POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

INF2705 Infographie

Spécification des requis du système

Travail pratique 4

Le système de particules sur GPU

Département de génie informatique et génie logiciel
École polytechnique de Montréal
Hiver 2017

Table des matières

1 Introduction

Ce document décrit les exigences fonctionnelles et non fonctionnelles du TP4 « *Le système de particules sur GPU* » du cours INF2705 Infographie.

1.1 But

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

1.2 Portée

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

1.3 Remise

Vous remettrez un fichier `zip` contenant tout le code source du TP et le rapport (`*.cpp`, `*.h`, `*.glsl`, `makefile`, `*.txt`).

Faites « `make remise` » pour créer l'archive « `remise.zip` ».
Vous déposerez ensuite ce fichier dans Moodle.

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. Ce TP permet aussi de mettre en pratique le mode de rétroaction pour faire des calculs sur GPU et l'utilisation des lutins.

2.2 Travail demandé

Partie 1 : le système de 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 l anaissance (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.

Les collisions avec les parois de la bulle (une demi-sphère déformable) sont assez faciles à gérer : on doit simplement 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 (voir annexe D).

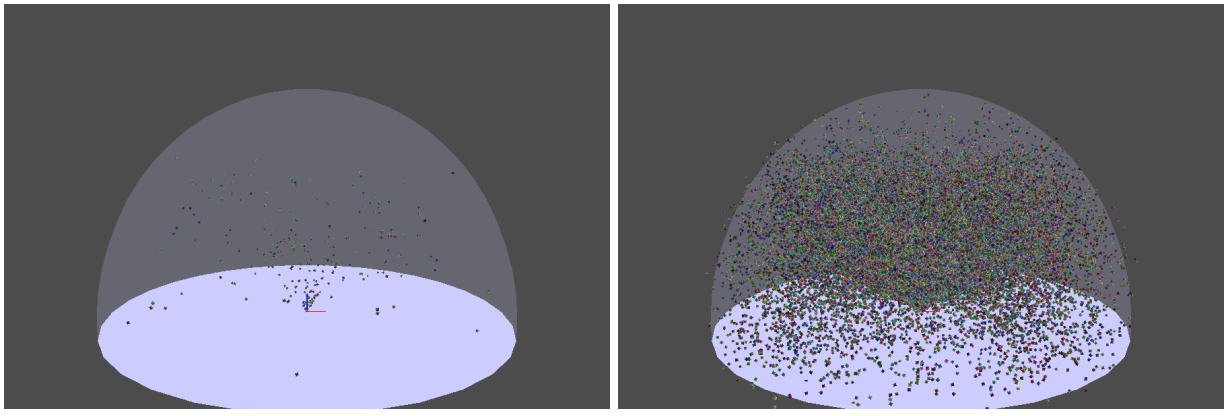

FIGURE 1 – Système de particules avec peu ou énormément de particules avec un nuanceur de géométrie qui affiche des points sans utiliser de texture

Partie 2 : l'affichage avec des lutins

Les particules sont affichées avec des lutins (figure 2). Les lutins utilisés sont fournis dans des textures et représentent une étincelle, un oiseau ou un leprechaun[1]. On utilisera le canal alpha des textures afin de ne pas afficher (`discard`) les fragments transparents lorsque `texel.a < 0.1`.

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 tourner l'étincelle autour de son centre (Figure 3) en construisant une une matrice de rotation qui sera appliquée aux coordonnées des sommets. On fera voler l'oiseau (Figure 4) et tourner le leprechaun (Figure 5) en variant les coordonnées de texture de façon à accéder à une des 16 sous-images de ces deux textures. (Un angle de rotation de « `4.*tempsDeVieRestant` » et une fréquence de battements d'ailes de « `20.*tempsDeVieRestant` » fonctionnent très bien.) Enfin, pour produire la couleur du finale du fragment, on utilisera la couleur de la particule pour teinter un peu la texture en utilisant la formule « `mix( couleur.rgb, texel.rgb, 0.6 )` », tout en conservant la valeur alpha de la couleur de la particule.

