POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

# INF2705 Infographie

# Spécification des requis du système

## Travail pratique 4

## *La fontaine de particules*

Département de génie informatique et génie logiciel
École polytechnique de Montréal
Automne 2019

# Table des matières

# 1 Introduction

Ce document décrit les exigences du TP4 « *La fontaine de particules* » du cours INF2705 Infographie.

## 1.1 But

Le but des travaux pratiques est de permettre à l'étudiant de directement appliquer les notions vues en classe.

## 1.2 Portée

Chaque travail pratique permet à l'étudiant d'aborder un sujet spécifique.

## 1.3 Remise

Faites la commande « `make remise` » afin de créer l'archive « **INF2705_remise_TPn.zip** » que vous déposerez ensuite dans Moodle. (Moodle ajoute automatiquement vos matricules ou le numéro de votre groupe au nom du fichier remis.)

Ce fichier `zip` contient le fichier Rapport.txt et tout le code source du TP (`makefile`, `*.h`, `*.cpp`, `*.glsl`, `*.txt`).

# 2 Description globale

## 2.1 But

Le but de ce TP est de permettre à l'étudiant d'assimiler des notions de mouvements d'objets basés sur des phénomènes physiques tels la gravité, le temps et les collisions en mettant en pratique le mode de rétroaction pour faire des calculs sur GPU. Ce TP utilise aussi des panneaux, des lutins et l'affichage en stéréoscopie.

## 2.2 Travail demandé

**Partie 1 : le mode rétroaction pour déplacer des particules sur GPU**

On demande d'afficher un système de particules évoluant dans le temps, comme illustré dans la Figure 1. Des particules naissent dans un puits de particules et meurent après une certaine période de temps. Un nombre constant (mais variable) de particules restent actives, ce qui veut dire que les particules mortes « revivent » à partir de la position courante du puits. Lors de la naissance (ou de la renaissance) d'une particule, on lui donne une direction aléatoire de départ, une couleur aléatoire, de même qu'une durée de vie aléatoire.

Les attributs des particules sont stockés dans un VBO et ces attributs sont modifiés en utilisant le mode de rétroaction afin que les calculs soient faits en parallèle sur le GPU. Pour démarrer, on pourra utiliser les fichiers « `main.cpp` » et « `nuanceurs.glsl` » des exemples du mode de rétroaction utilisant un VBO : cours.polymtl.ca/inf2705/exemples/11-RetroactionC-vbo/.

Les collisions avec les parois de la bulle (un demi-ellipsoïde déformable) sont relativement faciles à gérer : on doit seulement vérifier que la particule demeure toujours à l'intérieur de la bulle. On peut transformer les positions et les normales vers une sphère de rayon unitaire pour faire les calculs de collision comme montré à l'annexe D.

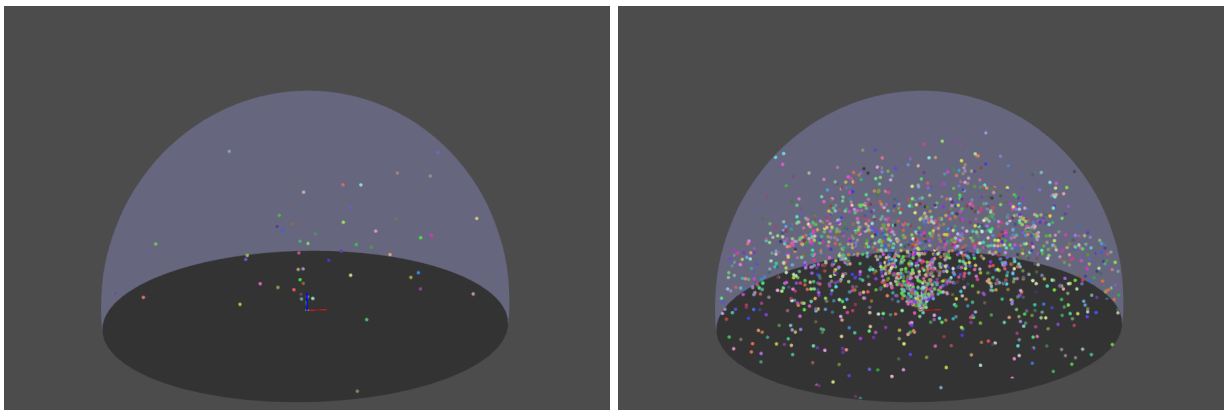

FIGURE 1 – Système de particules avec peu ou énormément de particules avec un nuanceur de géométrie qui affiche des points sans utiliser de texture

