



ÉCOLE POLYTECHNIQUE

INF-585 : COMPUTER ANIMATION

Modélisation et animation d'une cascade

Professor :
Damien Rohmer

Group :
Julien Bergerot
Jules Arbelot

30 avril 2021

Table des matières

1	Introduction et motivations	2
2	Notre projet	3
2.1	Création rapide de l'environnement	3
2.2	Simulation du mouvement des particules	4
2.3	Construction du mesh du liquide	5
2.3.1	Screen Space Fluid Rendering	5
2.3.2	Marching Cubes	5
3	Futur travail	7
4	Conclusion	8

1 Introduction et motivations

La question de savoir comment simuler de l'eau a toujours été une question au centre des études de l'image de synthèse. Depuis l'apparition de la créature d'eau dans Abyss en 1989, nous avons vu l'eau générée par image de synthèse apparaître de plus en plus au cinéma et dans les jeux vidéos.



FIGURE 1 – Créature d'eau du film Abyss

Aujourd'hui il est commun de voir des simulations d'eau très réalistes et de nombreux softwares comme Maya, Blender permettent d'encadrer ces simulations pour les artistes.

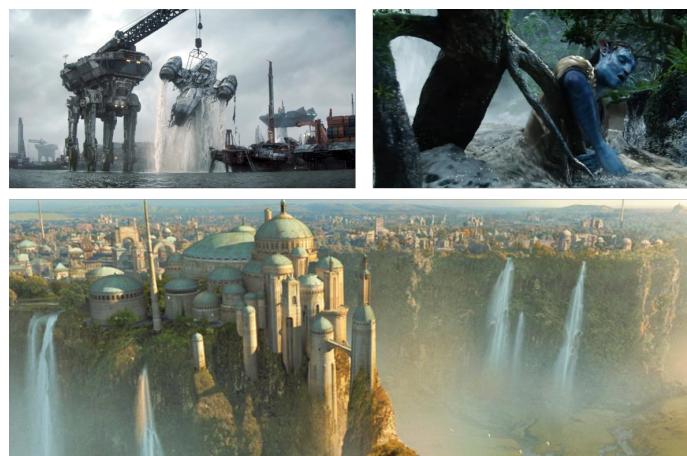


FIGURE 2 – Eau CGI réalistes des films "The Mandalorian", "Avatar" et "La Menace Fantôme"

En cours, nous avons étudié à plusieurs reprises les questions de simulations de particules et comment à partir de cela nous pouvons obtenir une animation de fluide. Dans les deux derniers TD,

nous avons vu comment se modélisent la pression, la densité ou la viscosité sur les particules. Ainsi donc, il nous a semblé intéressant de voir ce qu'on pouvait ajouter à ces notions pour obtenir une simulation de cascade plus réaliste.

2 Notre projet

Ainsi donc nous nous sommes lancés dans ce projet en récupérant le code du SPH que nous avions déjà développé pour en faire une cascade dans un décor et un affichage 3D du fluide plutôt qu'un affichage 2D. Nous avions pour objectif de simuler une cascade, mais cela servait surtout d'excuse pour tenter d'étendre la simulation de fluide de la 2D à la 3D, où le rendu prend une toute nouvelle difficulté.

2.1 Création rapide de l'environnement

Dans un premier temps nous avons modélisé un rapide environnement pour y faire passer notre cascade. Un simple terrain est créé, dont la principale variation est due à la fonction arctan. Pour donner un aspect plus réaliste, du bruit de perlin a été ajouté, ainsi qu'une faible pente avant la cascade. L'arrivée de la cascade, quant à elle, a été modélisée par une retenue d'eau, capable d'accueillir les résultats de notre cascade.

Nous avons alors une partie centrale, avec une texture rocheuse, définissant le fond de la cascade, c'est-à-dire l'endroit où passerait l'eau. Autour, une étendue d'herbe colle parfaitement au passage de l'eau. L'objectif n'est pas de faire un environnement magnifique mais simplement de pouvoir facilement se repérer et comprendre l'environnement rapidement.

