Système de particules
Projet d'IN55

Adrien BERTHET, Gautier CLAISSE et Karim NAAJI

PRINTEMPS 2014

Introduction

Pour ce projet, le sujet abordant le système de particules (numéro 2) a été retenu. Le choix de ce sujet s'est fait sur l'intérêt de la gestion physique de ces particules de manière efficace. Dans les jeux vidéos, les systèmes de particules sont trop peu souvent utilisés, pour gérer des phénomènes tels que la fumée ou le feu, au profit de simples sprites (ce phénomène est toujours présent même dans des jeux récents, à l'instar de Watch_Dog).

Dans une première partie, le projet ainsi que les objectifs fixés seront présentés. Les différentes phases de réalisation et d'implémentations de l'application seront ensuite abordés. Ce rapport se terminera sur l'évocation des problèmes rencontrés tout au long du projet, et les possibles améliorations à apporter.

Sommaire

1 Présentation du projet

1.1 Objectifs fixés

Le sujet choisi avait pour consignes de paramétrer le rendu de particules constitutives d'un phénomène physique (fumée, feu, eau) et de simuler leur rendu dans une scène. La gestion de la physique doit être réalisée avec des calculs GPU, c'est à dire avec GLSL et des shaders.

À partir de ces consignes, nous avons décidé de réaliser un programme de rendu contenant un émetteur de particules. Cet objet sert de point d'émission aux particules, qui vont alors se déplacer suivant différents *patterns* précis : par exemple un cône, une pyramide ou encore une sphère (émission non limitée dans tous les sens). Pour ne pas avoir d'attente trop haute, ces objectifs sont les principaux fixés. D'autres, considérés comme secondaires, ont été évoqués. Il serait ensuite possible de placer plusieurs émetteurs (directement par le code ou par l'utilisateur via la souris) et ainsi cela permettrait d'observer la réaction de particules se rencontrant entre elles.

1.2 Résultat final

Le rendu final est celui escompté : deux types d'émissions sont disponibles (émissions sphérique et cônique) et également un « plan » de particules qui a un mouvement semblable à des vagues. L'ensemble des tranformations des particules a été réalisée à travers un ensemble de /emphshaders, que cela soit pour les mouvements comme pour les couleurs. L'interface est également présente, même si limitée, par manque de temps principalement et peu de connaissance sur le sujet des widgets Qt (voir paragraphe 3.4).

