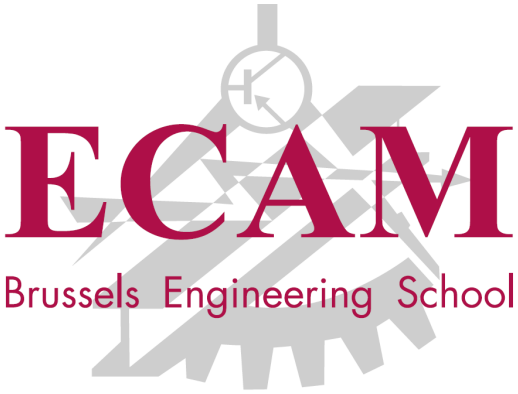

École Centrale des Arts et métiers

Module Code – Synthèses d'images

Laboratoire Synthèses d'images

Auteurs:
Hubert Mugabo Pitie
Salim Boutlendj

Enseignant:
Quentin Lurkin

1er juin 2018

Table des matières

1 Objectif

Sur base du code 3D-Engine-Base, réaliser une simulation de fluide basée sur la méthode FLIP/PIC décrite dans le document pic-flip.pdf.

2 Application

Notre simulation utilise un cube dans lequel nous faisons apparaître les particules, une vidéo de la simulation est disponivle en annexe sur le repository **Github**

2.1 Implémentation du cube

Nous avons décidé de modéliser un cube pour faciliter l'implémentation de la simulation. Le fluide apparaît à l'intérieur du cube et y reste grâce à des limites que nous avons fixé.
Pour rajouter le cube, nous avons modifier la classe 'Simulator.cpp'.
Dans la fonction 'Simulator : :render()', nous avons ajouté un renderer pour les lignes formant le cube :

```
linesRenderer->render(PRIMITIVE_LINES, 24);
```

Le premier paramètre permet de dessiner une droite en précisant les deux extrémités de cellle-ci. Le deuxième paramètre est le nombre de points nécessaire. Comme le cube est composé de 12 droites, il faut dessiner 24 points. Ceux-ci sont stockés dans un array de float. ('linesPoint')

```
float limit = 10.0;
float linesPoint[] = {
    -1*limit,-1*limit , -1*limit,
     limit, -1*limit, -1*limit,

    -1*limit,-1*limit , -1*limit,
    -1*limit, limit , -1*limit,

    .
    .
    .
};
```

Listing 1: Implémentation des lignes du cube

Le paramètre 'limit' permet de faciliter l'implémentation du cube en utilisant une variable correspondant à la longueur d'une droite. Ensuite l'array est utilisé pour créer un Buffer contenant les droites à dessiner. Le rendu de celui est ensuite géré par un renderer ('linesRenderer dans le code').

2.2 Implémentation de l'apparition des particules

Tout d'abord nous avons défini le nombre de particules composant le fluide (50000) via une variable privée. ('countOfParticules')
Ensuite nous avons défini le positionnement et la vitesse de chaque particule via la fonction random ('rand()').
Par exemple 'rand()%10' permet de générer un float aléatoire entre 0 et 10.
Puis les particules sont stockées dans un buffer pour être rendu.

```
 for( int a = 1; a < countOfParticules * rendererVar; a = a + 1 ) {
   pp.push_back(Particule(vec3(
   (rand()%10)+1.0f,
   (rand()%18)+1.0f,
   (rand()%10)+1.0f),

   vec3(
   (rand()%3+rand()%4)+1.0f,
   (rand()%3+rand()%4)+1.0f ,
   (rand()%3+rand()%4)+1.0f), vec3(1)));
}
```

Listing 2: Création des particules

2.3 Implémentation des limites des particules

La limitation des particules a été défini dans un **Compute Shader**, la logique qui a été implémentée est la suivante :

```
    if(p.y < -10) {
        p.y=-10;
        v.y=-0.6*v.y;
    }if(p.y > 10) {
        p.y = 10;
        v.y=-0.6*v.y;
    }  if(p.z < -10 ) {
        p.z=-10;
        v.z=-0.6*v.z;
    }  if(p.z > 10) {
        p.z=10;
        v.z=-0.6*v.z;
    }  if(p.x < -10) {
        p.x=-10;
        v.x=-0.6*v.x;
    }  if(p.x > 10) {
        p.x=10;
        v.x=-0.6*v.x;
    }
```

Listing 3: Core du Compute Shader qui gère la logique de la Simulation

Les limites correspondent aux dimensions du cube pour que les particules restent dedans. Une fois la limite atteinte, la vitesse est inversée pour s'assurer que les particules restent à l'intérieur du cube.

3 Conclusion

Pour conclure, le réalisation de la simulation de fluide basée sur la méthode FLIP/PIC n'est pas aboutie, par faute de connaissances et de temps. Sans oublier le fait que l'utilisation d'**OpenGL** par la biais de Shaders est très complexe. De plus le processus de debugging est n'est pas une chose aisée. L'utilisation de framework tel que **Unity** avec des **shaders** aurait été plus simple et rapide pour un même résultat.

4 Améliorations

En ce qui concerne les améliorations, nous avons trouvé un livre de **référence** qui après étude et compréhension permettra sans doute la réalisation complète de la simulation.