UNIVERSITÉ DE SHERBROOKE

IFT 729

Conception de systèmes temps réel

Projet

L02

Génération procédurale de

planètes en temps réel

Travail présenté à :

Patrice Roy

Par :

Simon Lévesque-Gobeil (11 166 187)

Alex Provencher (11 173 124)

24 avril 2015

Voici un Comparatif des résultats obtenus avec les différentes versions de notre générateur de planète procédural. Les résultats ont été obtenus à l’aide du profileur (*profiler*) disponible dans Unity. Au cours de ces tests, les actions qui ont été effectuées par le joueur étaient de courir en continu autour de la planète tout en sautant.

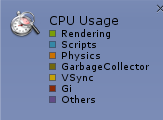


Figure - Légende

Dans les résultats qui vont suivre, nous allons voir que notre optimisation aura un impact sur les performances des scripts et de la collecte d'ordure donc les portions en bleu et brun vert.

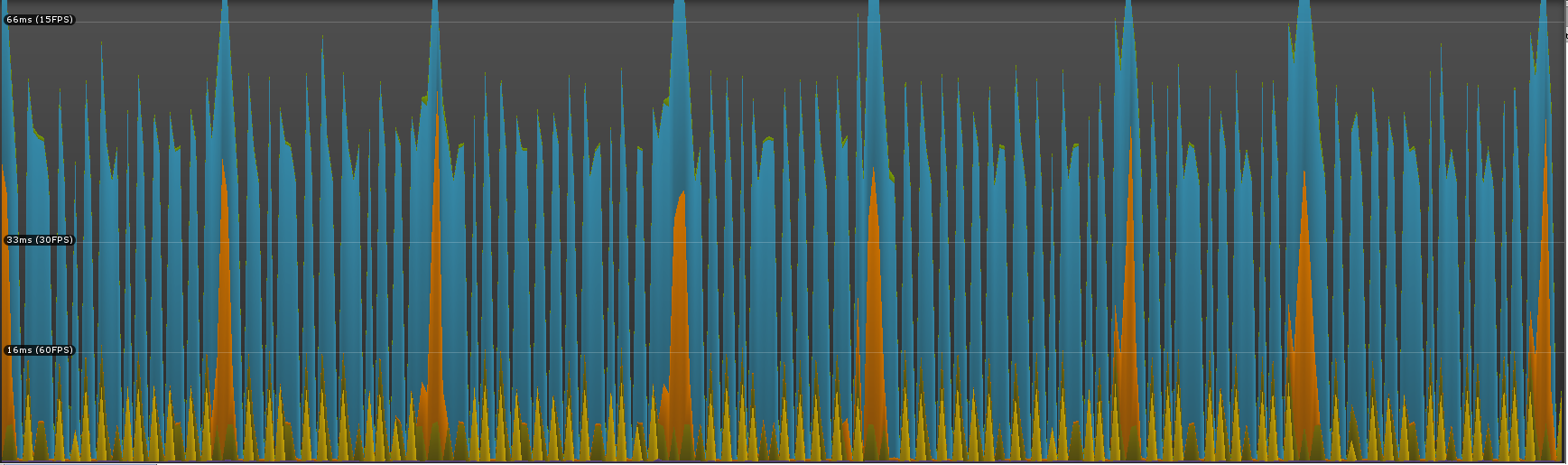


Figure - Solution séquentielle

Les résultats de la figure 2 ont été obtenus lorsque la gestion du *LOD* est effectuée séquentiellement. Les trois seuils du diagramme sont 60 fps, 30 fps et 15 fps (de bas en haut), nous pouvons donc clairement voir que nous ne respectons pas nos critères de 30 fps.



Figure - Solution avec *threads*

La figure 3 représente les résultats que nous avons obtenus lors de la génération en parallèle de la planète. Pour la majeure partie, cette solution semble respecter nos critères de temps réel. De plus, la collecte d'ordure occupe beaucoup moins souvent. Les fluctuations orange survenaient lorsque le joueur sautait ou tombait.