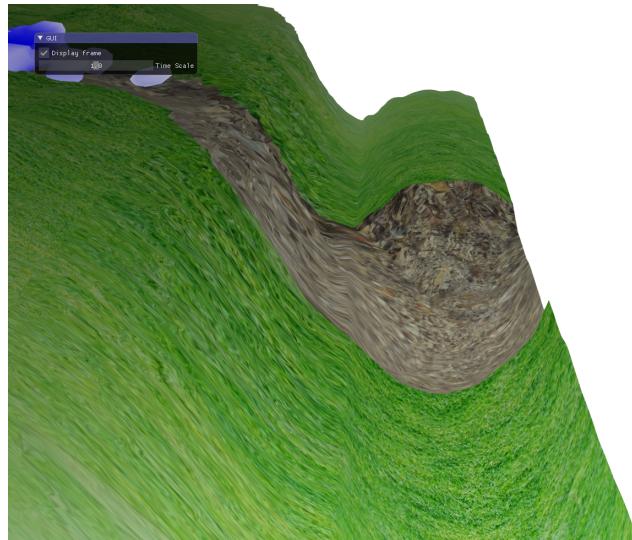


FIGURE 3 – Mise en place du décors conduisant la cascade

2.2 Simulation du mouvement des particules

En nous inspirant de notre dernier TD, l'eau est représentée par des particules d'un certain diamètre possédant certaines caractéristiques que nous avons adaptées (densité, rayon d'action, viscosité). Ces particules sont alors créées à intervalles réguliers depuis une position initiale (en haut de la cascade) avec des petits décalage aléatoires autour de celle-ci.

Ces particules vont alors évoluer au fil du temps en répondant à l'équation de Navier-Stokes. Cette équation est résolue à chaque instant. Pour cela, nous utilisons la méthode SPH, qui nous a été introduite dans le cours. Nous avons utilisé les fonctions que nous avons obtenues en résolvant les exercices pendant le TD. Les seules modifications que nous avons faites ont été de modifier (=augmenter) la valeur de la gravité, afin d'attirer réellement l'eau au sol lors de la cascade et de leur donner une certaine vitesse lors de leur descente après leur naissance.

Nous avions alors stocké dans un vecteur de particules toutes ces particules et leur paramètres. La fonction *simulate* déjà implémentée permettait d'obtenir ces résultats sans oublier les nouvelles particules régulièrement rajoutées.

Afin que les particules ne puissent pas circuler librement sur toute la carte, nous avons aussi rajouter des collisions invisibles entre les particules et les bords de la rivière. Ainsi, il était impossible aux particules de sortir du lit de la rivière. Néanmoins, il était possible, pour la partie cascade, aux particules de se retrouver aussi hautes que souhaité, pour simuler les éclaboussures.

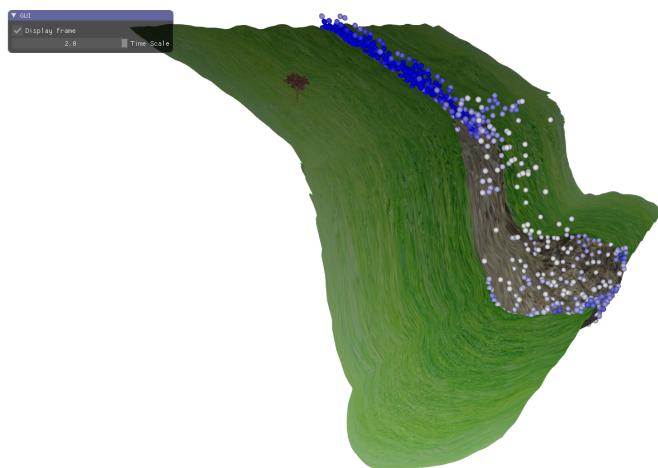


FIGURE 4 – Cascade de particules

Dans un premier temps, nous avons décidé d'obtenir un rendu très simple en ne dessiner que les particules simulées. Une première étape a été d'assimiler à chaque particule une couleur, en fonction de certains paramètres. Nous voulions au moins obtenir une couleur foncée pour le courant, clair pour la cascade et plus foncée pour la retenue d'eau, avec quelques éclaboussures blanches.

