# Sujets de TP - automne

*Il faudra modifier les fichiers ViewerEtudiant.h et ViewerEtudiant.cpp*

## 1. Manipulation des formes de base

**1) Définition d'un cube avec des GL_TRIANGLE_STRIP**

a. Relisez le code de la méthode **Viewer ::init_cube()** qui définit le maillage d'un cube.
b. Relisez le code de la méthode **Viewer::draw_cube()** qui permet l'affichage d'un cube.
c. Utilisez la méthode **draw_cube()** pour afficher des cubes à différents endroits de la scène, avec des cubes de tailles différentes.

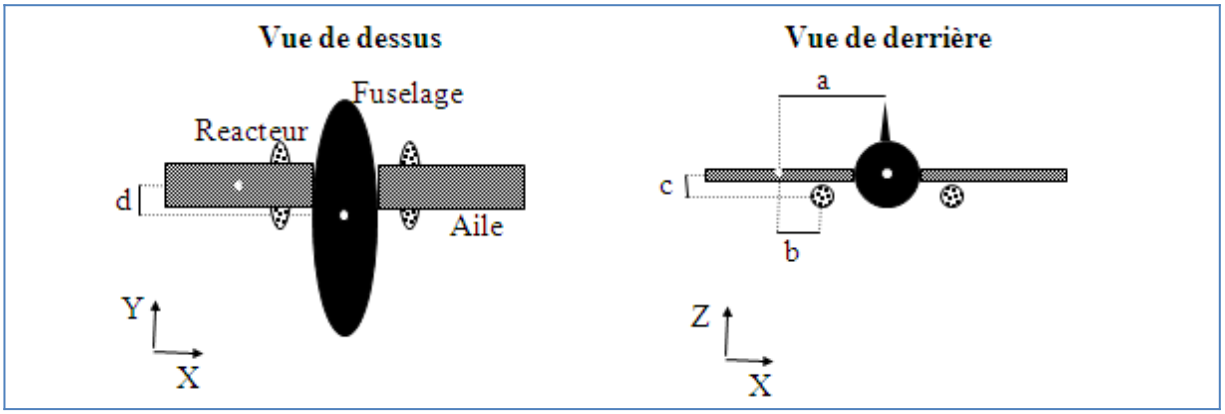
2) **Définition des formes de base : cylindre, cône, sphère.** Pour chacune des formes, il s'agira d'écrire les méthodes suivantes :

a. **ViewerEtudiant::init_forme()** permettant de créer le Mesh de la forme de base qui sera centrée à l'origine et de taille unitaire.
b. **ViewerEtudiant::draw_forme(const Transform & T)** permettant l'affichage de la forme de base en utilisant la transformation géométrique T mise en paramètre.

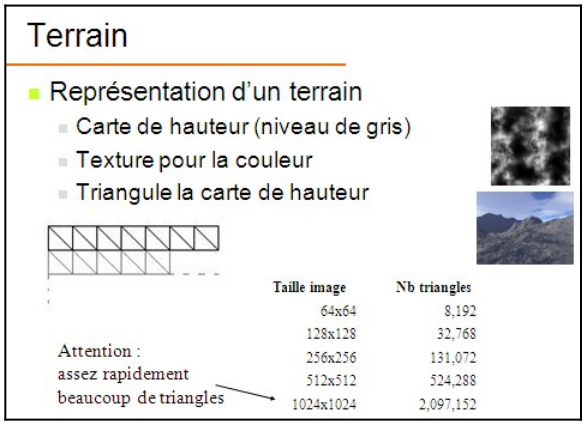
3) Ajoutez les **normales** et les **coordonnées textures** à ces formes de base. Testez à chaque fois le résultat de ces ajouts.