**Partie 2 : l'utilisation de lutins**

Les particules seront affichées avec des lutins (figure 2). Les lutins utilisés sont fournis dans des textures et représentent une étincelle ou un oiseau. Le canal alpha des textures sera utilisé afin de ne pas afficher (`discard`) les fragments transparents lorsque `texel.a < 0.1`. Ce simple test dans le nuanceur de fragments fera le travail.

Pour démarrer, on s'inspirera des nuanceurs des exemples pour l'affichage de panneaux et des lutins : cours.polymtl.ca/inf2705/exemples/10-Panneau/

Afin de bien comprendre l'utilisation des nuanceurs de géométrie, on y générera des coordonnées de texture qui varieront en fonction du temps de vie restant à chaque « particule ». On fera ainsi tourner l'étincelle autour de son centre (Figure 3) en construisant une une matrice de rotation, avec un angle de rotation de « `6.0 * tempsDeVieRestant` » (6 rad/s), qui sera appliquée aux coordonnées des sommets. (Pourrait-on simplement faire tourner les coordonnées de texture ? Aller lire la Q2 du Rapport.txt. ;-))

De façon semblable, pour faire voler l'oiseau (Figure 4), on fera varier les coordonnées de texture afin d'accéder à une des 16 sous-images de la texture des oiseaux. (Une fréquence de battements d'ailes de « `18.0 * tempsDeVieRestant` » (18 Hz) fonctionnent très bien.) La fonction `modulo` permet alors de calculer facilement le numéro de la sous-texture (== le décalage à appliquer) dans la texture originale :

```
int num = int( mod( 18.0 * tempsDeVieRestant, 16.0 ) );
texCoord.x = ( texCoord.x + num ) / 16.0;
```

Enfin, pour colorer le rendu final du fragment lorsqu'une texture est utilisée, la couleur de la particule modifiera la couleur de la texture avec la formule « `mix( couleur.rgb, texel.rgb, 0.6 )` », tout en conservant la valeur alpha de la couleur de la particule.