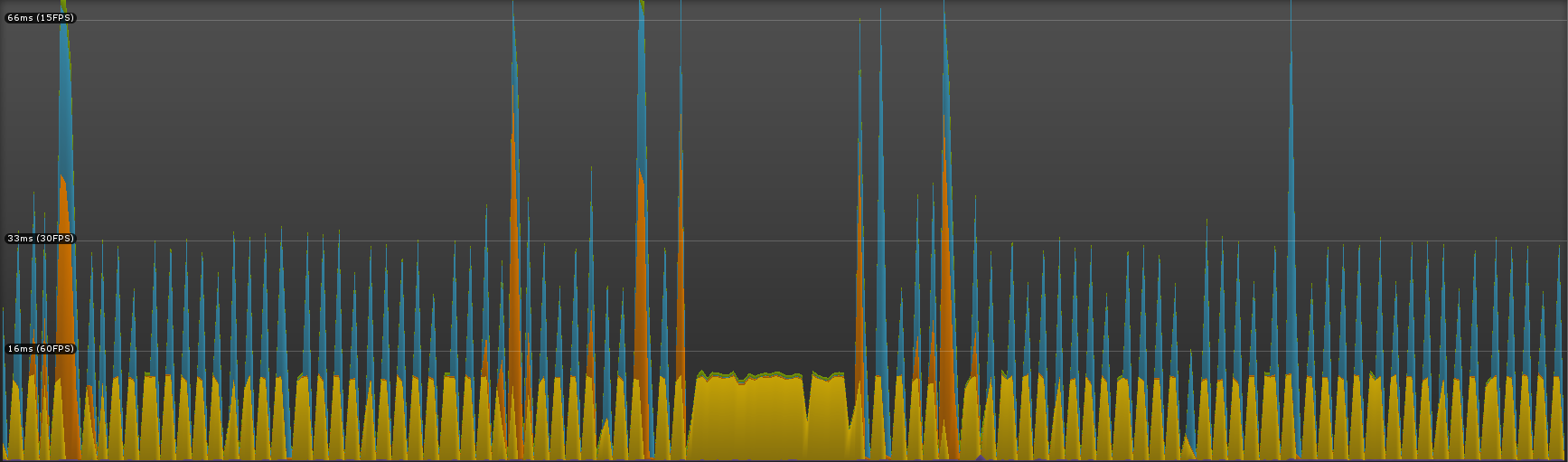


Figure - Threads et fps minimal de 30

Dans ce diagramme nous pouvons observer les résultats que l'on obtient lorsque l'on inclut à la solution la validation du fps. Cela a pour effet que lorsque le fps est inférieur à 30 la régénération du monde est interrompue. Cette interruption est perceptible dans la figure par la zone qui n'a pas de bleu.



Figure - *Threads* et fps minimal de 40

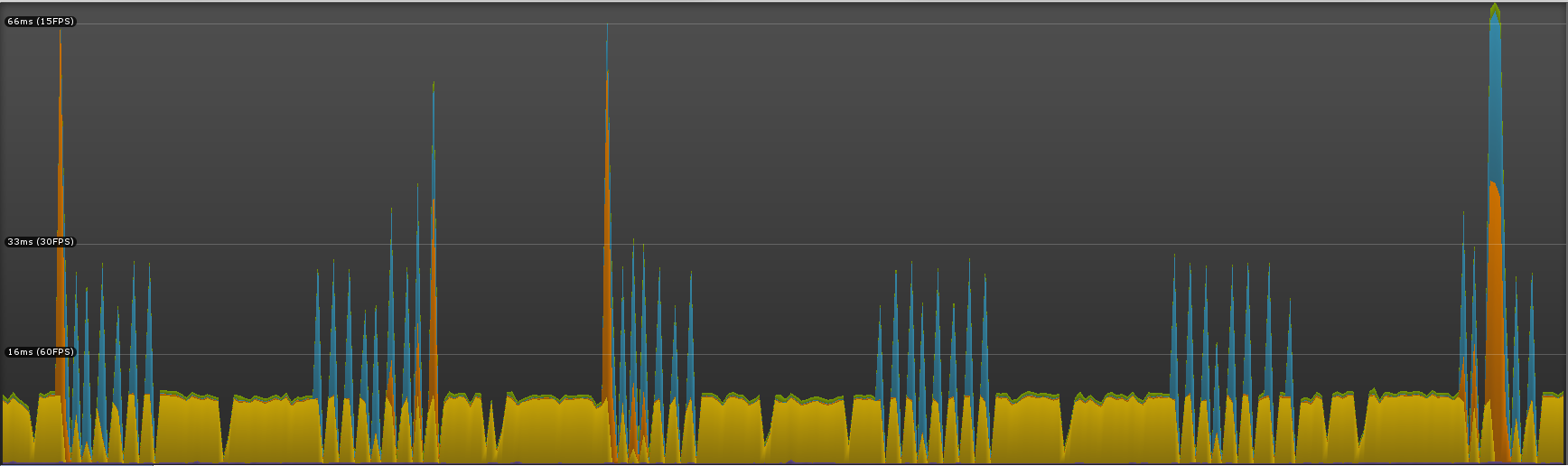


Figure - *Threads* et fps minimal de 50

Dans les figures 5 et 6, nous observons les résultats obtenus lorsque nous augmentons le critère de validation à 40 et 50 fps. Les zones d'arrêt sont beaucoup plus présentes, puisque la machine ne parvenait pas à respecter ces critères.

Pour ce qui est de la seconde partie du livrable, dans le code source qui vous a été envoyé précédemment les classes impliquées lors de cette optimisation sont les suivantes (Assets/Scripts/World).

VoxelWorld:

Sépare le calcul de la planète sur six MeshManager.

MeshManager:

Gère les différents dictionnaires d'objets concurrents, le recyclage des objets et initialise le premier MeshCalculator qui initialise par la suite quatre autres MeshCalculator (haut gauche, haut droite, bas gauche, bas droite).

MeshCalculator:

Chaque calculateur ajoute une tâche de créer sa portion de la grille au *thread pool*.

ConcurrentDict:

Dictionnaire d'objet à accès concurrent.