Nous avons logiquement pensé à considérer la densité pour représenter la couleur. Bien que chaque

particule possédait un argument *densité*, celui-ci n'était pas pertinent car le résultat n'était que peu réaliste (une seule couleur, même en jouant sur une normalisation pour vraiment varier entre 0 et 1). Nous avons alors souhaité calculé nous même la densité pour chaque particule, en sommant sur toutes les particules, avec un coefficient relatif à la distance à la particule à colorer, qui donne cette formule (à sommer sur chaque particule, où r est la distance).

$$0.25 * \exp\left(-\frac{r^2}{0.1^2}\right) \quad (1)$$

C'est alors le résultat que vous pouvez observer ci-dessus.

Néanmoins, nous avons des problèmes de performances, car nos ordinateurs ont tendance à beaucoup ralentir à partir d'un certain nombre de particules. Nous n'avions ainsi pas assez de temps pour obtenir un lac "rempli" et celui-ci apparaissait toujours trop clair. Il est à noter que ce problème de lac trop grand et de particules pas assez nombreuses (à cause de nos machines) allait sûrement poser des problèmes pour notre rendu plus développé.

2.3 Construction du mesh du liquide

Maintenant que le mouvement des particules était calculé, il nous fallait travailler sur le rendu des particules. Bien que le rendu précédemment mentionné était encourageant, nous souhaitions obtenir quelque chose de plus réel et continu.

Nous avons alors décidé d'opter pour des solutions différentes.

2.3.1 Screen Space Fluid Rendering

La première est le *Screen Space Fluid Rendering*. Cette technique est assez peu gourmande au niveau des ressources mais demande pas mal de travail. De plus, il nous fallait travailler au niveau des shaders, et cela nous imposait de coder les fichiers *.glsl*, un langage qui nous était inconnu. Les résultats semblent intéressants, nous avons tenté de nous familiariser avec ce langage, mais malgré nos quelques tentatives, nous n'avions pas le temps pour aller assez loin et obtenir le résultat désiré.

La seconde méthode est la solution que nous avons choisi d'implémenter, les *Marching Cubes*.

2.3.2 Marching Cubes

Cette méthode est très souvent utilisée pour définir des isosurfaces, basés sur un champ de scalaire. Bien que donnant un rendu acceptable, cette méthode est souvent assez coûteuse, et cela risque de ne pas donner des résultats très visibles (FPS assez faible) et appréciable à l'œil à vitesse réelle.

Nous avons logiquement travaillé en 3D, et nous allons résumer rapidement le fonctionnement de cet algorithme. La première étape de séparer l'espace en petits cubes. Chaque sommet va posséder une certaine valeur, issue du champ scalaire donné en entrée. Accompagné de ce champ, une isovaleur va être spécifiée. L'objectif est de créer le mesh dont le contour aura cette valeur de scalaire.

L'espace 3D est alors parcouru cube par cube. Pour chacun des huit sommets du cube, une valeur lui est associé s'il est au-dessus ou non de l'isovaleur (1 ou 0). Nous avons alors un nombre entre 0 et 255 (en convertissant les 0 et 1 en binaire ($2^8 - 1 = 255$)). Ces 256 configurations possibles sont

