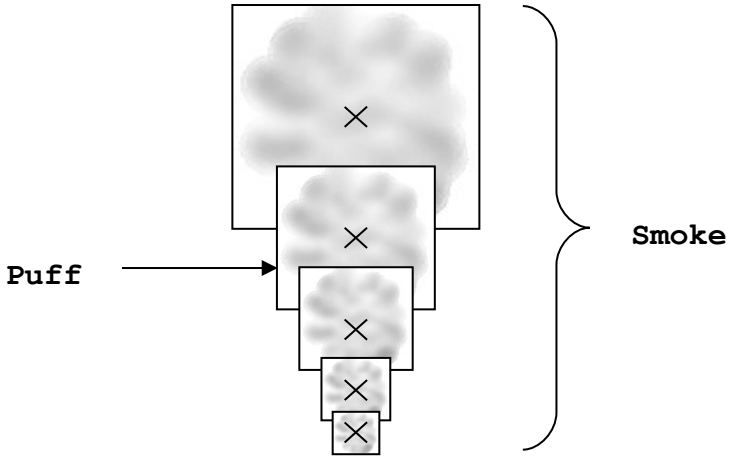
Aix-Marseille Université **2019 - 2020**
Master 2 GIG

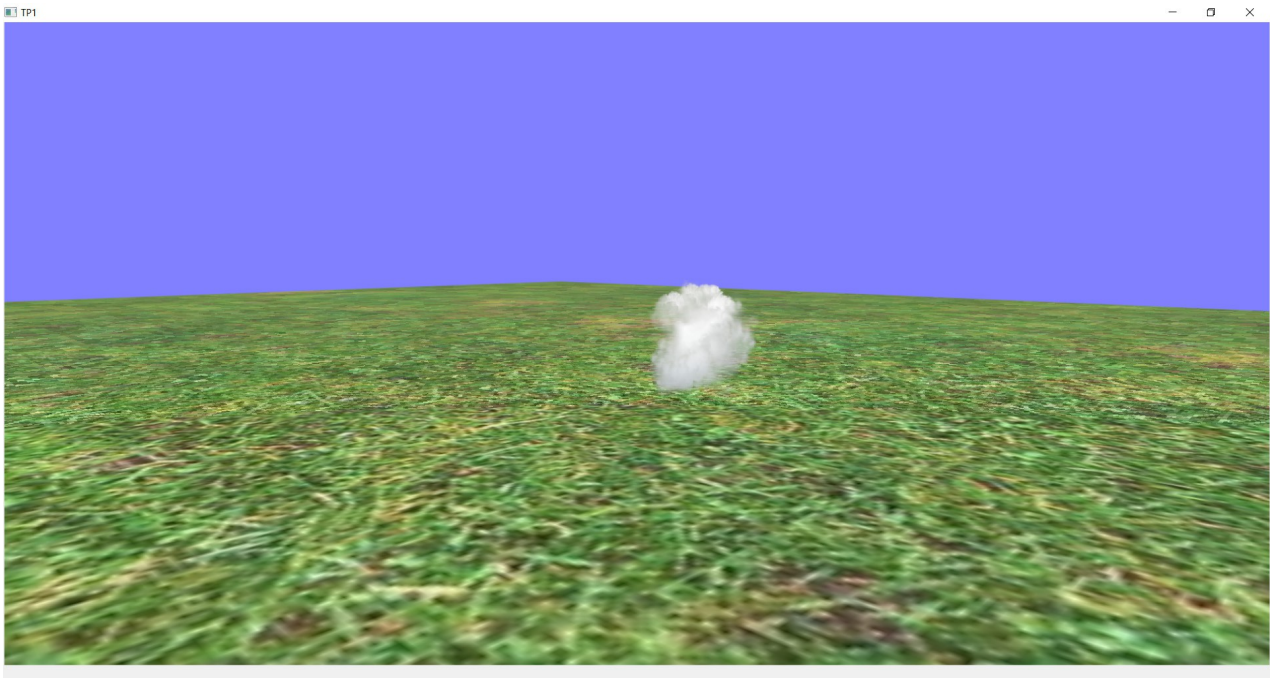
S51IN3B4 - Animation et rendu
TP 1

Animation de fumée par système de particules

On veut représenter une colonne de fumée par un système de particules où chaque particule correspondra à une bouffée de fumée. Les bouffées de fumée seront affichées au moyen de quadrilatères texturés affichés selon le principe des *billboards* (des quadrilatères faisant toujours face à la caméra).