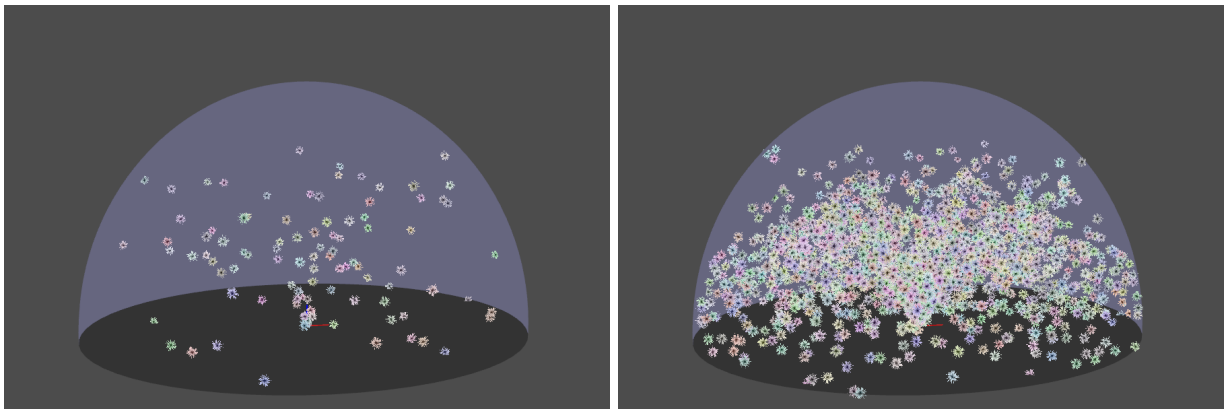

FIGURE 2 – Système de particules avec des lutins : peu ou beaucoup d'étincelles

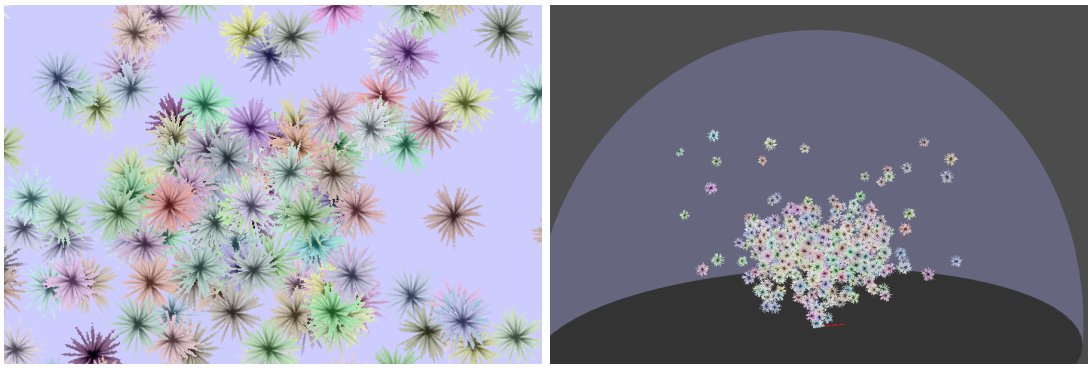

FIGURE 3 – Système de particules avec les étincelles (deux points de vue différents)

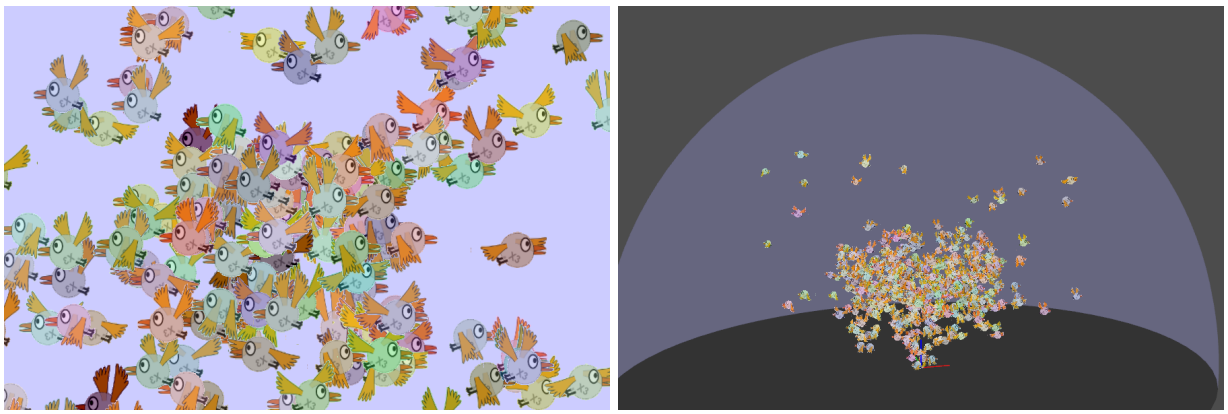

FIGURE 4 – Système de particules avec les oiseaux

**Partie 3 : Affiner le rendu et l'affichage en stéréo**

Pour que le rendu soit plus cohérent, on s'assurera que les oiseaux volent toujours vers l'avant plutôt que de quelquefois voler en « marche arrière ». On corrigera ainsi le vol des oiseaux vers la gauche ou vers la droite, selon l'axe des X. (Indice : utilisez la fonction `sign()`.) Les étincelles peuvent aussi être affectées, mais ce n'est pas nécessaire.

D'autre part, afin de mieux percevoir la profondeur (et pour s'amuser en 3D !), la scène pourra être vue en stéréoscopie en mode anaglyphe ou en mode double en utilisant une clôture différente pour chaque œil (voir Figure 5). On pourra utiliser les exemples du cours pour l'affichage stéréo en anaglyphe : `cours.polymtl.ca/inf2705/exemples/13-StereoAnaglyphe/`.

