Troma

Rapport de soutenance 2

5 mai 2014

Thibault *Dethi* **Deutsch** (*deutsc_t*)
Rémy *Shadows* **Bernier** (*bernie_r*)
Marc *Leshlague* **Fresne** (*fresne_m*)
Anthony *AnthonySG* **Belthier** (*belthi_a*)

Table des matières

Introduction

Tâches	Thibault	Rémy	Marc	Anthony
Modélisation			X	
Moteur graphique	X		X	
Moteur physique		X		X
Menu		X		X
Réseau	X		X	X
Gameplay	X	X		
Audio		X		X
Site web	X	X		

FIGURE 1 – Répartition des tâches de la deuxième soutenance

Légende :
- Rouge : Commencé
- Orange : Avancé
- Vert : Terminé

I. Modélisation

1. La conception

Après la soutenance 1 nous avions déjà une première carte destinée au mode solo du jeu. Cependant nous étions limités au niveau de la taille de celle-ci. En effet comme expliqué dans le rapport précédent nous avions utilisé une heightmap pour la gestion du sol et du relief. L'implémentation que nous avions choisie nous limitait à un carré de 513 par 513 unités. Afin d'augmenter la surface nous avons donc dans un premier temps construis un tableau à deux dimensions contenant 9 heightmap (3×3). La taille est donc passée à 1536 par 1536 unités. Ce premier changement a entrainé une modification de la carte solo de manière superficielle, mais a permis de poser les bases pour la conception de la carte multi-joueurs.

Le décor de la seconde carte de notre jeu (multi-joueurs) à pour cadre un milieu urbain, un petit village européen. Nous nous sommes inspirés d'un quartier de Cracovie (Pologne) pour la disposition des bâtiments.

TODO : Figure plan original Cracovie VS plan de la map multi

Comme visible ci-dessus il existe de nombreuses différence, notamment la voie de chemin de fer que nous avons rajouté.

2. La modélisation

Une fois le plan conçu, Marc a pu commencer la modélisation toujours à l'aide de Blender et Gimp. Dans l'ensemble les éléments des décors sont toujours basés sur des formes géométriques simples, cependant il dispose d'un peu plus de détails, tels que des portes, des fenêtres en relief.

Afin de facilité la création de l'ensemble de bâtiments, la construction de la ville s'est déroulé en plusieurs étapes. Dans un premier temps la création et la mise en place sur la carte des murs des bâtiments texturés (grâce à l'UV mapping expliqué dans le rapport précèdent). Puis l'ajout sur chaque bâtiment du toit, des fenêtres et des portes. Plusieurs modèles de fenêtres et de portes ont été réalisé au préalable (eux aussi texturés) et dupliquer sur chaque bâtiments.

A ce moment, tous les modèles étaient composés de plusieurs « meshes » (objets en français), mais chaque modèle avait besoin de plusieurs textures.

3. Le normal mapping

En parallèle de la modélisation, Thibault s'est attelé à l'implémentation d'un shader afin d'améliorer le rendu.

> "Un shader (le mot est issu du verbe anglais to shade pris dans le sens de « nuancer ») est un programme informatique, utilisé en image de synthèse, pour paramétrer une partie du processus de rendu réalisé par une carte graphique ou un moteur de rendu logiciel.", *Wikipédia*

La technique de normal mapping ressemble à la technique de l'heightmap que nous vous avions présenté lors de la soutenance 1. Cependant alors que le shader utilisé pour l'heightmap déforme la géométrie de l'objet sur lequel il est appliqué, le normal mapping feind cette déformation. En effet, la déformation n'est que visuelle dans le cas du normal mapping.

Cette technique permet sans ajouter de polygones supplémentaires aux modèles, de donner l'impression de relief. Afin de permettre cette modification, le shader se base sur une image en couleur (RVB).

TODO : figure exemple normal map

TODO : detail explication

L'utilisation de ce shader a nécessité la réunification des textures des modèles. Pour ce faire, Marc a du fusionner tous les meshes de chaque modèle pour modifier l'UV mapping, puis réunir les textures en une seule en suivant la carte UV.

Ensuite est arrivé l'étape de création des normal map. La technique utilisé habituellement consiste a réalisé deux modèles. Un dit Low poly (composé de peu de polygones) et un dit Hight poly (composé de beaucoup de polygones).