Téléchargez sur Ametice le fichier **base_tp1.zip** qui contient une base de programme Qt affichant de la 3D dans une fenêtre. Compilez-le et exécutez-le. Vous devriez obtenir ceci :



Dans ce code de base, un sol est affiché sous la forme d'un quadrilatère texturé au moyen d'un VBO et d'un program shader (simple.vsh et simple.fsh).
A titre d'exemple pour la question 3, un billboard est affiché sous la forme d'un quadrilatère texturé au moyen d'un VBO et d'un program shader (billboard.vsh et billboard.fsh).
Vous pouvez utiliser la souris pour contrôler la caméra de la scène (bouton gauche : rotation, bouton du milieu : zoom , bouton droit : translation).

Classe `Puff`

1. Ecrire une classe **`Puff`** pour représenter une bouffée de fumée, qui sera affichée au moyen d'un billboard. Une bouffée est caractérisée par sa position (**`QVector3D`**), sa taille (**`float`**), son vecteur vitesse (**`QVector3D`**) et le temps qu'il lui reste à « vivre » en secondes (**`float`**).
   Ecrire le constructeur de cette classe, permettant d'initialiser ces données.

2. Ecrire une méthode **`animate()`** de la classe **`Puff`** qui recevra en paramètre le temps écoulé en secondes entre deux affichages :

   ```
   void Puff::animate(float dt);
   ```

   Elle permettra de modifier la position de la bouffée en fonction du temps écoulé et de sa vitesse :
   nouvelle position = position_courante + vitesse * dt
   On réduira aussi le temps de vie restant de la bouffée en lui soustrayant le temps écoulé.

3. Ecrire une méthode **`display()`** de la classe **`Puff`**, permettant d'afficher la bouffée de fumée au moyen d'un billboard :

   ```
   void Puff::display();
   ```

   Basez-vous sur le code de base fourni pour l'affichage du billboard avec un VBO et des shaders billboard.vsh et billboard.fsh.

Classe `Smoke`

4. Ecrire une classe **`Smoke`** pour représenter la colonne de fumée constitué de bouffées de classe **`Puff`**. Elle contiendra les données membres suivantes :
   - La position de l'origine de la colonne de fumée (**`QVector3D`**).
   - Une liste chaînée d'objets de classe **`Puff`**. On utilisera la classe **`list`** de la STL :

     ```
     list<Puff> puffsList;
     ```

   - **`timeInterval`** : L'intervalle de temps en secondes qui doit s'écouler entre deux émissions de bouffées de fumée (**`float`**).
   - **`elapsedTime :`** Le temps qui s'est écoulé depuis la dernière émission de bouffée (**`float`**).

5. Ecrire le constructeur de cette classe, permettant d'initialiser ces données (sauf la liste et **`elapsedTime`** qui sera initialisé à 0).

6. Ecrire une méthode **`animate()`** de la classe **`Smoke`** qui recevra en paramètre le temps écoulé en secondes entre deux affichages (donnée membre **`dt`** de la classe **`GLArea`**, obtenue avec un **`QElapsedTimer`**) :

```
void Smoke::animate(float dt);
```

On décomposera cette fonction en trois étapes :

1. **Ajout de bouffées** : ajouter la variable **`dt`** reçue en paramètre à la donnée membre **`elapsedTime`**. Si ce temps écoulé est supérieur à l'intervalle de temps **`timeInterval`**, alors remettre **`elapsedTime`** à zéro et ajouter un nouvel objet de classe **`Puff`** à la liste de bouffées (méthode **`push_back()`** de la classe **`list`** de la STL). Via le constructeur de cet objet **`Puff`**, vous lui donnerez une durée de vie aléatoire entre 5 et 8 secondes, une position initiale correspondant au point d'origine de la fumée, et un vecteur vitesse.

2. **Suppression de bouffées** : on parcours la liste de bouffées à la recherche de celles dont la durée de vie s'est complètement écoulée (≤ 0). Si c'est le cas, elles seront supprimées de la liste grâce à la méthode **`erase()`** de **`list`** :

```
list<Puff>::iterator i;
i = puffsList.begin();
while( i != puffsList.end() )
{
   if( i->life <= 0 )
      i = puffsList.erase(i);
   else
      i++;
}
```

3. **Calcul des nouvelles positions** : on calcule la nouvelle position des bouffées de la liste en appelant leurs méthodes **`animate()`** à laquelle on passe la variable **`dt`** en paramètre.