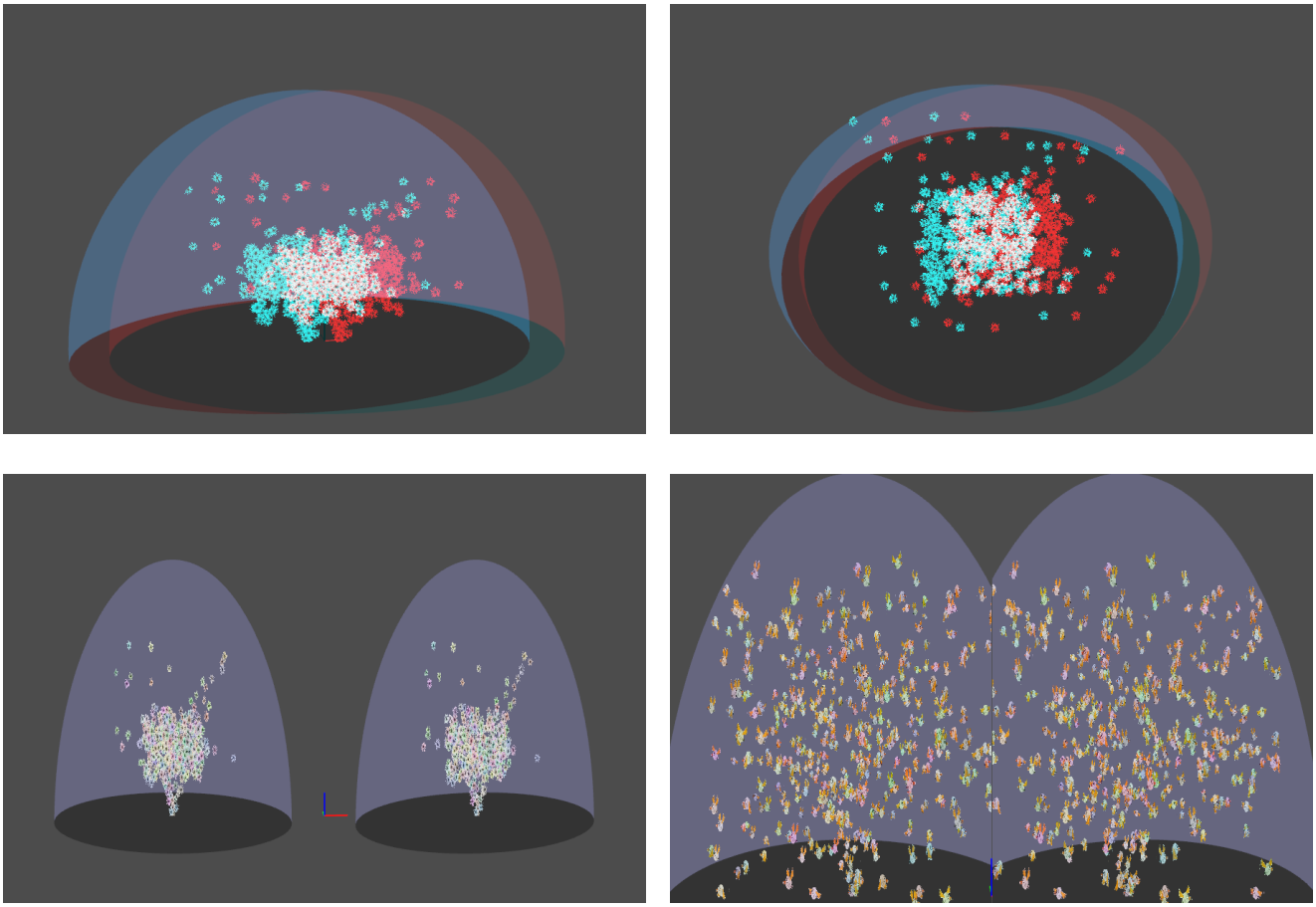

FIGURE 5 – Affichage en anaglyphe (pour des lunettes anaglyphes) ou en mode double (pour un écran autostéréoscopique)

