Memo

# Le système de particules

L’éditeur de particules était cascade

Maintenant avec Unreal 5 ils ont implémenter Niagara

Question : Est-ce que cascade et Niagara fonctionnent ensemble ?

## Optimisation

* Technique du Spritesheets
* Technique Flipbooks
* shaders simplifiés qui nécessitent moins de calculs par pixel
* Utilisé des matériaux instanciés (pas besoin de recompiler les matériaux)
* Configuration du LOD manuellement

Optimisation Mémoire

**Réduire la taille des textures** : Utilisez des textures aussi petites que possible sans sacrifier la qualité nécessaire à l'effet visuel désiré.

**Utiliser des matériaux optimisés** : Créez des shaders plus simples pour les particules, car des shaders complexes peuvent être coûteux en termes de performance.

**Atlas de textures** : Regroupez plusieurs petites textures dans un seul atlas pour réduire les changements de contexte de dessin, ce qui peut améliorer la performance.

**Contrôle de la surcharge** : Évitez les superpositions excessives de particules qui peuvent non seulement consommer de la mémoire, mais aussi affecter les performances graphiques.

**Pools de particules** : Envisagez l'utilisation de pools de particules pour réutiliser les systèmes de particules, ce qui peut réduire la charge sur la mémoire.

**Nettoyage et gestion de la durée de vie** : Assurez-vous que les particules sont correctement détruites ou recyclées après leur durée de vie pour libérer la mémoire.

# Le Garbage collector (GC)

**Gestion proactive :** Utilisez des profils de mémoire pour identifier les objets qui ne sont plus nécessaires et qui peuvent être nettoyés.

**Réduire la pression sur le GC :** Minimisez l'allocation de mémoire pendant les moments critiques, comme les séquences d'action ou les rendus complexes, pour éviter les pics d'utilisation du GC.

**Utilisation de UPROPERTY :** Assurez-vous d'utiliser le macro UPROPERTY pour les références d'objets dans les classes d'Unreal Engine pour que le GC puisse les suivre correctement.

**Destruction explicite :** Dans certains cas, il peut être judicieux de détruire manuellement des objets particulièrement volumineux ou nombreux dès qu'ils ne sont plus nécessaires pour alléger la charge sur le GC.

**Monitorer les générations :** Unreal Engine divise les objets en différentes générations pour le GC. Comprendre comment cela fonctionne peut vous aider à mieux contrôler le timing et l'impact du GC.

# Améliorer les temps de chargement

**LOD (Level of Detail) :** Ajuster les LOD pour les modèles et textures afin de charger des versions moins détaillées lorsque les objets sont éloignés.

**Occlusion Culling :** Ne pas rendre les objets qui sont bloqués par d'autres objets et donc hors de vue du joueur, ce qui économise les ressources.

**Level Streaming :** Charger et décharger dynamiquement des parties du monde de jeu pour que seul le contenu nécessaire soit chargé en mémoire.

**Optimisation des Textures :** Utiliser des textures compressées et de taille appropriée pour réduire l'espace mémoire nécessaire.

**Éclairage et Baking :** Précalculer l'éclairage pour les scènes statiques peut réduire le besoin de calculs d'éclairage en temps réel.

**Optimisation des Ombres :** Ajuster la résolution et la portée des ombres pour éviter des calculs inutiles.

## Conseils additionnels :

**Pooling d'Objets :** Réutiliser des objets qui sont fréquemment créés et détruits, comme les balles dans un jeu de tir ou les débris dans une explosion, pour réduire la charge sur le GC et les temps de chargement.

**Compression des Assets :** Utiliser la compression des assets pour réduire la taille des fichiers sur le disque et accélérer les temps de chargement.

**Chargement Asynchrone :** Charger les assets en arrière-plan pendant que le jeu est déjà en cours pour éviter les temps d'attente pour le joueur.

**Optimisation des Scripts et Blueprints :** S'assurer que les scripts et les blueprints sont bien optimisés pour ne pas retarder le chargement du niveau.

