Plateforme d’animation et de rendu

Fabrice Lamarche
ESIR / Université de Rennes 1

Contenu

Ce document a pour objectif de vous décrire la structure de la plateforme d'animation et de rendu qui vous est fournie dans le cadre de vos travaux pratiques.

1 Dépendances de la plateforme

La plateforme possède plusieurs dépendances, toutes compilables sous au moins deux systèmes : Windows et Linux. Ces dépendances sont les suivantes :

- **freeglut** (Free OpenGL Utility Toolkit) : une bibliothèque permettant d'ouvrir une fenêtre de rendu OpenGL et fournissant des interactions clavier souris simplistes.
- **glew** (OpenGL Extension Wrangler Library) : une bibliothèque permettant d'accéder simplement aux extensions OpenGL.
- **glm** (OpenGL mathematics) **:** une bibliothèque fournissant des classes et fonctions utiles pour tous les aspects mathématiques d'OpenGL. Les classes et fonctions fournies sont en partie basées sur celles disponibles dans GLSL.
- **lib3ds** : une bibliothèque permettant de lire des fichiers graphiques au format 3DS.
- **Assimp** : une bibliothèque permettant de lire différents formats de fichiers 3D.
- **SOIL** (Simple OpenGL Image Library) : une bibliothèque permettant de lire des images encodées dans différents format et permettant de les transformer simplement en textures OpenGL.
- **Intel TBB** (Intel Threading Building Blocks) : une bibliothèque permettant de facilement et rapidement créer des applications C++ utilisant le multithreading.
- **FCL (flexible collision library) :** une bibliothèque gérant la détection de collision entre objets géométriques.

Pour pouvoir utiliser la plateforme, vous devez posséder des versions compilées (pour votre système) de ces bibliothèques.

2 Organisation du projet

2.1 Conventions de nommage de fichiers, classes etc...

Le projet a été conçu en respectant un certain nombre de conventions du point de vue du nommage des fichiers et de l'arborescence des répertoires :

- La déclaration de la classe X se trouve dans un fichier nommé X.h
- Si la classe X a été décrite dans un espace de nommage Y, le fichier X.h est dans un répertoire nommé Y.
- L'implémentation de cette même classe (si elle n'est pas faite dans le fichier header), se trouve dans le répertoire Y/src dans le fichier X.cpp.

Conventions de nommage des espaces de nommage, classes et attributs :

- Le nom d'un espace de nommage commence toujours par une majuscule et une nouvelle majuscule est utilisée pour séparer les mots en cas de nom composé de plusieurs mots.
- Les noms de classe commencent toujours par une majuscule et une nouvelle majuscule est utilisée pour séparer les mots.
- Les noms d'attributs de classes sont toujours précédés de « m_ » (raccourci pour member) et une majuscule est utilisée pour séparer les mots.
- Les noms de méthodes commencent toujours par une minuscule et une majuscule est utilisée pour séparer les mots.

2.2 Les différents espaces de nommage

La plateforme utilise 8 espaces de nommages, regroupant les classes par fonctionnalités.

- **Animation** : ensemble de classes en charge de la gestion des animations.
- **Application** : ensemble de classes en charge de la gestion / création d'une application de test. La gestion des applications de test a été conçue pour vous permettre de tester tous vos TP au sein de la même application en choisissant, au lancement de l'application, le cas de test que vous souhaitez lancer.
- **gl3** [*OpenGL 3.3*] : ensemble de classes utilisées pour les TPs en OpenGL 3.3.
- **HelperGL** [*OpenGL 1.0-2.0*] : ensemble de classes facilitant la vie du programmeur OpenGL : gestion de buffers, chargement de textures et de fichiers 3D, représentation d'une géométrie…
- **Math** : ensemble de classes modélisant des outils mathématiques tels que les vecteurs, les quaternions, les matrices…
- **Motion planning** : ensemble de classes conçues pour les TPs sur la planification de mouvement.
- **Utils** : ensemble de classes utilitaires, en rapport avec les structures de données.
- **SceneGraph** [*OpenGL 1.0-2.0*]: ensemble de classes permettant de représenter la structure d'un graphe de scène.
- **System** : ensemble de classes / fonctions permettant de s'abstraire du système d'exploitation. Ces fonctionnalités possèdent une implémentation pour Windows et une implémentation pour Linux, la bonne version étant automatiquement compilée en fonction du système sous lequel votre application est compilée.

Comme vous pouvez le remarquer, plusieurs versions de code

3 Création d'une application OpenGL avec la plateforme

La création d'une application OpenGL dans la plateforme fournie s'effectue en deux étapes. D'une part, vous devez créer une classe héritant de la classe Application::Base (dans le fichier Application/Base.h) et d'autre part, vous devez enregistrer votre application dans le fichier main_application.cpp afin que celle-ci apparaisse dans le menu de lancement. Les sections suivantes décrivent les deux étapes de ce processus.

