Moteurs de jeux – Communication, calcul et gestionnaire de ressources

Benoit Lange benoit.lange@inria.fr  http://lange.xyz @tyracus

# Objectifs

Le but de ce TP est de mettre en place plusieurs composants primordiaux pour un moteur de jeux. Ainsi, nous allons nous intéresser à plusieurs tâches :

* Mettez en place des communications client/server
* Accélérez les calculs réalisés
* Mettez en place un gestionnaire de ressources

**Bonus :**

* Restructurez votre application en SOA
* Ecrivez vos kernels en SPMD (avec ISPC)
* Simulez la création d’une rivière
* Chargez plusieurs autres types de modèles 3D
* Améliorez le gestionnaire de ressources

# Gestionnaire de version

Clonez le quatrième TP :

git clone http://lange@www.lange.xyz/GMIN317/TP/groupe<#>/tp4.git

Au plus tard la semaine prochaine (minuit), vous devrez rendre un compte rendu de ce TP sur votre espace GIT, ainsi que votre code source.

# Mettez en place des communications client/server

La semaine dernière, vous avez mis en place une décomposition de votre application en plusieurs sous-fenêtres. Nous allons réutiliser cette approche pour étudier la communication client/serveur. Afin de simplifier l’implémentation, nous resterons sur la même application.

Pour chaque fenêtre, nous allons intégrer un thread séparé avec la fonction : ***QThread***.

Une fenêtre sera utilisée comme serveur (avec la fonction ***QTcpServer***) et les autres fenêtres seront simplement client, en utilisant la classe Qt : ***QTcpSocket***.

Le but est de transférer un message à chaque « application » cliente, en fonction d’un temps écoulé (environ 5 minutes) sur l’application serveur ; pour cela vous devrez utiliser la fonction ***QTimer***.

Nous allons simuler pour chaque fenêtre une période différente de l’année (les 4 saisons). Le message devra ainsi contenir la saison courante de la fenêtre.

Puis, faites circuler les différentes saisons entre les fenêtres.

Vous devez créer une fonction qui initialise le serveur et réaliser un ***connect***entre un signal : ***newConnection()*** et un slot que vous devez écrire : ***sendSeason()***.

Dans votre header, rajouter le *tag* ***slots***au niveau de votre fonction***sendSeason()***. Cette fonction devra envoyer un message à intervalles réguliers à vos différents clients.

Pensez à prendre le temps de faire un commit et un push sur git.

**Accélérez les calculs réalisés**

Nos saisons seront caractérisées par différents comportements.

L’été – une lumière vive, une couleur vive au sol

L’automne – pluie et une couleur rouge/orange au sol

L’hiver - chute de neige et une couleur blanche au sol

Printemps - des couleurs vives au sol

Rajoutez dans votre fichier ***.pro*** les commandes suivantes :

QMAKE\_CXXFLAGS+= -fopenmp

QMAKE\_LFLAGS += -fopenmp

Pour vos fenêtres, vous allez devoir ajouter deux nouvelles données :

Une structure pour gérer la pluie, neige (il s’agit de particules)

Et une pour chaque sommet de notre terrain et un attribut de couleur.

Pour la chute des particules, vous allez devoir les habiller à l’aide de ***glPoint***.

Accélérez le calculs de chute de particules en utilisant le parallélisme de boucles ***FOR.***

**Attention : la pluie ne traverse pas le terrain, elle disparaît.**

Pensez à prendre le temps de faire un commit et un push sur git.

**Mettez en place un gestionnaire de ressources**

Dans le TP2, nous avons chargé une ***height map*** (carte de profondeur). Pour ce faire, nous avons chargé une image.

Ici, nous allons mettre en place un gestionnaire de ressources qui servira à plusieurs choses :

* Sauvegarder la saison de nos différentes fenêtres (chaine de caractères),
* Choisir le nombre de fenêtres à visualiser de type client (entier),
* Permettre le chargement dynamique de terrain

(Composé d’un nombre de sommets caractérisés par : X, Y, Z et une couleur)

* Charger des objets 3D sur le terrain

(Avec comme information : une liste de sommets, un nom d’objet et leurs coordonnées)

Réalisez différentes classes qui permettent de stocker ces différentes données, pour cela réutilisez le plus possible les mêmes structures de données.

Pensez à faire un commit et un push sur git pour chaque élément du gestionnaire de ressources.

Afin de vous aider, pour le chargement des objets 3D, nous allons utiliser des formats simples : OFF, STL, PLY. La spécification de ces formats est disponible à cette adresse :

http://paulbourke.net/dataformats/

Réalisez votre chargeur de données pour ces 3 types de modèles et permettez leur d’être placés sur votre scène.

**Attention : pour cette partie, vous allez devoir mettre en place un système uniforme de coordonnées pour le placement des objets.**

# Compte rendu

Présentez toutes vos nouvelles fonctionnalités

Expliquez les points que vous n’êtes pas arrivé à réaliser et pourquoi.

Expliquez votre démarche de développement.

Présentez votre structure de données.

Expliquez comment vous vous y prendriez pour les parties bonus.

# Bonus

* Restructurez les données compatibles en utilisant le paradigme SOA
* Réécrivez vos opérateurs de transformations en utilisant des kernels SPMD, et une approche multi-cœur si possible.
* Apprenez à gérer vos particules

Proposez une approche pour accumuler vos particules, et les transformer en surface (rivière, tas de neige)

* Localisez les effets climatiques

Affichez un rendu différent pour chaque fenêtre, mais gérer les mêmes conditions météo.

* Ajoutez d’autres formats 3D à votre gestionnaire de ressources
* Créez un gestionnaire de villages automatique
* Ajoutez à votre gestionnaire de ressources d’autres données

Par exemple : un format ***debug***, un format release, le chargement de texture pour le terrain.