# 3 Exigences

## 3.1 Exigences fonctionnelles

Partie 1 :

E1. Le mode de rétroaction est bien utilisé pour avancer les particules. [3 points]

E2. Les particules (re)naissent toutes à la position du puits avec une trajectoire aléatoire de départ. [1 point]

E3. Les particules ont une durée de vie aléatoire (entre 0 et `tempsDeVieMax` secondes). [1 point]

E4. Les particules ont une couleur aléatoire (chaque composante entre COULMIN et COULMAX). [1 point]

E5. La gravité est implémentée correctement, donnant une trajectoire de parabole aux particules. [2 points]

E6. Les particules rebondissent sur les parois par collision rigide. [2 points]

Partie 2 :

E7. Les particules sont représentées avec des lutins. [2 points]

E8. Les étincelles tournent autour de leur centre. [3 points]

E9. Les oiseaux volent. (Les lutins varient en fonction du temps.) [3 points]

E10. Les variations de l'étincelle et le vol des oiseaux dépendent du temps de vie restant de chaque particule. [2 points]

Partie 3 :

E11. Les oiseaux volent toujours dans le bon sens et non quelquefois vers l'arrière. [3 points]

E12. L'affichage peut être en mono, en stéréo anaglyphe ou en stéréo double. [5 points]

## 3.2 Exigences non fonctionnelles

De façon générale, le code que vous ajouterez sera de bonne qualité. Évitez les énoncés superflus (qui montrent que vous ne comprenez pas bien ce que vous faites !), les commentaires erronés ou simplement absents, les mauvaises indentations, etc. [2 points]

## 3.3 Rapport

Vous devez répondre aux questions dans le fichier `Rapport.txt` qui sera inclus dans la remise. Vos réponses doivent être complètes et suffisamment détaillées. (Quelqu'un pourrait suivre les instructions que vous avez écrites sans avoir à ajouter quoi que ce soit.) [2 points]

# 4 Textures utilisées

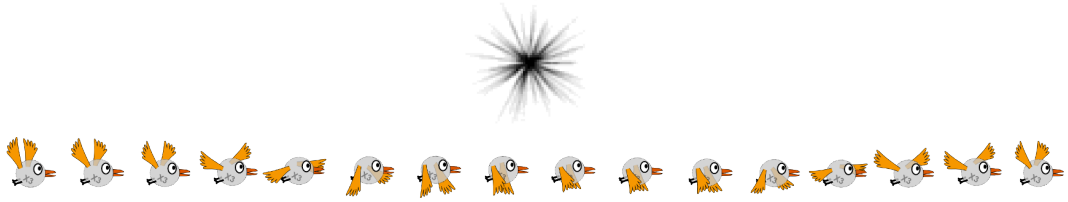

FIGURE 6 – Les textures fournies

# ANNEXES

## A Liste des commandes

| Touche | Description |
|---|---|
| `q` | Quitter l'application |
| `x` | Activer/désactiver l'affichage des axes |
| `v` | Recharger les fichiers des nuanceurs et recréer le programme |
| `j` | Incrémenter le nombre de particules |
| `u` | Décrémenter le nombre de particules |
| `DROITE` | Augmenter la dimension de la boîte en X |
| `GAUCHE` | Diminuer la dimension de la boîte en X |
| `HAUT` | Augmenter la dimension de la boîte en Y |
| `BAS` | Diminuer la dimension de la boîte en Y |
| `PAGEPREC` | Augmenter la dimension de la boîte en Z |
| `PAGESUIV` | Diminuer la dimension de la boîte en Z |
| `0` | Remettre le puits à la position (0,0,0) |
| `PLUS` | Avancer la caméra |
| `MOINS` | Reculer la caméra |
| `b` | Incrémenter la gravité |
| `h` | Décrémenter la gravité |
| `l` | Incrémenter la durée de vie maximale |
| `k` | Décrémenter la durée de vie maximale |
| `f` | Incrémenter la taille des particules |
| `d` | Décrémenter la taille des particules |
| `t` | Changer la texture utilisée : 0-aucune, 1-étincelle, 2-oiseau |
| `i` | on veut faire une impression |
| `a` | Boucler sur les textures automatiquement |
| `s` | Varier le type d'affichage stéréo : mono, stéréo anaglyphe, stéréo double |
| `g` | Permuter l'affichage en fil de fer ou plein |
| `ESPACE` | Mettre en pause ou reprendre l'animation |
| `BOUTON GAUCHE` | Manipuler la caméra |
| `BOUTON DROIT` | Déplacer le puits |
| `Molette` | Changer la distance de la caméra |