3.1 Création d'une nouvelle application

Pour créer une nouvelle application, vous devez suivre le processus suivant :

1. Créer une classe héritant de la classe Application::Base. Cette classe devrait préférentiellement se trouver dans l'espace de nommage Application et respecter les conventions de la section 3.1. Cette classe doit mettre à disposition un constructeur sans paramètre.
2. Dans cette nouvelle classe, implémentez la méthode abstraite
   ```
   virtual void initializeRendering() = 0 ;
   ```
   Cette méthode est appelée à la création de votre application et a pour but de vous permettre d'initialiser cette dernière. Il peut s'agir de charger vos fichiers de géométrie, créer votre graphe de scène, initialiser vos animations… Il est à noter qu'avant l'appel à cette méthode, une première initialisation est gérée par la classe Application::Base. Cette dernière initialise OpenGL, GLEW, crée une fenêtre de rendu GLUT et configure OpenGL en autorisant le Z-Buffer, l'éclairement, le back face culling, la normalisation des normales… Ces initialisations devraient être suffisantes pour la plupart de vos applications.

3. Dans cette nouvelle classe, implémentez la méthode abstraite
   `virtual void render(double dt) = 0 ;`
   Cette méthode est appelée à chaque nouveau rendu d'image. Il s'agit donc ici de gérer vos animations ainsi que les affichages OpenGL adéquats. La paramètre *dt* vous fournit le temps en secondes, qui s'est écoulé depuis le dernier appel à la méthode de rendu et vous permet donc de faire évoluer l'état de vos animations en fonction du temps qui passe.

Ce processus vous permet de rapidement créer une application minimale pour tester vos TPs. Par défaut la caméra est placée en (0,0,0) et regarde suivant –Z.

Vous pouvez vous référer à la section 3.3 pour découvrir les autres fonctionnalités qui sont mises à votre disposition (gestion du clavier, de la souris, des menus…).

3.2 Référencer votre nouvelle application

Une fois votre application créée, comme décrit dans la section précédente, il faut la référencer afin qu'elle apparaisse dans le menu de lancement. Pour ce faire, ouvrez le fichier *main_application.cpp* et ajoutez la ligne suivante :

```
Application::ApplicationSelection::registerFactory<Application::Test>("Hello world") ;
```

Cette commande permet d'enregistrer l'application décrite par la classe Application::Test ainsi que sa description qui est en l'occurrence « hello world ». Une fois cette commande ajoutée, l'application apparaitra dans le menu de lancement (sous la forme de sa description) et pourra être lancée depuis ce dernier.

3.3 Interaction avec l'application

La classe Application::Base qui sert de classe mère à toute application met à votre disposition d'autres fonctionnalités vous permettant de gérer les interactions de l'utilisateur : gestion de menu contextuel, gestion du clavier et de la souris.

3.3.1 Gestion des menus

L'application dispose, par défaut, d'un menu vous permettant d'afficher en haut à gauche de l'écran le nombre d'images par secondes et le nombre de millisecondes prises par le rendu de la dernière image. Ce menu est accessible via la méthode de Application::Base suivante :

```
Menu * getMenu() ;
```

Ajout d'un élément. Cette méthode retourne un pointeur sur la classe gérant le menu contextuel nommée Application::Menu. Pour ajouter un élément dans le menu contextuel, vous disposer de la méthode suivante :

```
void addItem(::std::string const & name, const ::std::function<void ()> & callback);
```

Cette méthode vous permet d'ajouter un élément nommé *name* dans le menu et d'y associer une fonction de callback, autrement dit, une fonction qui sera appelée lorsque cet élément de menu sera sélectionné par l'utilisateur. Le paramètre associé à la fonction de callback (de type std::function[1])

[1] http://fr.cppreference.com/w/cpp/utility/functional/function

peut être une lambda fonction[2], une instance de foncteur ou encore un classique pointeur sur fonction. Cette fonction de callback ne doit pas prendre de paramètre.

Ajout d'un sous menu. Vous pouvez aussi ajouter un sous menu dans le menu principal. Pour ce faire, vous devez créer une instance de la classe Application::Menu qui possède un constructeur prenant le nom du menu en paramètre. Une fois cette instance créée, vous pouvez l'ajouter comme sous menu en utilisant la méthode suivante :

```
void addSubMenu(Menu * menu);
```

3.3.2 Gestion du clavier

La classe Application::Base propose deux méthodes virtuelles qui sont appelée lors d'une interaction utilisateur avec le clavier. Ces méthodes sont possèdent les signatures suivantes :

```
virtual void keyPressed(unsigned char key, int x, int y) ;
virtual void keyReleased(unsigned char key, int x, int y) ;
```

Ces méthodes sont appelées lorsque qu'une touche est pressée et relâchée. Le paramètre *key* de ces méthodes fournit le caractère associé à la touche pressée / relâchée et (x,y) sont les coordonnées de la souris dans la fenêtre.