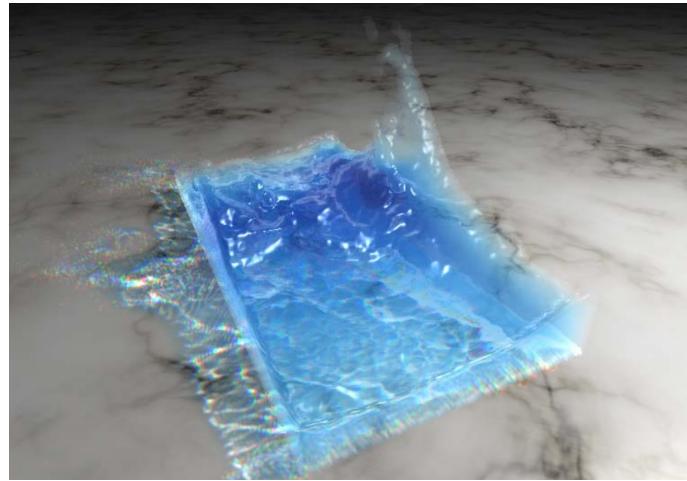


FIGURE 5 – Rendu d'eau avec la méthode du Screen Space Fluid Rendering

connues et il est alors possible de déterminer par où passera le mesh final dans ce cube. Le long de chaque arrête séparant un 0 d'un 1, un point sera créé afin d'être relié pour former une partie du mesh final (ce point verra sa position ajustée par interpolation linéaire entre les deux sommets).

Cette image permet d'expliquer le fonctionnement de l'algorithme en 2D. Bien qu'en 2D le mesh créé soit effectivement toute la surface bleue, en 3D seuls les points séparant de l'isovaleur formeront le mesh et l'intérieur sera vide.

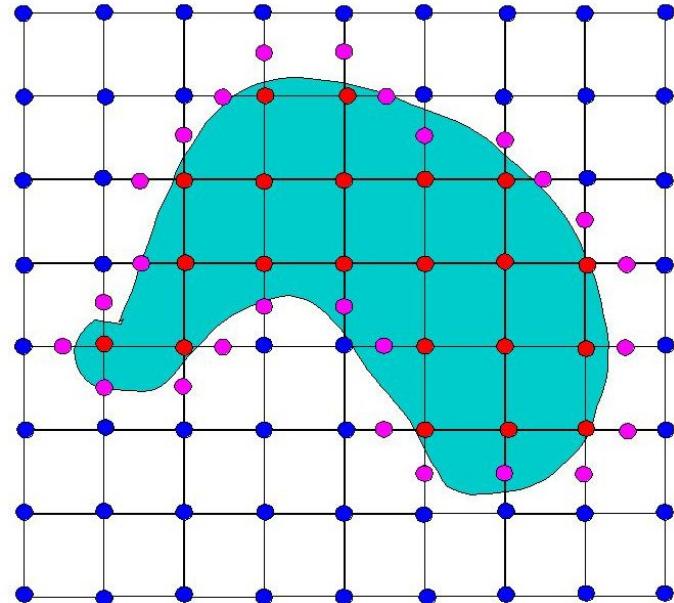


FIGURE 6 – Exemple de Marching Cubes en 2D

Après avoir compris le fonctionnement de cet algorithme, nous avons décidé de récupérer un code fonctionnel, par gain de temps, l'étape de codage étant assez longue. Il nous fallait alors désigner un champ scalaire et une isovaleur. Le champ de scalaire, comme vu dans la partie 2.2, est défini par la densité en chaque point. Une somme sur les particules, pondérée par les distances au point considéré a défini les valeurs. L'isovaleur, elle, a été défini par expérience, pour obtenir un résultat adéquat.

Cet algorithme étant assez long, nous avons décidé de séparer notre espace en trois pour obtenir de meilleurs résultats et supprimer des calculs inutiles (on sait qu'il n'y aura jamais d'eau sous la terre à cause des contraintes posées). Nous avons alors défini trois grilles pour obtenir trois mesh. La première partie est la partie descente, rivière de l'eau. La seconde est la partie cascade et éclaboussure. La troisième est la partie calme, la partie lac de notre simulation. Cela permet d'ajouter des détails à nos différentes parties sans avoir un temps absurde de simulation.