TODO : figure comparaison de deux modèles

Ensuite on superpose les deux modèles puis le logiciel de modélisation est capable de générer une normal map en calculant la différences de relief en chaque point entre les deux modèles. Cette technique offre une normal map de qualité, mais est relativement longue puisqu'il est nécessaire de réaliser deux modèles dont un très détaillé.

Heureusement, il existe une autre solution. A partir d'une texture il est possible grâce à un logiciel de retouche d'image, de généré en fonction des couleurs et de la luminosité, une normal map. Cette technique permet de réaliser rapidement une normal map si la texture de l'objet a déjà été réalisée, et les résultats sont plutôt convaincants. Nous avons donc opté pour la deuxième technique, moins chronophage.

N.B. Une autre technique permet l'illusion de relief et utilise aussi un shader : le Bump mapping. Cependant, l'image utilisé pour le bump mapping, a l'instar de celle utilisé pour l'heightmap est en noire et blanc. La modification du relief ne s'effectue que sur un axe (la normale a la surface concernée), le rendu est donc moins détaillé, c'est pourquoi nous opté pour le normal mapping.

TODO : figure de comparaison normal map bump http ://www.etude-blender.fr/uv-bump-normal-map.php voir image

4. Les animations

5. Pour la prochaine soutenance

Jusqu'à la première soutenance, les animations 3D n'avait pas fait leurs apparitions dans notre jeu.

Pour cette deuxième soutenance, quelques animations sont visibles. La première est celle des cibles, qui se "lèvent" comme si un ennemi apparaissait et qui se "couchent" lorsque le joueur leur tire dessus. Les suivantes concernent les personnages en lui-même, les mouvements tels que la marche ou la mise accroupie.

Pour obtenir les animations, deux techniques sont possibles. Soit réaliser le modèle et "programmer" manuellement les mouvements des différents meshes depuis Visual Studio. Soit réaliser les animations depuis blender et les exportés pour les "lire" ensuite.

Par commodité Marc a choisi d'utilisé la seconde méthode. La création d'une animation en utilisant des "bones" s'effectue en trois étapes : le "rigging", le "skinning", et enfin l'enregistrement de l'animation.

Tout d'abord, pourquoi utiliser une armature composée de "bones" ? Cela permet d'animer des modèles complexes, le personnage par exemple, en modifiant le maillage en fonction du mouvement.

5.1 Première étape : le "riggin"

Le rigging consiste à créer une armature, un squelette, et à organiser une hiérarchie entre les os. Il s'agit de disposer les os de façon à être capable d'effectuer les mouvements désirés et d'indiquer la dépendance des os entre eux.

Un os peut avoir un parent ou non, et peut lui-même être père ou non. Si l'os est père d'un autre alors le mouvement du père entraine le mouvement du fils.

TODO : figure rigging

5.2 Deuxième étape : le "skinning"

Le skinning consiste à affecter à chaque os les points du maillage sur lesquels il aura une influence, c'est la notion de weight (le poids d'influence). La valeur de ce poids est comprise entre 0 et 1, 0 étant l'influence nulle et 1 l'influence maximale.

Blender offre plusieurs possibilités sur ce point :

- Soit affecter automatiquement les points du maillage aux os en fonction de leur "envelope", c'est à dire leur rayon d'action.
- Soit affecter automatiquement les points du maillage aux os en fonction de leur position par rapport aux os.

– Soit manuellement. Là encore deux solutions sont possibles :
  – Sélection de points et affectation a un «vertex group », comprendre groupe de point influencé par l'os
  – Selection par « Weight Painting », c'est le même principe sauf qu'au lieu de sélectionner les points du maillage, il suffit de « peindre » la surface qui dépendra de l'os.

TODO : figure skinning selection maillage VS Weight Painting

N.B. Il est très important que tous les points dépendent d'au moins un os. Une solution pour vérifier que tous les points ont un poids et de parcourir les "vertex groups" en mode Weight Painting et de chercher une région resté bleue tout le long du parcours.

Les deux premières méthodes de skinning ne sont à utiliser que pour des modeles très simple. Marc n'a donc pas pu les utiliser.

5.3 Troisième étape : la création de l'animation

Une fois le squelette construit et les dépendances avec le maillage assignées, il faut maintenant définir les mouvements qui composeront l'animation.

