# Rapport ANI3D


- Melvin Gidel (melvin.gidel@epita.fr (mailto:melvin.gidel@epita.fr))
- Mélanie Tchéou (melanie.tcheou@epita.fr (mailto:melanie.tcheou@epita.fr))


## Setup le projet

Pour cloner le dépôt avec la librairie, utilisez la commande suivante:

```
git clone --recursive git@github.com:MelBeer/ANI3D-particle-based-fluid-physics
```

Il vous suffira ensuite de build le projet de la façon suivante:

```
cd scene
mkdir build
cd build
cmake ..
make -j$(nproc)
cd ..
build/main
```

## Introduction

Nous avons choisi de nous baser sur le TP 4 : *Simulation - Spheres in Collision* pour créer une simulation de physique de fluide à base de particules à l'aide de la technique de marching cubes, le tout optimisé afin de rendre l'animation plus fluide.

## Les étapes principales de rendu

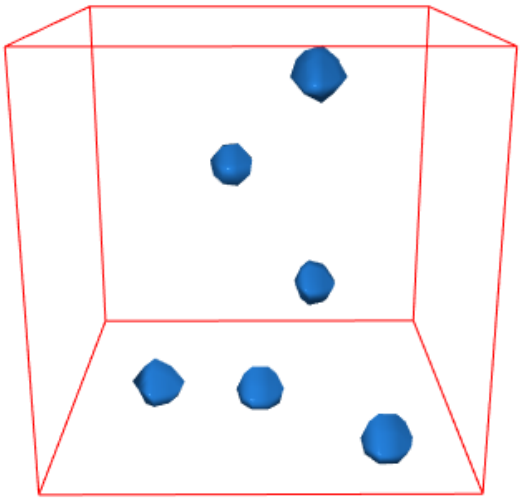
Les fichiers fournis par les TPs nous donnent accès à la classe scene_structure qui nous permet de stocker les différentes caractéristiques de la scène:

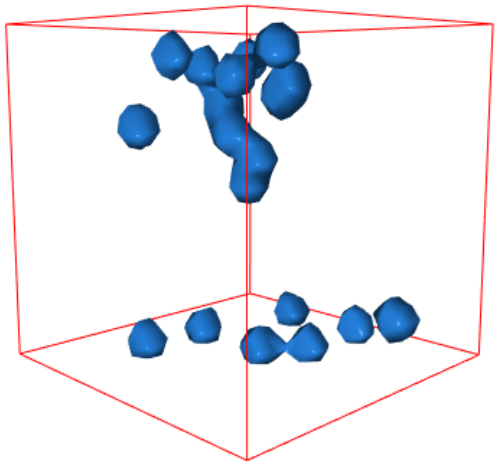
- **`initialize`** nous permet d'initialiser notre scène en l'appelant une fois au début. Nous avons modifié cette fonction pour ajouter la création des surfaces implicites, nécessaire à l'implémentation des marching cubes.
- **`display_frame`** est ensuite appelée pour chaque frame.

Le projet est basé sur du rendu OpenGL, mais grâce au code déjà fourni, nous n'avons pas à gérer ses spécificités, ce qui nous a rendu la tâche plus facile et agréable.

## La scène

La scène est composée d'un cube qui contiendra les particules de l'animation. Il s'agit donc du même que pour le TP. Des particules sont émises depuis son centre avec une vélocité et une direction aléatoire, tant que l'on n'a pas atteint la limite.