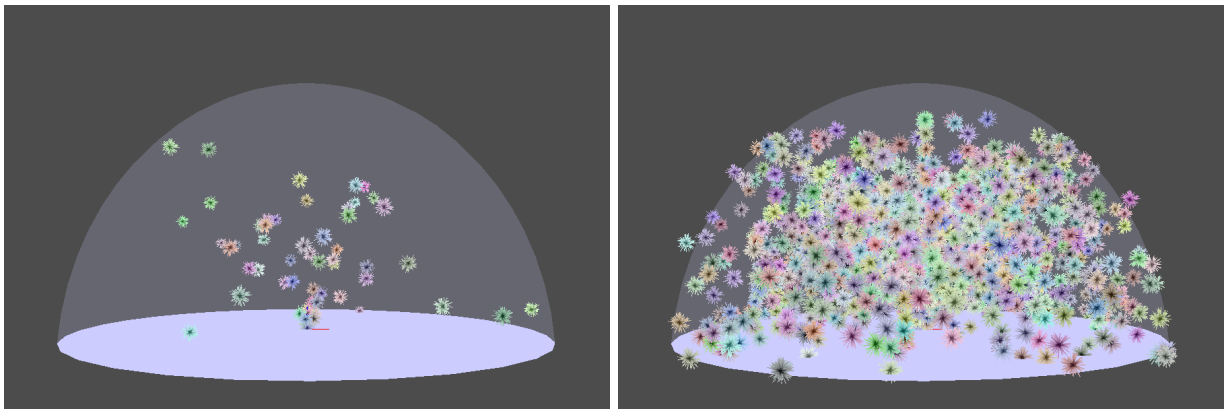

FIGURE 2 – Système de particules avec des lutins : peu ou beaucoup d'étincelles

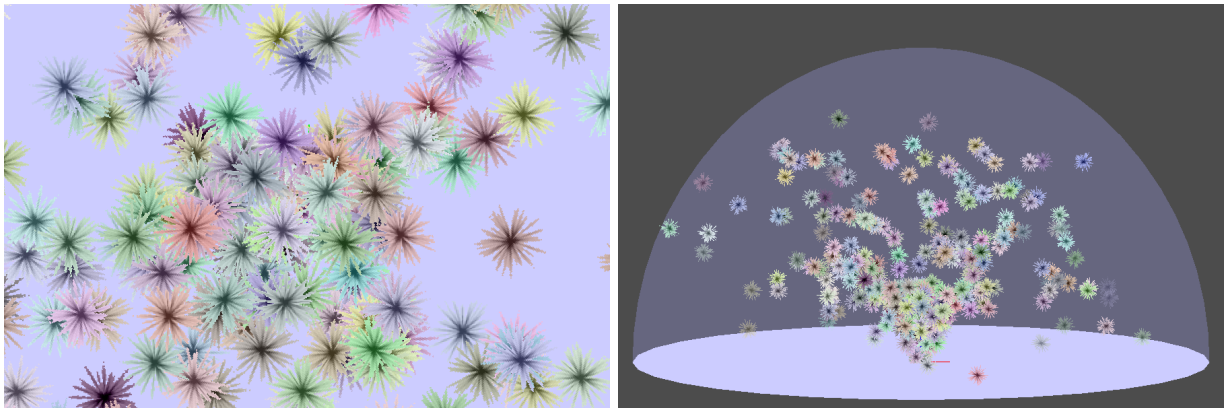

FIGURE 3 – Système de particules avec les étincelles (deux points de vue différents)

