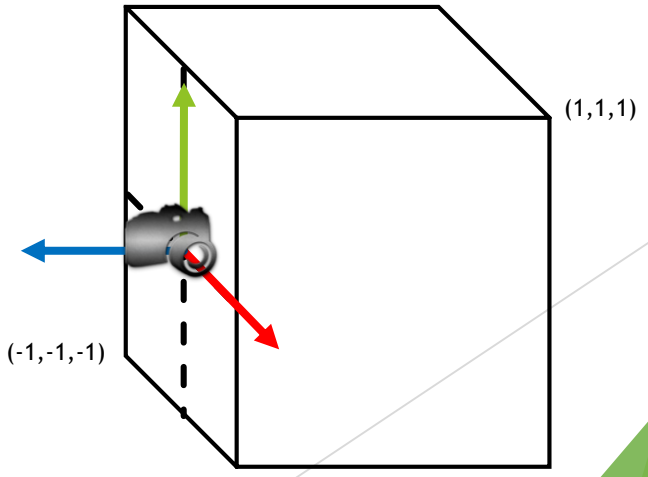
Bien placer un vertex en 3 étapes

- Etape 3: Projection sur le plan de la caméra





Projection matrix



Etape 3: Projection Matrix Perspective

- Pour la calculer : la fonction « perspective »

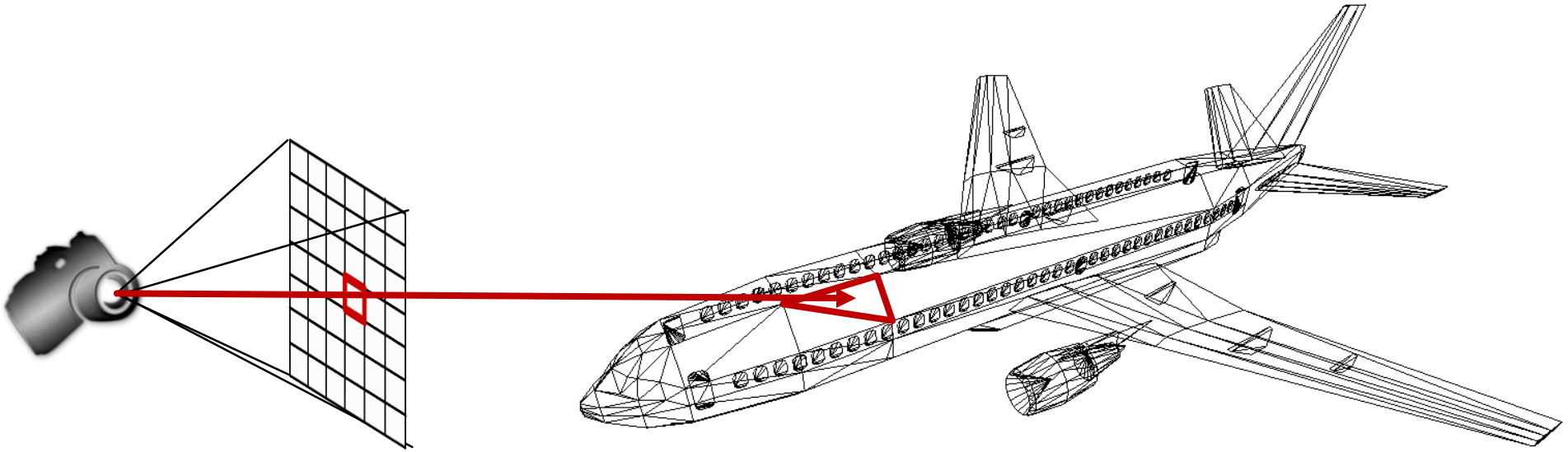
- Camera::perspective(FOV, aspect, nearPlane, farPlane)
  - FOV: Field of View, angle de vue en radians
  - aspect: « aspect ratio », le rapport entre les dimensions de votre fenêtre
    - Pensez aux format 16/9 ou 4/3 ! -> width/height
  - nearPlane: plan de plus proche projection
  - farPlane: plan de projection la plus éloignée
    - Attention: le near et le far sont des distances relative à caméra !
    - En dehors de la zone de projection: CLIPPING

Pour un support plus complet sur les matrices

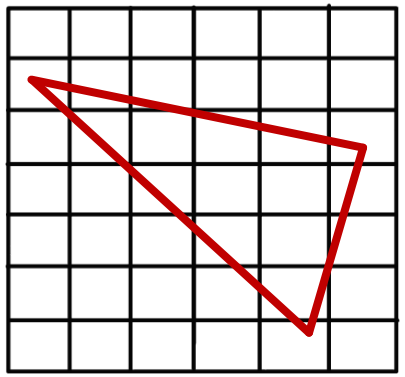
- http://www.opengl-tutorial.org/fr/beginners-tutorials/tutorial-3-matrices/

