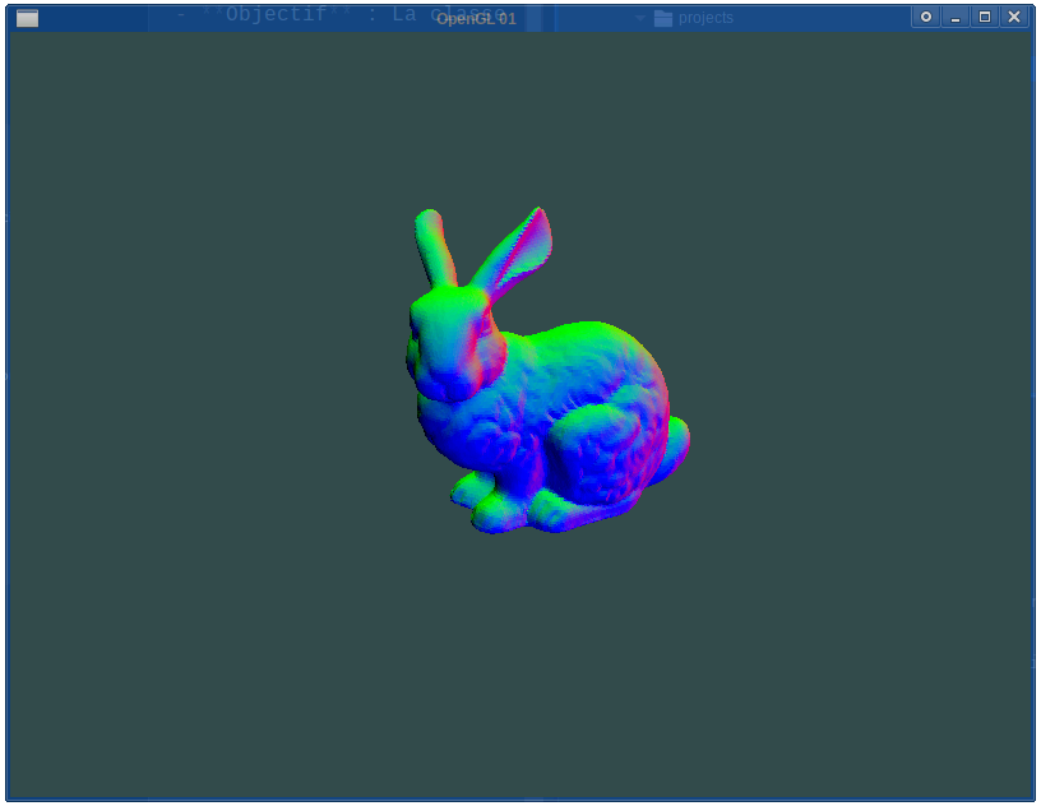
TP1 Maillage et Camera



Jehan-Antoine Vayssade

Prise en main

- **Donnée** : copier les dossiers, `w64devkit` et `opengl-cpp-template`
- **Compilateur** : w64devkit -> portable
- **Terminal** : lancé `w64devkit.exe`
- **Compilation** : `cd dossier_tp ; make -j 4`
- **Éxécutable** : `bin/mygame.exe`
- **Debug** : `gdb bin/mygame.exe`

Le plus simple

- Étape intermédiaire : Afficher un cube
- ObjReader -> Application -> GPUMesh -> Camera

Classe Application

- Fonctions clés :
  - Initialisation et configuration du contexte OpenGL.
  - Gestion des entrées utilisateur et des événements.
  - Mise à jour et rendu de la scène.
- A faire :
  - loadResources
    - charger le shader par défault
    - charger le maillage "stanford-bunny.obj"
  - renderLoop
    - binder le shader
    - mettre a jour les parametres
    - afficher le lapin

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

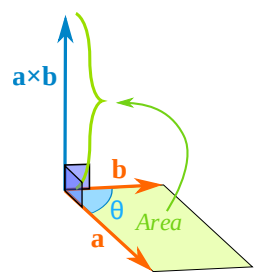
$$\mathbf{R_x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R_y} = \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R_z} = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