7. Ecrire une méthode **`display()`** de la classe **`Smoke`** qui affichera l'ensemble des bouffées de la liste :

```
void Smoke::display();
```

8. On constate que des bugs d'affichage se produisent en fonction de l'ordre d'affichage des fumées entre les billboards translucides et les objets opaques de la scène. Pour les éviter, il faut :

- Afficher les objets opaques (le sol, etc.)
- Activer le z-buffer en lecture seule avec **`glDepthMask(GL_FALSE)`**
- Afficher les objets translucides (les bouffées de fumée) du plus éloigné au plus proche
- Réactiver le z-buffer en lecture/écriture avec **`glDepthMask(GL_TRUE)`**

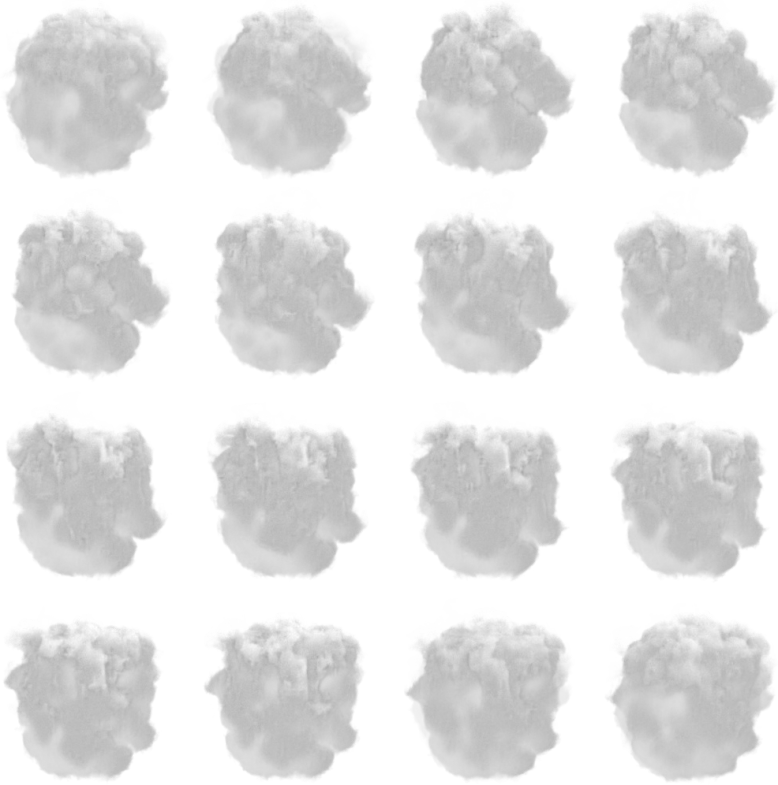
Shader de fumée

Vous allez apporter des modifications au fragment shader billboard.fsh :

9. Faites en sorte que les bouffées de fumée deviennent progressivement plus grosses et de plus en plus transparentes. Vous devrez passer le temps (**`elapsedTimer.elapsed()`**) en paramètre au fragment shader billboard.fsh.

10. Faites en sorte de pouvoir modifier la couleur de la fumée, grâce à une couleur passée au fragment shader.

11. Plutôt que de texturer les billboards avec l'image puff.png, vous allez maintenant utiliser l'image puffs.png qui contient un *atlas de textures* (c'est-à-dire plusieurs textures, ici 16, en une seule). Vous modifierez le fragment shader billboard.fsh de manière à utiliser successivement chacune des 16 sous-textures au cours du temps, de manière à donner à l'animation un aspect plus dynamique.



puffs.png

AMELIORATIONS POSSIBLES

12. Déplacez l'origine de la fumée dans le plan du sol avec les touches fléchées du clavier.

13. Ajouter à l'interface Qt des **`QSpinBox`** pour changer les paramètres du système de particules (intervalle d'émission, durée de vie des particules, couleur des particules avec un **`QColorDialog`**).

14. Utiliser un VBO par particule n'est pas une solution optimisée pour le GPU. Améliorez votre programme en utilisant le principe d'*instanciation* :
    http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/