- Attention, les exemples de codes utilisent la librairie GLM et non pas EIGEN

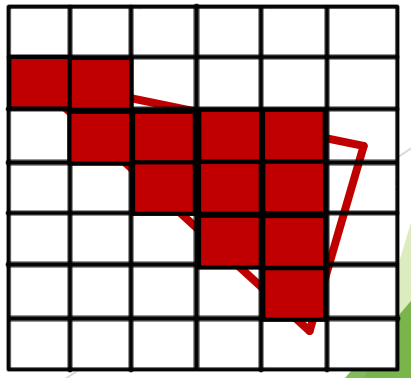
Rappel: la rastérisation



Projection des primitives sur la matrice de pixel

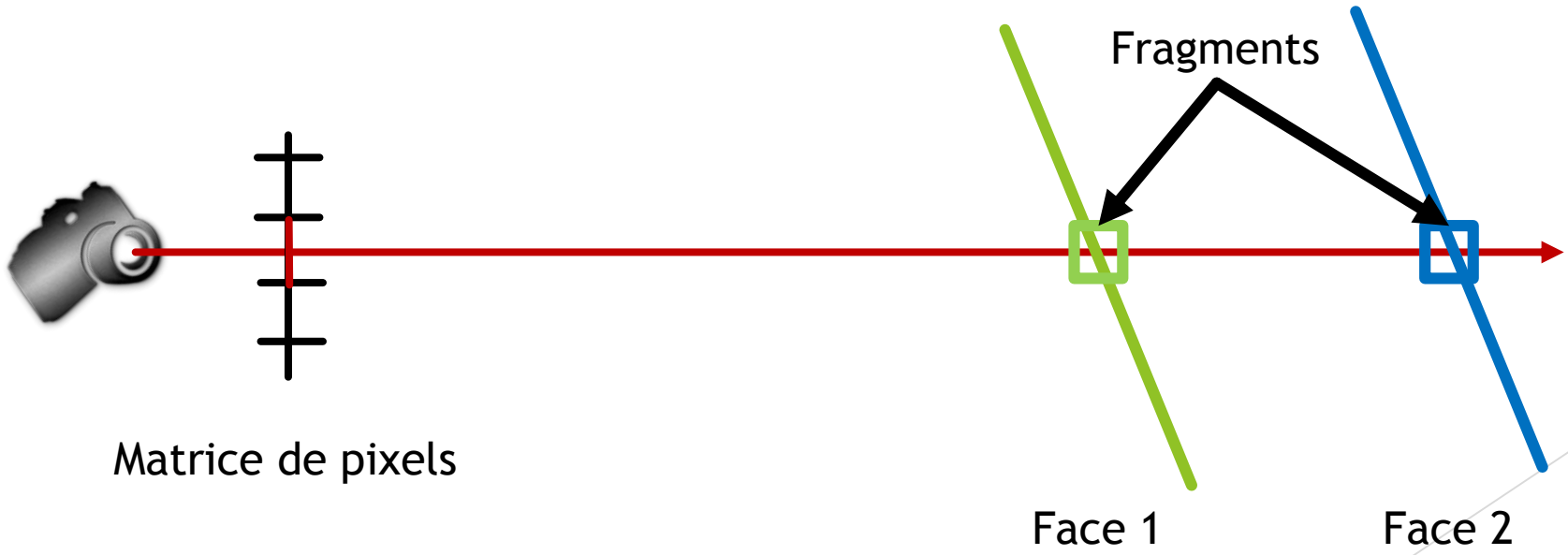


Détermination de la couverture de chaque pixel



Rastérisation: génération des fragments

- Un pixel couvert par une face = 1 fragment
- Un pixel couvert par n faces = n fragments



Fragment Shader ?

- Un fragment = un appel au Fragment Shader
- out vec4 XXX (« outColor » ou « toto » ou « peuImporte »…)
  - Un vec4 permettant de spécifier la couleur finale d'un sommet
- Pourquoi rendre cette étape programmable ?
  - Prochain TD !

Shader : in/out et variable uniform

- Mots clefs in/out: Permet d'accéder à des données présentes dans le pipeline
  - Le vertex shader accède aux attributs des sommets avec :
    - in vec3 vtx_position/vtx_color
    - Ces données sont déjà sur le GPU (c'est le VAO !)
    - Le mot clef « in » permet de spécifier au shader qu'elles existent déjà sous la forme « vec3 » et qu'elles sont stocker à un emplacement mémoire « vtx_position »/ « vtx_color »
    - Attention: vtx_XXX n'est PAS une variable OpenGL -> « mesSuperSommets » aurait fonctionné en faisant les appels adéquats
  - Le vertex shader peut renvoyer des données dans le pipeline avec « out »
    - Même logique que le « in »

Shader : in/out et variable uniform

- Mots clefs in/out: Pourquoi des « out » dans le vertex shader ?
  - Pour les fragments !
  - Permet de passer des informations depuis le vertex shader directement vers le fragment shader
  - « out vec3 v_color » dans le .vert -> « in vec3 v_color » dans le .frag

- Variable uniform: même logique
  - Mot clef « uniform » : uniform vec3 toto
  - getUniformLocation(« leNomDeLUniform ») -> addresse mémoire
  - glUniformXX(adresse, blabla, laValeurDeToto)