1. Un leprechaun est le farfadet par excellence pour la fête des Irlandais, le 17 mars ! :)

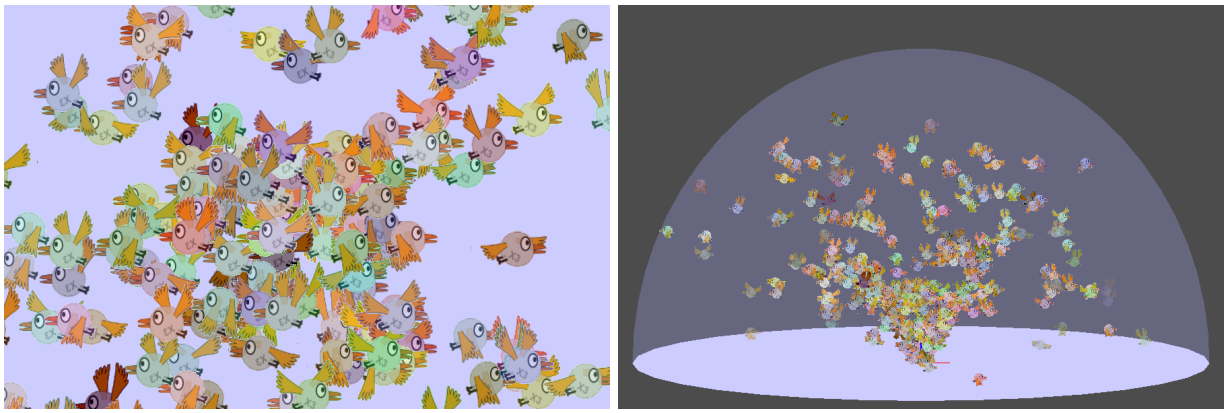

FIGURE 4 – Système de particules avec les oiseaux

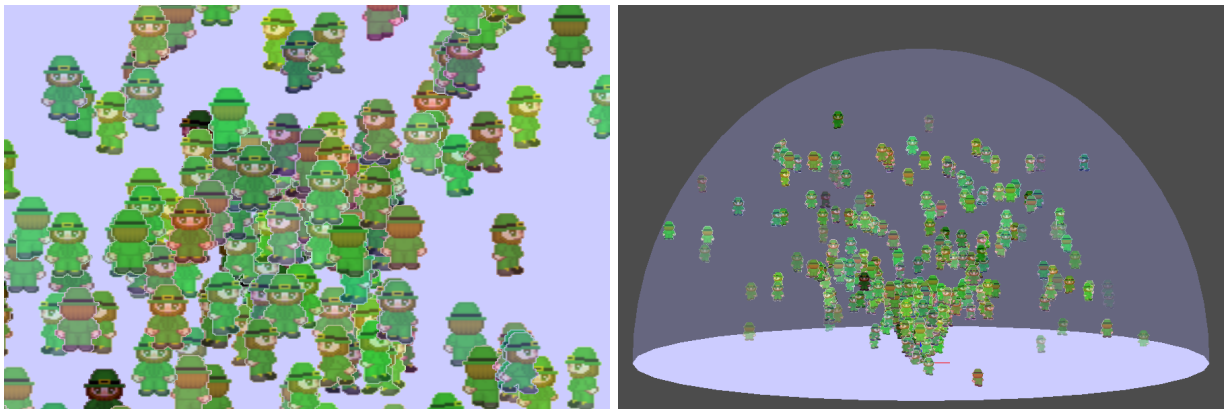

FIGURE 5 – Système de particules avec les leprechauns

Partie 3 : Affiner le rendu

À chaque collision d'une particule avec la paroi, la particule deviendra plus transparente en multipliant la composante alpha de sa couleur par un facteur de « 0.5 ».

On s'assurera aussi 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 et les leprechauns peuvent aussi être affectés, mais ce n'est pas nécessaire.

2.3 Fichiers fournis

Pour démarrer, on pourra utiliser les nuanceurs vus dans les exemples d'utilisation du mode de rétroaction : www.groupes.polymtl.ca/inf2705/exemples/09-retroaction/retroD-vbo.cpp.