Pour modifier le positionnement des os et que cela ait une influence sur le maillage il faut se placer en "pose mode". Blender permet de diviser l'écran en plusieurs zones pour pouvoir travailler sur différents aspects en même temps. Par défaut sous la vue 3D il y a un zone où le mode "TimeLine" est activé. Il s'agit d'un axe représentant le temps sur lequel on peut se déplacer, et servant à indiquer les différents paramètres du modèle en fonction du temps.

C'est grâce a cette "TimeLine" que l'on réalise une animation. Il suffit de se placer a un instant t et d'enregistrer les informations voulues pour les liées. Ces instants sont appelées KeyFrames.

Ainsi pour l'animation de la cible, à l'instant t1 la cible est relevée, à l'instant t2 la cible est baissée.

TODO : figure d'illustration

Grace à Blender ces deux instants suffisent pour que la cible se baisse et passe de la position-rotation de l'instant t1 a celle de l'instant t2 de manière progressive durant le temps qui sépare ces deux instants. En effet celui-ci va calculer les différents paramètres à appliquer au modèle à chaque instant pour que l'animation soit fluide.

Une fois l'action créée, il n'y a plus qu'à exporter le modèle.

5.4 Exploitation des animations dans XNA

Une fois le modèle et l'animation créés et exportés, il va falloir récupérer ces informations pour être capable de lancer l'animation dans XNA. Toutes les informations

concernant le modèle sont conservées sous forme de matrice (ex position des points) ou de tableau (ex poids des points, os). Par default le ModelProcessor d'XNA pour les fichiers .X ne récupère pas d'information concernant les animations. Il a donc fallu « l'étendre » pour récupérer les informations de skinning, les keyframes ainsi que la relation entre les bones.

Pour ce faire nous avons récupérer le sample nommé « Skinned Model » mis à disposition par Microsoft [1].

Une fois l'extension du Model Processor intégrés à notre code, nous étions en mesure de lancer des animations et de les afficher.

1. http://xbox.create.msdn.com/en-US/education/catalog/sample/skinned_model

II. Moteur graphique

1. Les différentes shader

TODO : lumière, normal map, et autre ci-dessous

2. Ciel

L'environnement de notre jeu se composait lors de la première soutenance d'un terrain et de modèle. Nous souhaitions rendre le ciel plus agréable à regarder que la couleur bleu par défaut d'XNA. Notre première idée a été d'utiliser un skydome. Il s'agit d'un modèle 3d prenant la forme d'une demi-sphère sur laquelle est apposé sur la partie intérieure, une image de ciel.

TODO : figure skydome et texture ciel

Après quelques test, bien que le rendu soit meilleur qu'un fond bleu uni, nous avons décidé de laisser cette idée de côté. D'une part parce que la délimitation entre le skydome et le sol posait problème. Et d'autre part car nous avions un ciel statique. Nous avons donc fait machine arrière et retiré le skydome. Nous nous contenterons du fond uni, auquel Thibault a ajouté des nuages.

TODO : ciel dynamique

3. Optimisations

Afin d'optimiser une foos de plus notre moteur de jeu, nous avons réecris nos shaders afin qu'ils soient au maximum compatible avec les preshaders. Les preshaders sont une optimisation du compilateur HLSL qui precalcule les constantes des shaders sur le processeur. Ainsi les calculs utilisant les même valeurs pour l'ensemble des shaders ne sont calculés qu'une seule fois.

Une autre optimisations a été réalisés au niveau des shaders au niveau du passage des variables au shader. En effet, le passage de variables au shader implique un transfère des données de la mémoire vive vers la carte graphique. Ce processus fait ralentir les calculs qui doivent attendre le transfers des données. Ce processus est d'autant plus long quand il s'agit de faire passer des variables. Pour contrer ce problème, nous avons décidés de n'actualiser que les données qui varient. Ainsi les textures ne sont transférer au shader que lors de l'initialisation. Nous utilisons donc plus de mémoire graphique mais nos rendus sont accélérés.

TODO : optimisation shader + opti carte abandonnée

4. Pour la prochaine soutenance

III. Moteur physique

1. Pour la prochaine soutenance

IV. Menu

1. Pour la prochaine soutenance

V. Réseau

VI. Gameplay

1. Pour la prochaine soutenance

VII. Audio

1. Pour la prochaine soutenance

VIII. Site web

1. Pour la prochaine soutenance

Conclusion

Table des figures