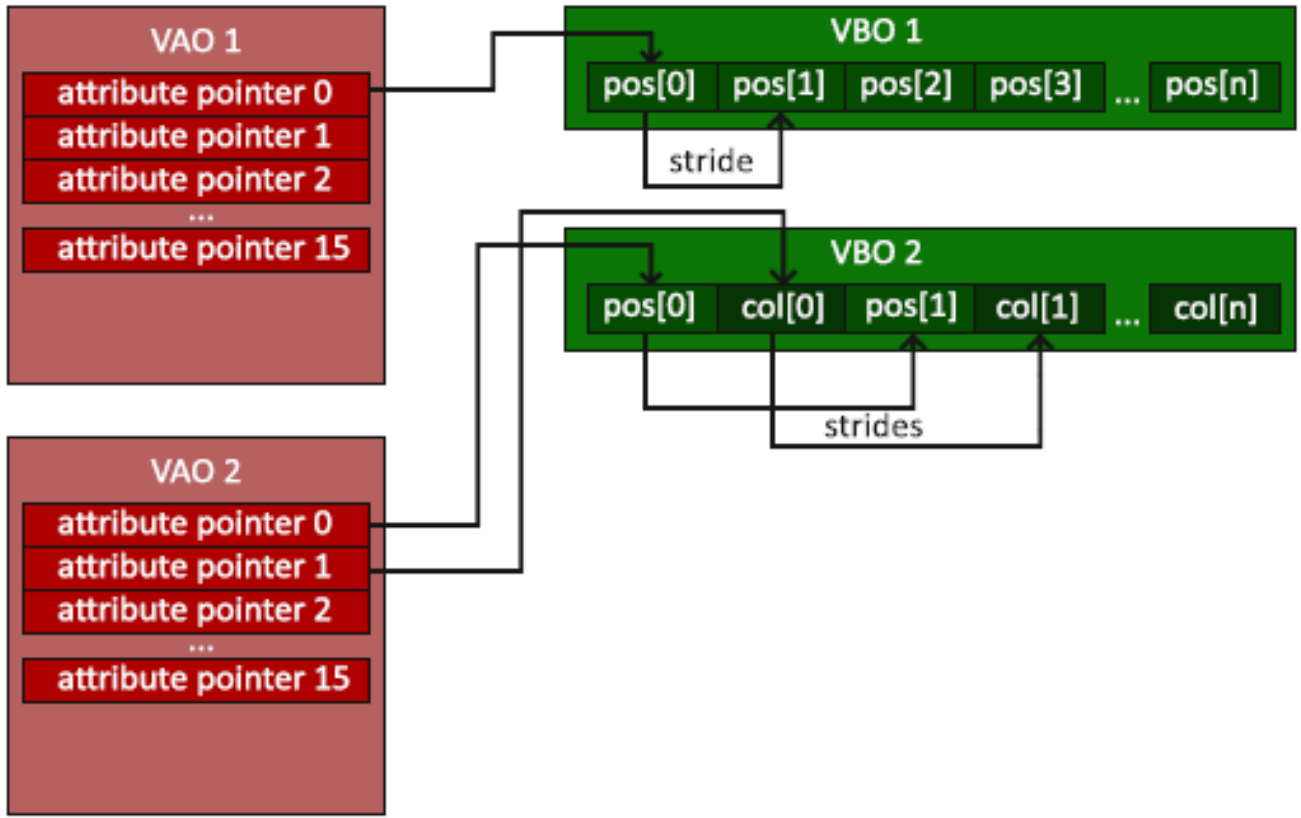
Classe ObjReader

- Fonctions clés :
  - Lire les sommets, les normales et les coordonnées de texture.
  - Lire les indices
  - Génération des faces (1 Triangle = 3 Vertices + 3 Indices)
- A faire :
  - Fusioner les sommets / normales / coordonnées vers Vertex (packed)
  - Regénérer les indices pour ces nouvelles faces
  - Le fichier Bunny n’a pas de normal, généré les ! (indice cross-product)