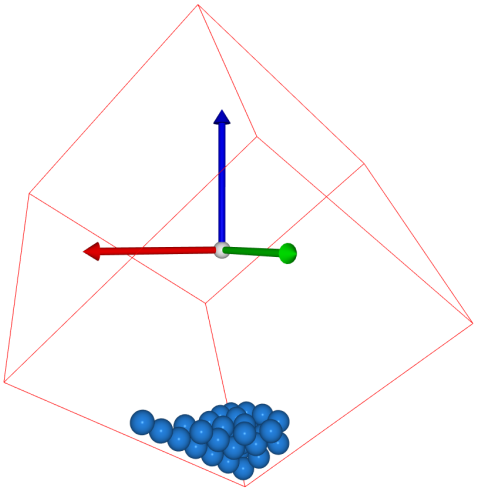


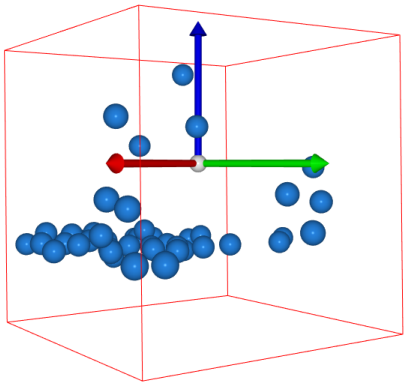
## Aparté : La suite du TP - Déplacer / Pivoter le cube

Pour le TP, nous avions originellement deux listes contenant respectivement la position des différentes faces du cube ainsi que leur normale.
Il suffisait alors de register les inputs de l'utilisateur pour créer des matrices de rotoation ou de translations.

Nous avions bind la touche LeftShift pour cette partie. Ainsi, appuyer sur la touche LeftShift et bouger la souris faisait pivoter le cube autour de l'axe représenté par la viewDirection, et appuyer sur la touche LeftCtrl en plus faisait déplacer le cube sur le plande la caméra.

Les opérations concernant la rotation était appliquées sur les faces et les normales, tandis que les normales n'étaient pas modifiées quand il s'agissait d'une translation.





Cette partie n'a pas apporté de difficultées spéciales et, même si elle est toujours disponible dans le rendu final, elle n'en représente pas une feature officielle. En effet, pour optimiser le rendu, nous avons utilisé une structure accélératrice que l'on détaillera par la suite qui est limité au cube, ainsi quand on transforme ce dernier les particules sortent de l'environnement de calcul et on reçoit quelques overflow de mécontentement du programme.

## Physique des particules

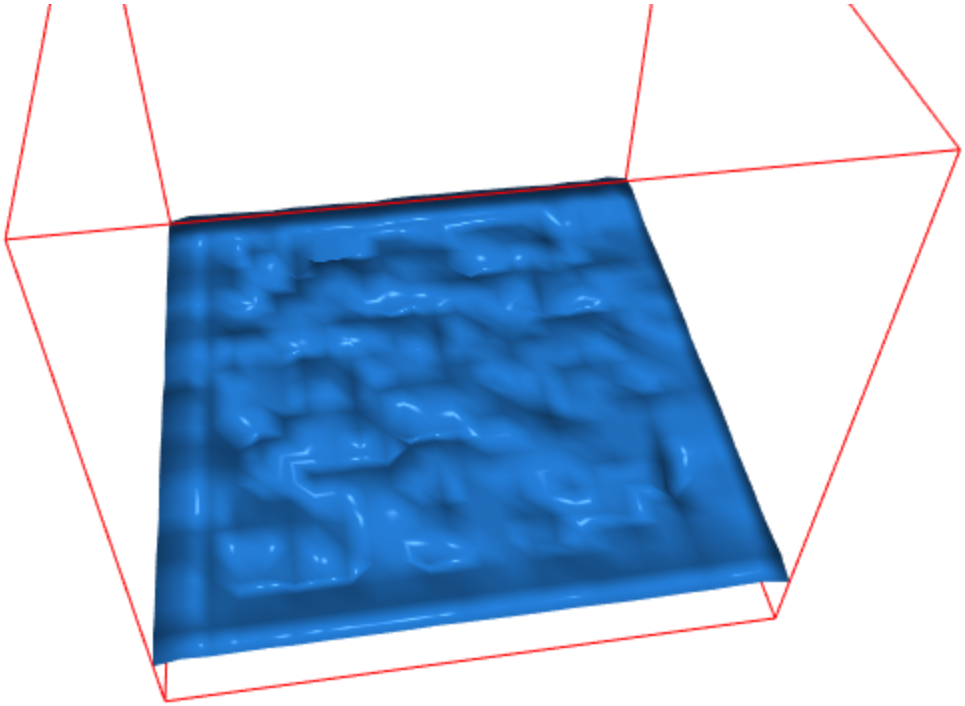
L'implémentation de la physique des particules a été plutôt aisée. Nous nous sommes basés sur la méthode SPH (Smoothed Particle Hydrodynamics), qui est notamment expliquée dans les slides du cours ici (https://imagecomputing.net/damien.rohmer/teaching/inf585/lecture/20_fluids_II/html/content/010/index.html). Il nous suffisait de transcrire les formules suivantes et les appliquer à notre modèle:

$$F_{weight} = m_i g$$

$$F_{pressure} = -\frac{m_i}{\rho_i} \sum_{j}^{j \neq i} m_j \frac{\mathrm{p}_i + \mathrm{p}_j}{2\,\rho_j} \nabla W_h^{spiky}(\|p_i - p_j\|)$$

$$F_{viscosity} = m_i\,\nu \sum_{j}^{j \neq i} m_j \frac{(\mathrm{v}_j - \mathrm{v}_i)}{\rho_j} \triangle W_h^{spiky}(\|p_i - p_j\|)$$

Nous n'avons cependant pas réussi à comprendre comment retranscrire parfaitement l'interaction des particules sur les murs, c'est pourquoi nous avons estimé une fonction d'énergie.



## Rendu avec marching cubes

Nous nous sommes inspirés des exemples présents dans le dépôt Github de la librairie CGP disponibles ici: https://github.com/drohmer/CGP/tree/main/examples/02_marching_cubes (https://github.com/drohmer/CGP/tree/main/examples/02_marching_cubes)

CGP effectue de lui-même l'algorithme de marching cube. Nous avons donc représenté la scène sous forme de grille 3D dans laquelle nous appliquons une fonction d'évaluation du terrain. Cela nous permet d'évaluer le terrain en fonction des objets présents dans la scène. CGP utilise ainsi les valeurs de chaque cellule pour déterminer les surfaces implicites de la scène.

## Optimisation

Nous avons utilisé une `uniform grid` pour enregistrer les particules, et au cours de la simulation, nous récupérons la liste des cellules qui non-vides qui en contiennent pour ne vérifier que celles-ci ainsi que leurs voisines, ce qui nous assure un gain de temps plus que non négligeable. C'est d'ailleurs d'ici que vient notre plus grosse optimisation.

## Problèmes rencontrés

Le plus gros problème aura été de comprendre comment intégrer l'algorithme de marching cubes de CGP, et adapter de tout aux hashgrids pour l'optimisation. Le tout s'est révélé plus

complexe que prévu.

Nous avons aussi eu des soucis sur la qualité du rendu au début. En effet, nous étions en pair programming sur Visual Studio Code grâce à l'extension Live Share. Nous faisions tourner le projet sur l'ordinateur de Melvin mais les résultats étaient peu fluides, et nous pensions que les problèmes venaient uniquement de notre algorithmique. Cependant, une fois que nous avions fait tout ce que nous pouvions, le rendu n'était pas parfaitement fluide. Nous avons alors essayé de faire tourner le projet sur l'ordinateur de Mélanie, et le résultat a tout de suite été plus fluide.

## Rendu final

Voici une vidéo de nos résultats. Les particules restant assez grosses par rapport à la taille du cube, on se retrouve avec une texture assez visqueuse qui rappelle un peu celle de l'huile.

La vidéo est également disponible ici (https://youtu.be/I6efF40hBK8).

## Conclusion

Les TPs et le projet à la fin étaient très intéressants et plaisants à faire de par leur sujet et leur format, d'autant plus que la librairie proposée nous aidait énormément quant à l'utilisation d'OpenGL ce qui était plus que plaisant. Cela nous a permis de nous concentrer sur les aspects plus intéressants de notre animation.