On pourra aussi utiliser les nuanceurs de l'exemple pour l'affichage de panneaux et des lutins : www.groupes.polymtl.ca/inf2705/exemples/09-panneau/

3 Exigences

3.1 Exigences fonctionnelles

Partie 1 :

E1. Le mode de rétroaction est bien utilisé pour avancer les particules.

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

E3. Les particules ont une durée de vie aléatoire (entre 0 et `tempsVieMax` secondes).

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

E5. La gravité est implémentée correctement, donnant une trajectoire de parabole aux particules.

E6. Les particules rebondissent sur les parois par collision rigide.

Partie 2 :

E7. Les particules sont représentées avec des lutins.

E8. Les étincelles tournent autour de leur centre.

E9. Les oiseaux volent et les leprechauns tournoient sur eux-mêmes.

E10. Les variations de l'étincelle, des oiseaux et des leprechauns dépendent du temps de vie restant de chaque particule.

Partie 3 :

E11. Les particules deviennent deux fois plus transparentes à chaque collision.

E12. Les oiseaux volent toujours dans le bon sens et non quelquefois vers l'arrière.

3.2 Rapport

Vous devez répondre aux questions dans le fichier `Rapport.txt` et l'inclure 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.)

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
`BAS`	Augmenter la dimension de la boîte en Y
`HAUT`	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
`t`	Changer la texture utilisée : 0-aucune, 1-étincelle, 2-oiseau, 3-leprechaun
`p`	Permuter la projection : perspective ou orthogonale
`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

