PASCAL Antoine

Diminution des calculs d’un raymarcher.

Motivation (21 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.

Ancrage (25 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 efficaces.

Positionnement thématique

INFORMATIQUE (informatique théorique), INFORMATIQUE (informatique pratique)

Mots-clés (5 expressions)

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

Bibliographie commentée (514 mots)

Les moteurs 3D sont utilisés dans divers domaines de loisirs comme les jeux vidéos ou scientifiques pour simuler des géométries non euclidiennes[1] par exemple.

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 fonctions de distance signées (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.

Pour calculer la couleur d’un pixel, un rayon est envoyé depuis la caméra dans une direction spécifique. Chaque pixel de l’écran correspond à un rayon. Le rayon progresse dans l’espace par étapes successives. À chaque étape, le raymarcher calcule la distance minimale entre la position actuelle du point de l’espace atteint par le rayon et la surface de l’objet le plus proche (distance estimée par une fonction de distance signée[1]). Si la distance calculée est très petite (en dessous

d’un seuil) ou nulle, cela signifie que le rayon a touché un objet. On peut donc afficher sa couleur sur le pixel.

Certaines implémentations diminuent simplement le nombre maximum d’itérations pour chaque pixel (c’est à dire le nombre d’étape dans l’avancée du rayon traversant ce pixel), approximant ainsi le rendu puisque le rayon aurait peut être touché un objet plus loin dans la scène.

L’implémentation d’arbres de recherche dans un espace découpé[2] peut également être utilisée pour éviter de calculer la distance entre le point courant et certains objet trop lointain inutilement.

De plus, une part de raytracing[3], méthode qui consiste directement à calculer le point d’intersection entre un objet et un rayon, sur des formes simples peut être implémentée, calculs qui se font plus rapidement que d’avancer pas à pas avec un raymarcher dans certains cas. Le raytracing est utilisé pour calculer l’avancée du rayon sur une plus grande distance d’un coup en approximant les formes de la scène par d’autres plus simple comme des sphères servant d’enveloppe aux objets. Une fois une intersection trouvée, le raymarcher reprend son calcul pour affiner le point d’intersection du rayon avec la vraie forme.

L’implémentation de moteur est faite par des API de rendu graphique comme OpenGL qui permet de compiler des programmes en GLSL[4] (OpenGL shader langage) et de les exécuter par le processeur graphique de l’ordinateur. Cette compilation et l’affichage se fait par des programmes exécutés par le processeur, en C par exemple. Les shaders GLSL permettant de faire des raymarcher sont les fragment shaders. Ils permettent de calculer la couleurs de chaque pixel un à un.

Les shaders, étant exécutés sur le processeur graphique, utilisent une mémoire graphique dédiée différente de la mémoire RAM dans laquelle se trouve les données du programme (celui en C par exemple) affichant le rendu. Ainsi, la communication de données entre les 2 programmes se fait grâce à des outils particuliers, les variables uniform[5], qui permettent ainsi de passer en paramètre du shader des données extérieurs.

Problématique retenue (15 mots)

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

Objectifs du TIPE (76 mots)

1. Implémenter un premier moteur de raymarching (programme C et fragment shader GLSL).
2. Implémenter des variables uniforms pour transmettre des informations de pré-traitement de la scène faites en C au shader.
3. Trouver un indicateur permettant de comparer le coût de calcul de 2 rendus.
4. Implémenter plusieurs algorithmes comme des découpages de l’espace avec des arbres (kdtree par exemple) ou un raytracer, pouvant permettre de diminuer les calculs.
5. Comparer les résultats et conclure.

Références bibliographiques (5 références)

[1] Rémi Coulon, Elisabetta A. Matsumoto, Henry Segerman, Steve J. Trettel, Ray-marching Thurston geometries, Cornell University, 13/01/2022 pages 10-16,
[2] Inigo Quilez, binary search, https://iquilezles.org/articles/binarysearchsdf/
[3] victor's tech art blog, raytracing, https://viclw17.github.io/2018/07/16/raytracing-ray-sphere-intersection
[4] Documentation OpenGL, Shader Compilation, https://www.khronos.org/opengl/wiki/Shader_Compilation
[5] Documentation OpenGL , Uniform Buffer Object, https://www.khronos.org/opengl/wiki/Uniform_Buffer_Object