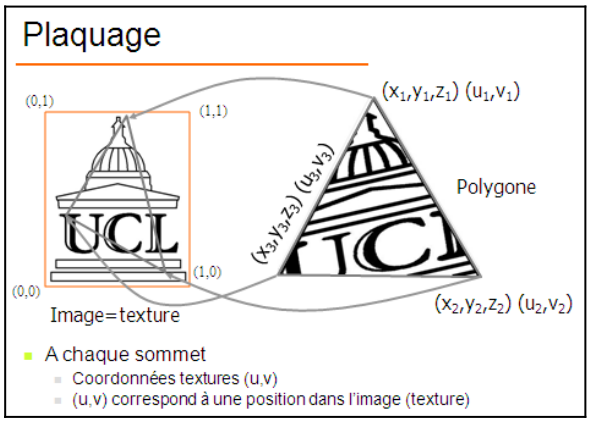
## 2. Affichage à l'aide de transformations géométriques

Vous disposez désormais de plusieurs formes de base (sphère, cône, cube, cylindre). A partir de ces formes, écrivez une procédure **draw_objet** permettant l'affichage d'un objet plus complexe (avion, voiture, personnage, animal, vélo, etc.). Cette procédure combinera les formes de base en utilisant des transformations géométriques. Ci-dessous un exemple pour un avion.



## 3. Terrain, texture, *billboard* (arbre) et *cubemap*




**1) Définition d'un terrain.**

a. Écrivez la méthode **ViewerEtudiant ::init_terrain()** qui permet de créer le Mesh d'un terrain basé sur des triangles. Les sommets du terrain seront définis à partir d'une image interprétée comme une carte de hauteur.

b. Effectuez l'**affichage de ce terrain** en effectuant les transformations géométriques adéquates pour ajuster sa taille et son positionnement dans la scène.

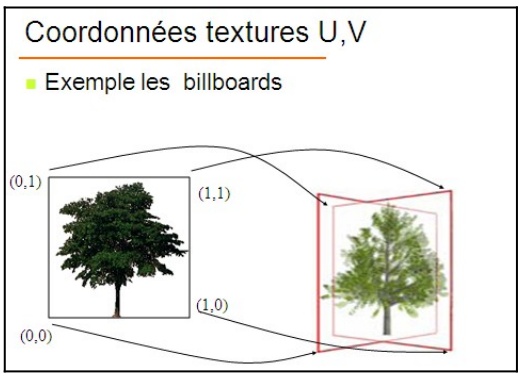
c. Ajouter le **calcul de normal** pour chaque sommet du maillage du terrain.

d. Ajouter les **coordonnées textures** pour chaque sommet du maillage du terrain.

**2) Ajout d'arbres sur le terrain.**

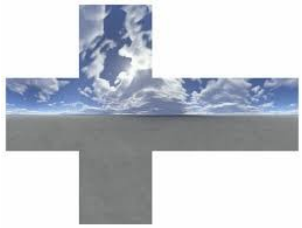
a. Écrivez la méthode **ViewerEtudiant ::init_tree()** permettant de définir le maillage qui servira de support à votre *billboard* représentant un arbre.

b. Écrivez la méthode **ViewerEtudiant ::draw_tree(const Transform &T)** affichant un arbre représenté par un *billboard* subissant la transformation géométrique T.

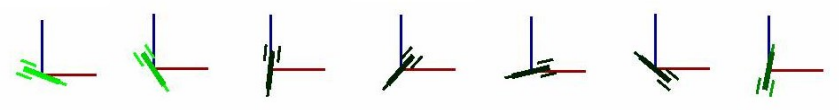



c. **Affichez un ensemble d'arbres sur le terrain** en utilisant la carte de hauteur pour les positionner correctement.

3) **Ajout d'un *cubemap autour du terrain*.**
   a. Écrivez la méthode **ViewerEtudiant ::init_cubemap()** permettant de créer le Mesh du *cubemap* en spécifiant les coordonnées de textures et de normales adéquates.
   b. **Affichez un *cubemap*** autour de votre terrain.

