PASCAL Antoine

Diminution des calculs d’un raymarcher.

Motivation

J’ai programmé mon premier moteur 3D il y a 3 ans et l’idée m’est venue de l’améliorer via des procédés appris depuis. Les moteurs 3D sont utilisés dans divers domaines de sciences ou de loisirs. Il peut être important de diminuer le coût de calcul quand les rendus peuvent prendre des heures.

Ancrage

Le raymarching permet de la conception de scènes 3D définis mathématiquement. Il permet donc la transformation d’objets mathématiques en objets sensibles.

Positionnement thématique

INFORMATIQUE (informatique théorique), INFORMATIQUE (informatique pratique)

Mots-clés

Mots-clés (en français)	Mots-clé (en anglais)
rendu de scène 3D	rendering 3D scenes
rayons	rays
fonction de distance signée	signed distance function (SDF)
arbres	trees
marche de rayons	raymarching
shader	shader

Bibliographie commentée

Le raymarching[1] est une technique de rendu d’objets 3D qui consiste pour chaque pixel d’une image, à calculer la couleur de celui-ci grâce aux SDF définissant chaque objet de la scène[2]. Cette technique est principalement utilisée pour calculer des nuages ou encore afficher des fractales 3D. Ce procédé est coûteux en calcul et ce coût augmente avec le nombre d’objets et le calcul d’éclairage. L’objectif est donc de trouver une manière de diminuer ces calculs. Certaines implémentations diminuent simplement le nombre maximum d’itérations pour chaque pixels, approximant ainsi le rendu. L’implémentation d’arbres de recherche dans un espace découpé (kdtree par exemple) peut également être utilisée[3]. Nous utiliserons pour implémenter le moteur, une API de rendu graphique, OpenGL qui permet de compiler des programmes en GLSL (OpenGL shader langage) [3] et de les exécuter par la composante graphique de l’ordinateur. Cette compilation et l’affichage se fera avec un programme en C. Nous utiliserons un type de shader particulier, les fragment shader qui permettent de calculer la couleurs de chaque pixel un à un dans une zone donnée (ici toute la fenêtre). Il est important de noter également que la mémoire utilisée par le fragment shader et le programme C sont différentes. Ainsi, effectuer le pré-traitement de la scène (comme son découpage qui doit se faire dans le programme C et non dans le shader qui est appelé pixel par pixel par exemple) nécessite un moyen de transmettre les

informations au shader et de les utiliser à l'arrivée. Cette méthode est moins efficace que de coder directement les paramètre de la scène dans le shader via des constantes car dans le second cas, la compilation va pré-calculer certaines choses et les informations seront stockées dans une autre partie de la mémoire à accès plus rapide. C'est pourquoi nous compareront les performances du moteurs uniquement avec le passage en paramètre de la scène pour obtenir des comparaisons acceptables et non biaisés, l'objectif étant seulement de diminuer les calculs.

Problématique retenue

Il s'agit ici d'implémenter un raymarching naïf et de comparer ses performances en coût de calcul avec des implémentations utilisant des algorithmes de recherche spatiales.

Objectifs du TIPE

1. Implémenter un premier moteur de raymarching (programme C et fragment shader GLSL) et une scène de référence.
2. Implémenter une manière de transmettre les informations de pré-traitement au shader.
3. Implémenter plusieurs algorithmes pouvant permettre de diminuer les calculs.
4. Comparer les résultats et conclure.

Références bibliographiques

[1] William Guimont-Martin, raymarching, https://willguimont.github.io/cs/2021/03/16/ray-marching.html
[2] Inigo Quilez, signed distance functions, https://iquilezles.org/articles/distfunctions
[3] Inigo Quilez, binary search, https://iquilezles.org/articles/binarysearchsdf/
[4] GLSL, https://learnopengl.com/Getting-started/Shaders