# B Figures supplémentaires

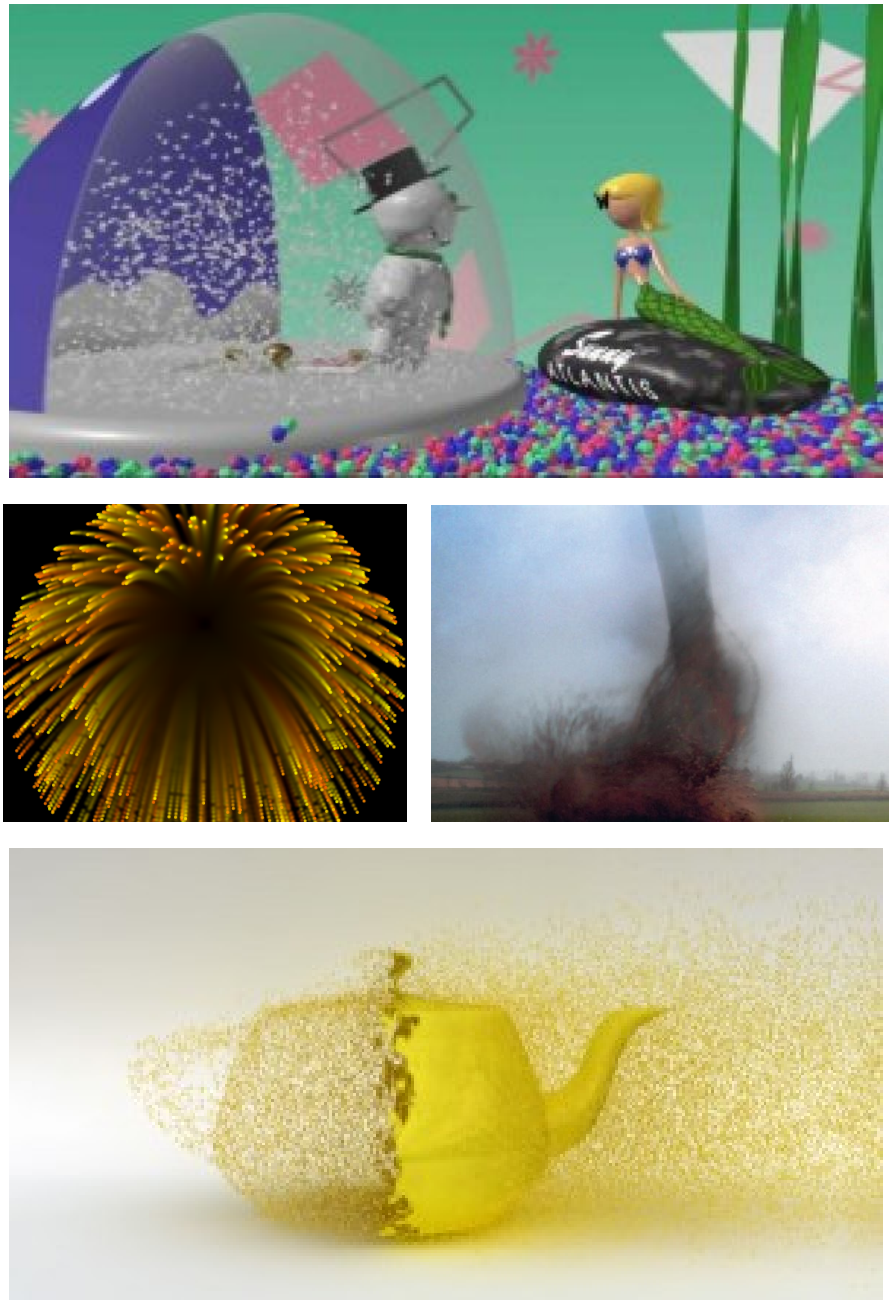

FIGURE 7 – Divers systèmes de particules

# C Apprentissage supplémentaire

1. Amortir le mouvement des particules lors de chaque collision.
2. Gérer les collisions sur des parois orientées différemment.
3. Gérer les collisions sur des objets quelconques dans la scène.
4. Lorsque le nombre de collisions est impair, inverser la couleur de la particule dans le nuanceur de sommets.
5. Utiliser une couleur différente selon le nombre de collisions et faire que la particule meure après un certain nombre de collisions.
6. Modifier la transparence de la particule selon son âge.
7. Faire en sorte que la gravité soit appliquée de façon un peu différente selon la couleur de la particule (par exemple : une rouge tombe plus vite et une bleu moins vite).
8. Ajouter des contrôles pour changer la direction de la gravité.
9. (*) Afficher des sphères avec un niveau de détail différent selon la distance à l'observateur.
10. (*) Faire en sorte que chaque particule laisse une trace dans l'espace.
11. (*) Faire que les particules soient des sources lumineuses, chacune avec une intensité de « `1.0/nparticules` », afin d'éclairer les parois intérieures.

* : *Pour ces éléments, il est préférable de n'afficher qu'un petit nombre de particules.*

# D Formules utilisées

### Intégration d'Euler pour avancer une particule

Pour avancer les particules, on peut utiliser le méthode d'Euler. Ce n'est pas la méthode la plus précise, mais elle est très simple :

```
positionMod = position + dt * vitesse;
vitesseMod = vitesse;
```

### Collisions avec les parois

Pour gérer les collisions « rigides » (sans perte de vitesse) avec les parois de la bulle (un demi-ellipsoïde déformé), on doit vérifier que la particule demeure toujours à l'intérieur de la bulle. On peut transformer les positions et les normales à une sphère de rayon unitaire pour faire ces vérifications de collision.