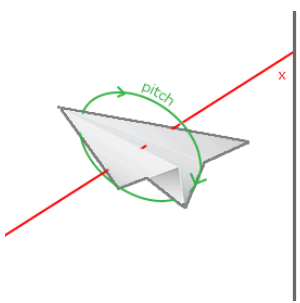
Classe GPUMesh

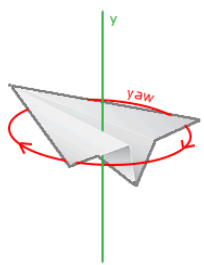
- Fonctions clés :
  - Chargement et stockage des données de maillage sur le GPU.
    - Position, Normales, Coordonnées
    - Ce que l'on veux, on décide dans le shader a quoi la donnée est utile
    - Couleur ? Interpolation automatique !
  - Gestion des états OpenGL, VAO / EBO / VBO + VertexAttrib
- A faire :
  - Crée les buffers `glGenVertexArrays + glGenBuffers`
  - Envoyé les données au GPU `glBufferData`
  - Dire comment accedé aux données
    - API => `glVertexAttribPointer(0, ... 3*sizeof(float), ...)`
    - Shader => `layout (location = 0) in vec3 aPos;`
    - Indice : `sizeof` + `offsetof`

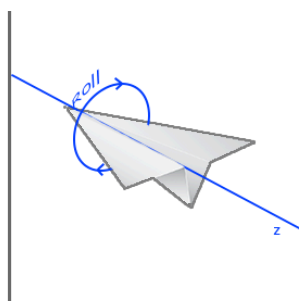

VAO 1
attribute pointer 0
attribute pointer 1
attribute pointer 2
...
attribute pointer 15
VAO 2
attribute pointer 0
attribute pointer 1
attribute pointer 2
...
attribute pointer 15
VBO 1
pos[0]
pos[1]
pos[2]
pos[3]
...
pos[n]
stride
VBO 2
pos[0]
col[0]
pos[1]
col[1]
...
col[n]
strides

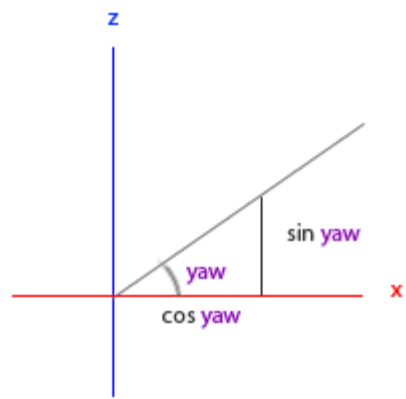
Classe Camera

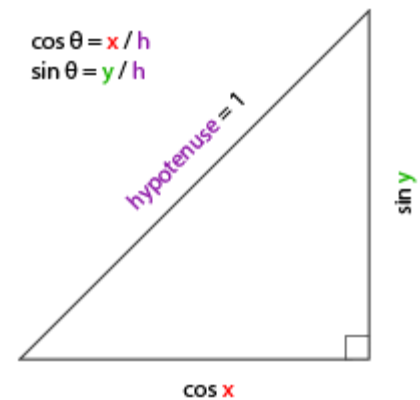
- Fonctions clés :
  - Définition et mise à jour de la position, et l'orientation.
- A faire :
  - updateCameraVectors (r, u, f) et getViewMatrix
  - indice trigonométrie -> utilisé Yaw et Pitch ainsi que des cos et des sin











$$\mathbf{View} = \begin{pmatrix} r_x & r_y & r_z & -e_x \\ u_x & u_y & u_z & -e_y \\ -f_x & -f_y & -f_z & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{P}_{\text{perspective}} = \begin{pmatrix} \frac{1}{\tan\left(\frac{fov}{2}\right) \cdot aspect} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{fov}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{far}{far - near} & \frac{-far \cdot near}{far - near} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Classe Shader

- **Fonctions clés :**
  - Chargement et compilation du code source des shaders.
  - Liaison des shaders dans un programme.
  - Définition des valeurs uniformes.
- **A faire :**
  - Modifier les positions en temps réel (vertex shader)

Chargement et Configuration des Maillages

- **Fonctions clés :**
  - Chargement des données de maillage à partir de fichiers via `ObjReader`.
  - Création d'un `GPUMesh` pour préparer les maillages pour le rendu
  - C'est ici que la va appeller `addVertexAttributePointers`

Bonne chance

- https://webglfundamentals.org/webgl/lessons/webgl-3d-camera.html
- https://antongerdelan.net/opengl/index.html#ebook
- https://www.enlightenment.org/develop/legacy/program_guide/evasgl_pg
- https://developer.apple.com/library/archive/documentation/GraphicsImaging/Conceptual/OpenGL-MacProgGuide/opengl_shaders/opengl_shaders.html
- https://opengl.developpez.com/tutoriels/apprendre-opengl/
- https://mini.gmshaders.com/p/vertex