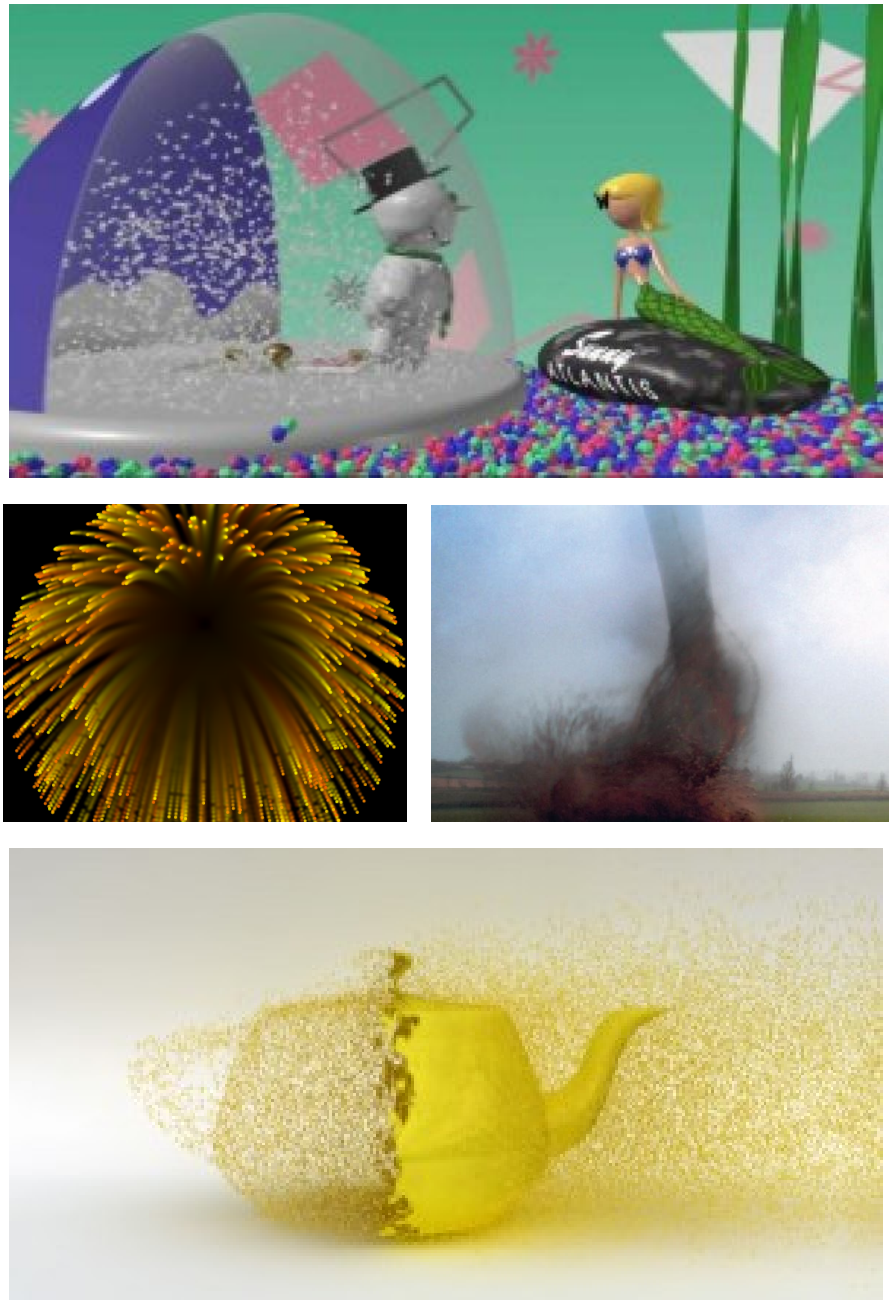

FIGURE 6 – Divers systèmes de particules

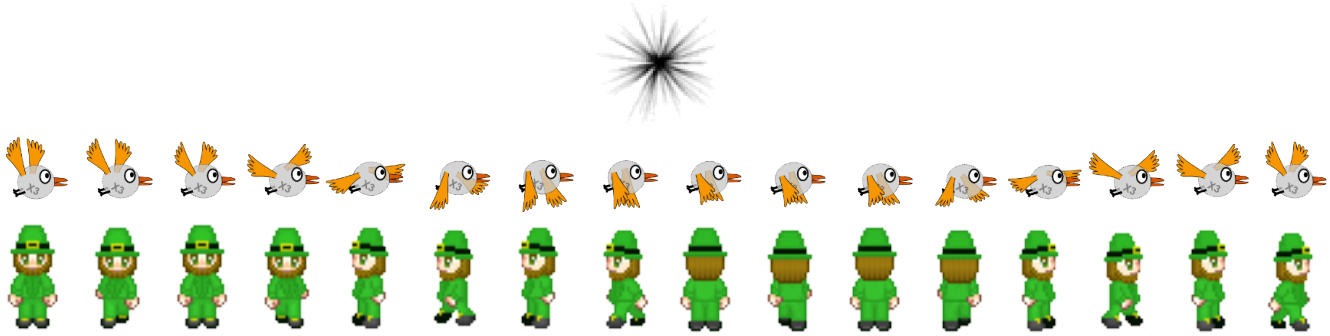

FIGURE 7 – Les textures fournies

C Apprentissage supplémentaire

1. Amortir le mouvement des particules lors de chaque collision.
2. Gérer les collisions sur des murs orientés différemment.
3. Gérer les collisions sur des objets quelconques dans la scène.
4. Ajouter des contrôles pour changer la direction de la gravité.
5. Lorsque le nombre de collisions est impair, inverser la couleur de la particule dans le nuanceur de sommets.
6. Utiliser une couleur différente selon le nombre de collisions et faire que la particule meure après un certain nombre de collisions.
7. Modifier la transparence de la particule selon son âge.
8. (*) Faire en sorte que chaque particule laisse une trace dans l'espace.
9. (*) Afficher des sphères avec un niveau de détail différent selon la distance à l'observateur.
10. (*) 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 + vitesse * dt;
vitesseMod = vitesse;
```

Collisions avec les parois

Pour gérer les collisions « rigides » (sans perte de vitesse) avec les parois de la bulle (une demi-sphère déformée), 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( -0.35, 0., 0.5 );`.)