# Texture et shader

## Bonne pratique

**Résolutions de textures adaptées :** Comme vous l'avez mentionné, il est essentiel d'adapter la résolution des textures en fonction de leur utilisation. Par exemple, des textures de haute résolution pour des objets importants vus de près par le joueur, et des résolutions plus basses pour les objets d'arrière-plan ou moins importants.

**Matériaux instanciés :** Utilisez des matériaux instanciés pour pouvoir modifier les propriétés des matériaux en temps réel sans avoir à créer de nouvelles instances de matériaux pour chaque variation.

**Optimisation des shaders :** Écrivez des shaders aussi simples que possible et évitez les calculs superflus dans les shaders qui sont exécutés de nombreuses fois par pixel.

**Subsurface Scattering :** Si vous avez besoin d'effets de subsurface scattering (par exemple pour la peau), utilisez-les avec parcimonie car ils peuvent être coûteux.

**Utilisation de Mipmaps :** Les mipmaps sont des versions précalculées de textures à différentes résolutions et peuvent améliorer les performances en réduisant la charge de la texture lorsqu'elles sont vues de loin.

**Profiling :** Utilisez les outils de profiling d'Unreal Engine pour identifier les matériaux et shaders qui sont les plus gourmands en ressources et optimisez-les en conséquence.

# Outils de profilage

**Unreal Insights :** C'est un système de profilage et d'analyse qui permet de collecter et d'afficher des données détaillées sur le comportement d'un jeu en cours d'exécution. Il peut aider à identifier les goulots d'étranglement de performance en présentant les données de timing pour les cadres, les tâches d'arrière-plan, et plus encore.

**Stat Commands :** Ce sont des commandes que vous pouvez taper directement dans la console du jeu pour afficher en temps réel diverses statistiques de performance, telles que les taux de trame (fps), le temps de rendu, la mémoire utilisée, etc.

**Profiler :** Un outil qui permet de visualiser en détail où le temps de processeur est dépensé dans votre jeu. Il vous montre des graphiques en temps réel et des captures de profilage pour les cadres individuels.

**GPU Visualizer :** Cet outil aide à décomposer les coûts de rendu sur le GPU, permettant aux développeurs de voir précisément combien de temps est consacré à rendre chaque élément du jeu.

**Memory Profiler :** Un outil pour examiner l'utilisation de la mémoire du jeu. Il aide à identifier les fuites de mémoire, les allocations excessives et d'autres problèmes liés à la mémoire.

# Audio

**Object pooling :** L'utilisation d'un pool d'objets pour gérer les ressources audio est une bonne pratique, car elle peut aider à minimiser le surcoût associé à l'instanciation et à la destruction fréquentes d'objets audio, comme les effets sonores dans un environnement de jeu dynamique.

**Compression audio adaptée :** Choisissez le bon niveau de compression pour vos fichiers audio pour équilibrer la qualité sonore avec l'utilisation de la mémoire.

**Culling audio :** Utilisez l'occlusion audio et la priorisation pour éviter de jouer des sons qui ne sont pas perceptibles par le joueur, soit parce qu'ils sont trop éloignés, soit parce qu'ils sont bloqués par d'autres objets.

**Utilisation de Sound Cues :** Les Sound Cues permettent de créer des sons complexes et de contrôler leur logique de lecture, comme le mélange de pistes, la randomisation et les conditions de lecture.

**Streaming audio :** Pour les longues pistes audio, comme la musique de fond ou les dialogues, envisagez de les streamer depuis le disque au lieu de les charger entièrement en mémoire.

**Limitez le nombre de voix simultanées :** Trop de sons joués en même temps peuvent non seulement devenir cacophoniques pour l'utilisateur, mais aussi lourds pour le système. Mettez en place une limite et priorisez les sons importants.

**Profiler audio :** Utilisez les outils de profilage d'audio d'Unreal Engine pour identifier et optimiser les performances audio.