Pour transformer la position vers la sphère unitaire, il faut appliquer l'inverse de la transformation de modélisation utilisée pour déformer la sphère originale (une `FormeSphere` de rayon 1). Dans le cas présent, on sait que c'est une unique mise à l'échelle par le `vec3 bDim`. On doit ainsi *diviser* chaque composante de la position par la dimension correspondante :

```
vec3 posSphUnitaire = positionMod / bDim;
```

Par contre, pour transformation un vecteur, il faut se rappeler le calcul des normales pour l'illumination. Il appliquer *l'inverse* de la transformation ci-dessus, c'est-à-dire l'inverse de l'inverse de la mise à l'échelle ! On doit ainsi *multiplier* chaque composante de la vitesse par la dimension correspondante :

```
vec3 vitSphUnitaire = vitesseMod * bDim;
```

Alors, pour vérifier si la particule est encore à l'intérieur de la bulle (=à l'intérieur de la sphère unitaire), on peut vérifier si la longueur de `posSphUnitaire` est inférieure à `1`.

Si la particule est sortie de la bulle, on peut la ramener à l'intérieur avec l'approximation ci-dessous pour obtenir un effet de collision et, surtout, utiliser la fonction GLSL « `reflect(V,N)` » pour obtenir une nouvelle direction du vecteur vitesse après la collision :

```
float dist = length( posSphUnitaire );
if ( dist >= 1.0 ) // ... la particule est sortie de la bulle
{
   positionMod = ( 2.0 - dist ) * positionMod;
   vec3 N = posSphUnitaire / dist; // normaliser N
   vec3 vitReflechieSphUnitaire = reflect( vitSphUnitaire, N );
   vitesseMod = vitReflechieSphUnitaire / bDim;
}
```

(*Note* : Pour tester vos collisions, vous pouvez temporairement modifier dans votre nuanceur et donner la même direction de départ à toutes les particules. Vous pouvez alors choisir explicitement une direction pour faire quelques tests, par exemple : `vitesseMod = vec3( -0.8, 0., 0.6 );`.)