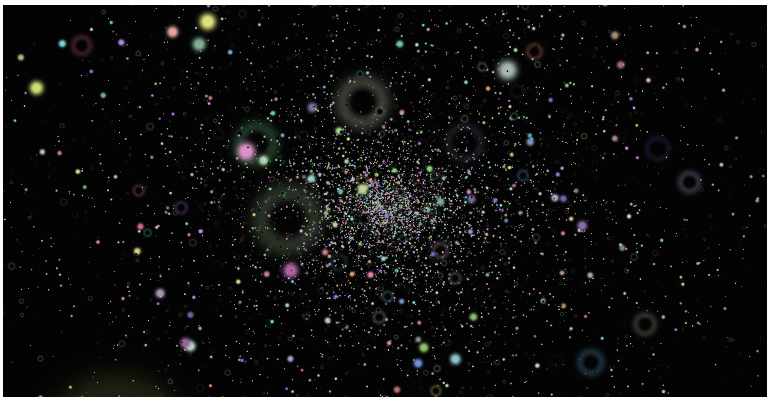

FIGURE 1.1 – Rendu à émission spérique

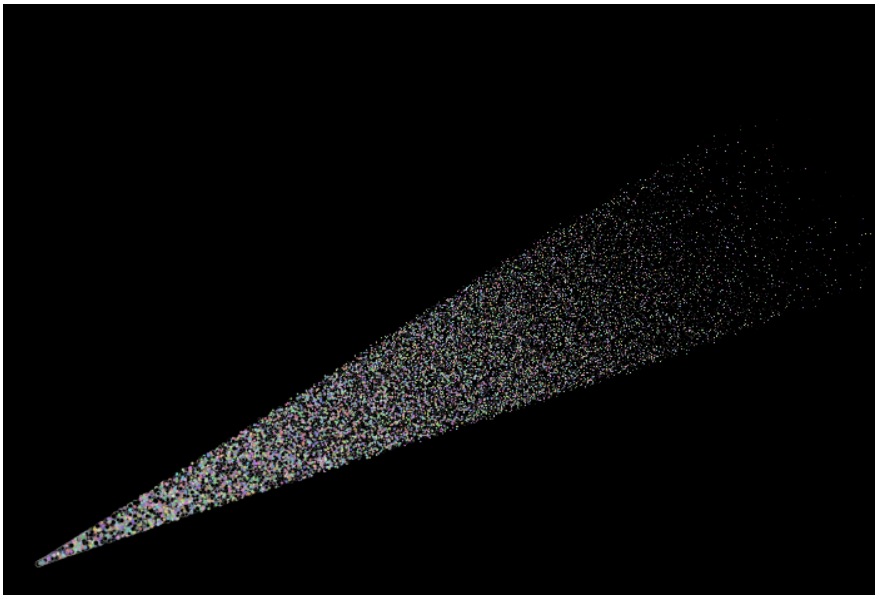

FIGURE 1.2 – Rendu avec diffusion cônique

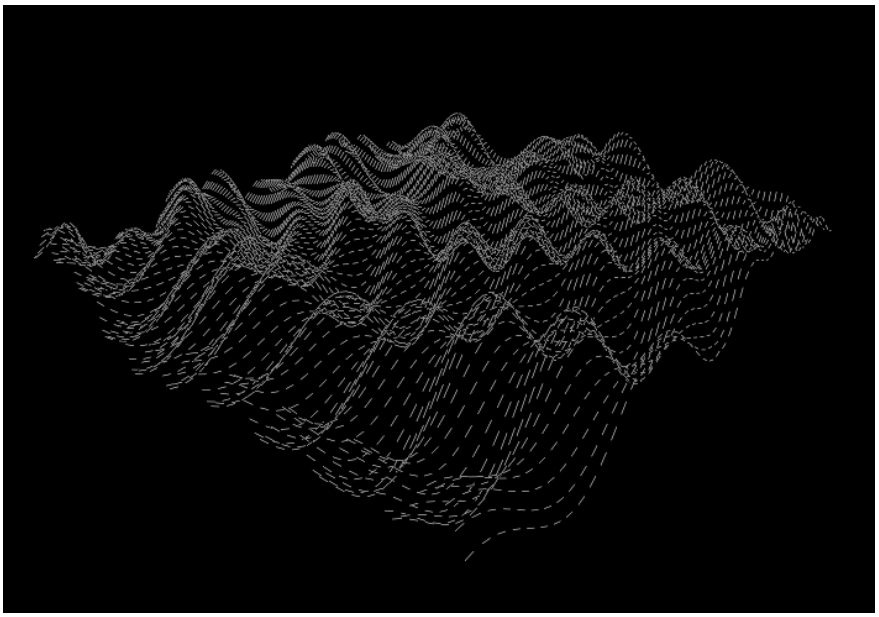

FIGURE 1.3 – Rendu avec des vagues

2

Modélisation et implémentation

2.1 Fonctionnement global

3
Difficultés rencontrées

Un certain nombre de problèmes ont été rencontrés tout au long du développement du projet. Ces complications sont de nature technique (par exemple avec la mise en place du GUI, la gestion de l'alpha avec les textures) mais principalement de nature géométrique.

3.1 Notations matricielles américaine et française

Le premier problème, qui a apporté un certain sentiment de confusion, a été le sens de notation. En effet, les notations américaine et française pour l'écriture matricielle ne sont pas les mêmes. Les colonnes et les lignes sont inversées entre les deux.

