PASCAL Antoine

Diminution des calculs d’un raymarcher.

Motivation (50 mots)

Ayant programmé mon premier moteur 3D il y a 4 ans, l’idée m’est venue de l’améliorer via des techniques apprises depuis. Les moteurs 3D sont utilisés dans divers domaines scientifiques ou de loisirs. Il est donc important de diminuer le coût de calcul quand les rendus prennent beaucoup de temps.

Ancrage (26 mots)

En étudiant le raymarching et ses techniques d’amélioration, mon TIPE met en lumière la transformation d’objets mathématiques en scènes 3D grâce à des algorithmes plus efficaces.

Positionnement thématique

INFORMATIQUE (informatique théorique), INFORMATIQUE (informatique pratique)

Mots-clés (8 expressions

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

Bibliographie commentée (337 mots)

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] ainsi qu’une part de raytracing sur des formes simples qui se fait plus rapidement[4].

Nous implémenterons le moteur avec une API de rendu graphique, OpenGL qui permet de compiler des programmes en GLSL (OpenGL shader langage) [5] et de les exécuter par le processeur 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 une fenêtre).

Il est important de noter 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) doit se faire dans le programme C et non dans le shader, qui est appelé pixel par pixel. Il est donc nécessaire d'avoir un moyen de transmettre les informations au shader depuis le programme.[6]

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 du shader.

Problématique retenue (15 mots)

Comment diminuer le coût de calcul d'un raymarching pour simuler efficacement des scènes 3D ?

Objectifs du TIPE (60 mots)

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. Trouver un indicateur permettant de comparer le coût de calcul de 2 rendus.
4. Implémenter plusieurs algorithmes pouvant permettre de diminuer les calculs.
5. Comparer les résultats et conclure.

Références bibliographiques (6 références)

[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] victor's tech art blog, raytracing https://viclw17.github.io/2018/07/16/raytracing-ray-sphere-intersection
[5] GLSL, https://learnopengl.com/Getting-started/Shaders
[6] Doc OpenGL, UBO, https://www.khronos.org/opengl/wiki/Uniform_Buffer_Object