Pour que votre application puisse gérer les interactions clavier, il vous suffit de redéfinir ces méthodes et d'y associer le comportement souhaité. Pour facilement gérer l'état du clavier, La classe *Application::KeyboardStatus* vous est fournie. Cette dernière, couplée avec les deux méthodes présentées ci-dessus vous permet de conserver l'état du clavier et d'y accéder depuis votre application.

3.3.3 Gestion de la souris

La gestion de la souris s'effectue elle aussi en redéfinissant le comportement de certaines méthodes virtuelles de la classe Application::Base. Elles sont au nombre de trois (calquées sur les fonctionnalités offertes par GLUT).

```
virtual void mouse(int button, int state, int x, int y) ;
```

La méthode ci-dessus est appelée lorsque l'utilisateur appuie ou relâche un bouton de la souris. Le paramètre *button* fournit le bouton pressé : GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON ou GLUT_MIDDLE_BUTTON. Le paramètre state donne l'état du bouton : GLUT_UP ou GLUT_DOWN. Les coordonnées (x,y) correspondent aux coordonnées de la souris dans la fenêtre au moment de l'appui ou du relâchement du bouton.

```
virtual void mouseMotion(int x, int y)
```

La méthode ci-dessus est appelée lorsqu'au moins un bouton de la souris est pressé et que la souris est en mouvement. Les paramètres (x,y) fournissent alors les coordonnées de la souris dans la fenêtre de rendu.

```
virtual void mousePassiveMotion(int x, int y)
```

[2] http://en.cppreference.com/w/cpp/language/lambda

La méthode ci-dessus est appelée lorsqu'aucun bouton de la souris n'est pressé et que la souris est en mouvement. Les paramètres (x,y) fournissent alors les coordonnées de la souris dans la fenêtre de rendu.

4 La bibliothèque mathématique

Les objets mathématiques fournis se trouvent dans l'espace de nommage Math. Nous y retrouvons un certain nombre de classes décrites ci-dessous.

Classe Math::Interval<Type>. Il s'agit d'une classe de gestion d'intervalles numériques décrite dans le fichier « Math/Interval.h ». Type est le type de scalaire utilisé par l'intervalle. Cette classe met à disposition des opérateurs d'intersection et d'union, propose une méthode *random* permettant de tirer un nombre aléatoire dans l'intervalle ou encore une méthode *clamp* permettant de restreindre une valeur numérique dans l'intervalle. Vous avez aussi à votre disposition une fonction *Math::makeInterval* permettant de créer rapidement un intervalle.

Classe Vector<Type, dimension>. Il s'agit d'une classe de gestion de vecteurs fournie dans le fichier « Math/Vector.h » . Cette classe générique est paramétrée par le type de scalaire utilisé pour le vecteur ainsi que par le nombre de dimensions de ce dernier. La classe redéfini les opérateurs classiques d'addition, soustraction, multiplication et produit scalaire (opérateur de multiplication entre deux vecteurs). Des spécialisations de cette classe sont fournies dans le fichier « Math/Vectorf.h », il s'agit de Math::Vector2f, Math::Vector3f et Math::Vector4f.

Classe Matrix4x4<Type>. Il s'agit d'une classe de gestion de matrices 4x4 fournie dans le fichier « Math/Matrix4x4.h ». Cette classe vous permet de facilement décrire des transformations géométriques sous la forme de matrices homogènes. Elle dispose des opérateurs classiques de multiplication et addition de matrices ainsi que d'opérateurs permettant de consulter / modifier les éléments. Elle met aussi à disposition des méthodes de classe permettant de rapidement créer des matrices identité, de rotation, de translation ou encore de mise à l'échelle. L'opération de multiplication entre cette matrice et un vecteur 4D est fournie. L'opération de multiplication entre la matrice et un vecteur 3D est aussi fournie. Cette dernière opération fait un aller-retour dans l'espace des coordonnées homogènes pour appliquer la transformation décrite par la matrice au vecteur 3D. Une spécialisation de cette classe de matrice est proposée dans le fichier « Math/Matrix4x4.h », il s'agit de Math::Matrix4x4f.

Classe Quaternion<Type>. Il s'agit d'une classe de gestion de quaternion, paramétrée par le type de scalaire et fournie dans le fichier « Math/Quaternion.h ». Ce quaternion peut être initialisé à partir de l'axe de rotation et de l'angle (en radians), redéfini les opérateurs classiques et dispose d'une méthode *rotate* permettant de calculer la rotation d'un vecteur 3D.

Classe Sampler. Il s'agit d'une classe permettant d'échantillonner de manière aléatoire et suivant une loi uniforme différentes formes géométriques telles qu'un cube, une boule, un disque etc… Cette classe est fournie dans le fichier « Math/Sampler.h ».

Constantes. Un certain nombre de constantes utiles (telles que PI par exemple), sont définies dans le fichier « Math/Constant.h ».