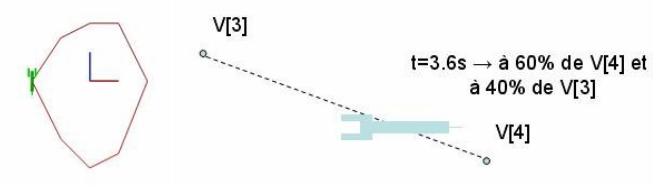


## 6. Animation



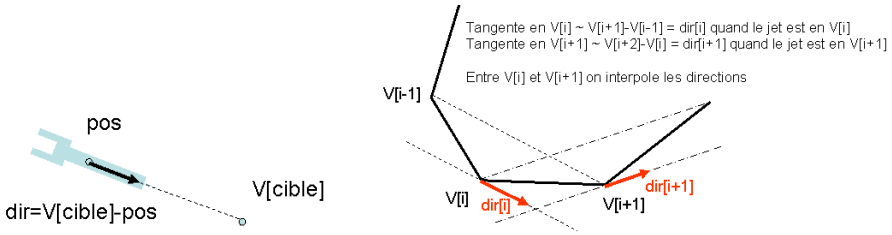
1) Écrivez le code permettant de **faire tourner un objet** sur lui-même en fonction du temps.

2) Nous pouvons définir un tableau de points (*Vec V[NB]*) qui correspond à la trajectoire que devrait suivre l'objet. Pour simplifier, l'animation se déroulera dans le plan X,Z.




Écrivez le code permettant de **placer un objet sur la trajectoire en fonction du temps** (ne pas considérer l'orientation de l'objet pour l'instant). Il faudrait qu'au *temps*=3.6, l'objet soit entre le point V[3] et le point V[4] (plus exactement à 60% du point V[4] et à 100-60=40% du point V[3]). Oui, c'est une interpolation linéaire !

3) Écrivez le code permettant d'**orienter correctement l'objet.** Nous connaissons la position de l'objet (pos), ainsi que sa direction de déplacement (dir=*V[cible]-pos*). Au repos, l'objet est aligné avec X=(1,0,0). Il faut donc trouver la matrice de passage faisant tourner l'objet vers sa direction. Nous avons donc que X → dir. Ici, Y ne change pas car les points V sont dans le plan X,Z. Pour trouver Z, nous pouvons faire *dir* $\wedge$ (0,1,0).




4) La direction de l'objet n'est pas continue. Pour rendre ses virages plus doux, nous pouvons également interpoler sa direction, ce qui revient en fait à calculer la tangente de la courbe. Écrivez le code correspondant.

## 7. Monde virtuel

Ajoutez différents éléments dans votre scène pour créer un monde virtuel plus complet : ajout de textures autres que celles initialement fournies, importation d'objets, ajout d'un lac, ajout d'objets animés en utilisant des textures animées, etc.

### Grille d'évaluation du TP

**Formes de base = 3 pts avec**

1 = forme de base : cône, sphère, cylindre

1 = normales ok sur toutes les formes de base

1 = texture ok sur toutes les formes de base

**Objet plus complexe = 3 pts selon la qualité**

**Terrain = 3 pts avec**

1 = terrain à la bonne taille

1 = terrain avec normale

1 = terrain avec texture

**Monde plus complexe = 3 pts avec**

1,5 = cubemap correct

1,5 = plusieurs billboards bien placés sur le terrain

**Animation = 3 pts avec**

1 = si animation le long de la courbe avec saut entre les points de contrôle

2 = si animation sur la courbe avec déplacement continu

3 = animation continu et direction prise en compte

**Monde virtuel = 3 pts**

Ajoutez différents élements pour obtenir une scène représentant un monde virtuel :

- ajout de textures (autres que celles initialement forunies)

- ajout d'objets en chargeant directement des maillages

- ajout d'un objet représentant un lac

- ajout d'objets animés (texture animée)

- ...

**Rapport = 2 pts**