Comme il a été décidé de ne pas reprendre le framework fourni dans les TPs, mais de réécrire complétement une plateforme from scratch, tous les calculs pour des classes de base (*Matrix4*, *Vec3*, etc...) ont été réalisés en utilisant la notation matricielle française. Cependant, le système de la caméra libre a été conçu d'après le TP concerné et plusieurs ressources sur Internet. Il y a donc eu une confusion sur l'écriture, puisque la totalité des ressources consultées concernaient la notation américaine. Ainsi, une quantité importante de soucis liés à ces malentendus est apparue, principalement pour la gestion des transformations de la camera, et le calcul de la matrice *MVP*. Par exemple, une déformation importante de l'objet était présente, comme sur une caméra avec un objectif de type FishEye. Mais le but étant d'obtenir une caméra classique, l'ensemble des matrices a été uniformisé dans la notation française.

3.2 Rotation semi-automatique de la caméra

La gestion de la caméra est sans doute l'élément le plus difficile de ce projet. Celle ci est contrôlée avec les touches fléchées, mais également avec la souris (il faut appuyer sur la touche *Alt* pour verrouiller/déverrouiller le mouvement de la souris). Lorsque l'on effectue plusieurs rotations avec la souris, alors que celle-ci est censée effectuer un simple mouvement sur deux axes (x et y), il semblerait qu'elle réalise également une rotation sur l'axe z. Ce problème est apparu et a disparu à plusieurs reprises, sans que la véritable raison soit réellement identifiée.

3.3 Gestion de l'alpha des textures

Les particules sont toutes considérées comme un point unique de l'espace, sur lesquelles est appliquée une texture. Dans un premier temps, la texture était un simple carré blanc. Puis celle-ci a évolué pour une image plus complexe, comportant un canal alpha.

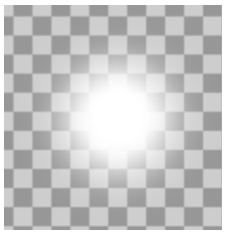

FIGURE 3.1 – Texture avec présence du canal alpha

Par défaut, OpenGL ne reconnaît aucun canal alpha avec les textures. Il faut le préciser à l'aide des constantes spécifiques. Le chargement de la texture se fait à travers la classe *Texture*. Dans un premier temps, il a été convenu d'utiliser la fonction *glBlendFunc* avec différentes constantes, comme *GL_SRC_ALPHA*. Cependant cette fonction n'a pas convenu à l'usage ci-présent, car elle d'une part beaucoup plus utilisée pour la gestion des couleurs directement en OpenGL, et d'autre part la texture n'est pas appliquée sur une zone spécifique, mais sur un point.

C'est dans la fonction *glTexImage2D* qu'il faut s'orienter. Celle-ci possède notamment un paramètre permettant de définir le nombre de couleurs composant l'image à utiliser en texture. La constante *GL_RGBA* est à associer à cette fonction, et permet ainsi un gestion du canal alpha pour cette texture de particule.

3.4 Insertion du rendu OpenGL dans un GUI

Une des consignes du projet était d'avoir une interface Qt autour de la fenêtre de rendu OpenGL. Cette interface doit permettre, au travers de boutons ou autres éléments d'interface, de changer plusieurs paramètres de l'émetteur mis en place. Personne ne possédant d'expérience avec les GUI et Widget Qt, un apprentissage en plus a dû être réalisé. Dans un premier temps, une fenêtre Qt de type « MainWindow » a été utilisé. Pour des raisons qui sont toujours obscures (il s'agirait sans doute d'un problème lié à la plateforme Mac OSX), il est impossible de dessiner le rendu OpenGL. Un message indique que la taille du widget contenant le rendu est trop petite, même pour une taille dix fois plus grande que le rendu.

La solution est d'avoir une base simple, c'est à dire un simple Widget Qt en fenêtre principale. Il suffit d'adapter par la suite l'application existante pour l'inclure dans ce nouveau widget, puis de réaliser les *bind* nécessaires pour relier les actions des boutons de l'interface aux actions de l'application. Il ne faut également pas oublier de déplacer la prise en compte des évènements claviers dans le widget parent, mais pas nécessairement les évènements souris.

4
Améliorations possibles

Conclusion