La dernière difficulté était celle de la couleur. Ayant utilisé une fonction de densité comme champ scalaire, si nous utilisions la même pour colorer le mesh, celui-ci serait logiquement de couleur uniforme (avec peu de variance tout du moins). Pour palier à cela, nous avons implémenter une seconde fonction pour calculer une densité, en modifier le rayon de captation, et l'influence des particules.

$$d * \exp\left(-\frac{r^2}{\sigma^2}\right) \quad (2)$$

Pour les deux fonctions de densité, cette valeur était sommée pour toutes les particules. r représente la distance au point considéré, et d et σ sont des variables à adapter. En choisissant les bonnes valeurs, nous avons réussi à obtenir des mesh cohérents et qui avait effectivement des couleurs plutôt foncées dans le courant et clair dans la cascade. Le problème du lac reste toujours un problème, dû à notre manque de ressource. Néanmoins, si un grand nombre de particules sont créées dans le lac uniquement, le résultat ressemble effectivement à un lac, ce qui reste rassurant, indiquant qu'après un temps conséquent de simulation, nous pouvons obtenir le cycle total (rivière, cascade et lac rempli!).

Les résultats présentés ne sont alors pas totalement à l'image des possibilités de rendu de notre programme. Si l'on se concentre sur certaines parties (par exemple que la rivière), nous obtenons un résultat très satisfaisant avec un liquide qui se déplace et répond aux lois de la physique. Le résultat est observable ci-dessous, et nous observons effectivement un bon changement de couleur lorsque les particules (et donc l'eau) chute de la cascade vers le lac vide.

3 Futur travail

Initialement, l'objectif était d'obtenir une mesh transparent et ainsi de voir le fond de notre rivière et lac. Une amélioration serait alors de rendre ce mesh transparent et de rajouter des objets 3D au fond (par exemple de rochers) voire qui dépassent de la surface. La seconde partie permettrait de voir comment l'eau va esquiver ces rochers (en gérant les collisions de notre côté) et quel rendu sera obtenu. Cependant, lorsque l'on considère des détails si précis, il faudra drastiquement augmenter le nombre de particules en réduisant leur volume d'interaction. De plus cela demandera de reduire le pas de nos grilles pour nos Marching Cubes, qui ne tournera en aucun cas sur nos machines.

Un des deux camarades possédant un GPU, il aurait aussi été envisageable de faire tourner ce

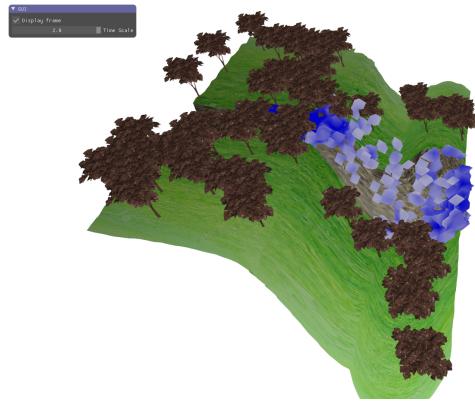


FIGURE 7 – Résultat final

programme sur le GPU pour obtenir des résultats plus fluides même avec un grand nombre de particules.

4 Conclusion

Pour résumé, nous avons mis au point un algorithme construisant une surface autour de particules animée par les règles physiques de simulation de fluides. Nous n'avons réussi qu'à animer un petit nombre de particules en un coup tout en gardant un résultat fluide.

Le résultat final est relativement satisfaisant en terme de comportement de l'eau ainsi que du traitement des couleurs mais trop lent pour une vraie application même pour un rendu "low poly", faible en nombre de polygones.

Cet exercice nous a cependant permis de nous familiariser avec la méthode des Marching Cubes utilisée plus largement que pour des fluides ; pour la génération de terrain à partir de bruit 3D ou encore pour la reconstruction d'objets 3D après un scan IRM ou autre.