Introduction à Irrlicht
première partie

CPE — ETI5–IMI

6 novembre 2015

1 Introduction

Dans ce TP, nous allons utiliser le moteur d'affichage Irrlicht pour créer une scène de plus en plus complexe, en ajoutant des fonctionnalités petit à petit.

Il s'agit d'un TP de type « découverte ». Il n'y a rien à rendre même si des questions sont posées (elles sont là pour vous guider).

Si vous êtes coincés dans une question qui vous intéresse peu, vous pouvez passer à la suivante en utilisant la solution fournie à chaque étape.

Faites attention cependant de ne pas télécharger toutes les solutions d'un coup, chacune contient des données de modèle et de texture qui risquent de vite saturer votre quota.

2 Ouvrons la fenêtre

Notre première scène sera juste une fenêtre vide.

On doit inclure ce fichier à chaque fois qu'on veut utiliser le moteur Irrlicht :

```
1 #include <irrlicht.h>
```

Ces quelques déclarations facilitent l'utilisation des fonctions et symbôles des différents namespaces :

```
1 using namespace irr;
2 namespace iv = irr::video;
3 namespace is = irr::scene;
```

Puis la structure suivante (création d'un device, puis boucle infinie du dessin de toute la scène encadré par un beginScene/endScene) sera toujours la même :

```
1  int main()
2  {
3    // Création de la fenêtre et du système de rendu.
4    IrrlichtDevice *device = createDevice(iv::EDT_OPENGL);
5
6    iv::IVideoDriver  *driver = device->getVideoDriver();
7    is::ISceneManager *smgr = device->getSceneManager();
8
9    while(device->run())
10   {
11     driver->beginScene();
12
13     // Dessin de la scène :
14     smgr->drawAll();
15     //
16     driver->endScene();
17   }
18   device->drop();
19 }
```

Cela affiche une fenêtre noire, rien de bien passionnant.

En changeant la ligne beginScene par :

```
1 driver->beginScene(true, true, iv::SColor(100,150,200,255));
```

on a une couleur de fond différente.

Le code correpondant à ceci est dans `etape1.tar.gz`

3 Gestion du clavier

Avant d'aller plus loin, nous allons ajouter un gestionnaire d'évenements permettant de gérer le clavier (et la souris plus tard).

Pour cela, il faut créer une structure dérivée de `irr::IEventReceiver` dans laquelle on surchargera la fonction `bool OnEvent(const SEvent &event);`.

Cela peut ressembler au code suivant :

```
1  struct MyEventReceiver : IEventReceiver
2  {
3    bool OnEvent(const SEvent &event)
4    {
5      // Si l'événement est de type clavier (KEY_INPUT)
6      // et du genre pressage de touche
7      // et que la touche est escape, on sort du programme
8      if (event.EventType == EET_KEY_INPUT_EVENT &&
9          event.KeyInput.PressedDown &&
10         event.KeyInput.Key == KEY_ESCAPE)
11       exit(0);
12
13     return false;
14   }
15 };
```

Vous pouvez évidemment ajouter du code pour gérer d'autres touches, paramétrer cette structure avec ce que vous voulez. Ce n'est pas très intéressant avec une fenêtre vide, mais cela va évoluer.

Pour que ce code soit utilisé, il faut passer une variable de ce type comme dernier paramètre à la fonction `createDevice()`. If faut donc donner tous les paramètres de cette fonction, de la façon suivante :

```
1  // Le gestionnaire d'événements
2   MyEventReceiver receiver;
3   // Création de la fenêtre et du système de rendu.
4   IrrlichtDevice *device = createDevice(iv::EDT_OPENGL,
5                                         ic::dimension2d<u32>(640, 480),
6                                         16, false, false, false, &receiver);
```

Cela suffit pour permettre de terminer notre programme en appuyant sur la touche *escape*. Vous pouvez trouver la liste des touches disponibles dans la documentation de Irrlicht, ainsi qu'une description des paramètres de `createDevice()`.

Le code correpondant à ceci est dans `etape2.tar.gz`

4 Ajout d'un personnage

Charger un personnage (ou un autre type de maillage) est grandement facilité par l'utilisation d'un moteur. Pour charger notre personnage, dont vous avez les données depuis le TP sur les ombres, il suffit d'ajouter la ligne suivante :

```
1  is::IAnimatedMesh *mesh = smgr->getMesh("data/tris.md2");
```

Irrlicht sait charger beaucoup de formats de fichier de maillage (.3ds, .b3d, .x, .md2, .md3, .obj et bien d'autres). Voir la documentation de la fonction `getMesh()`.

Une fois le maillage chargé, on doit l'ajouter à la scène de la manière suivante :

```
1   is::IAnimatedMeshSceneNode *node = smgr->addAnimatedMeshSceneNode(mesh);
```

La fonction `addAnimatedMeshSceneNode()` renvoie un nœud qui nous permettra de manipuler le maillage pour changer son aspect, le déplacer, etc.

Il ne nous reste plus qu'à définir une caméra pour que notre personnage soit visible (et animé !) avec la ligne suivante :

```
1  smgr->addCameraSceneNode(nullptr, ic::vector3df(0, 30, -40), ic::vector3df(0, 5, 0));
```

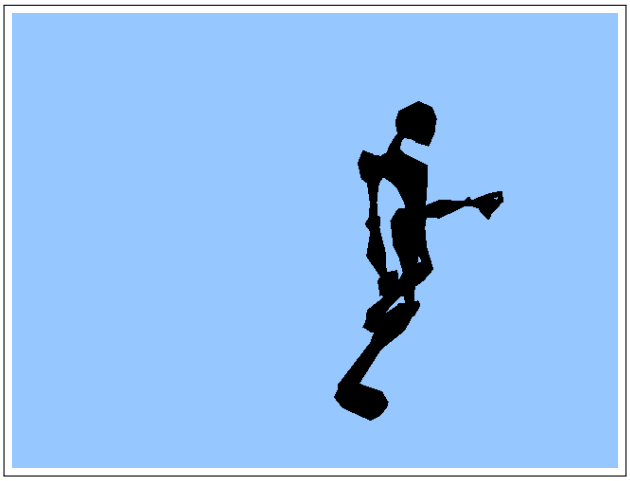

FIGURE 1 – Notre personnage, non texturé

Par défaut, les maillages sont éclairés, mais apparaissent noir, car leurs réglages de lumière diffuse, ambiante ou spéculaire sont nuls. De la même manière, l'animation par défaut consiste à jouer toutes les animations les unes à la suite des autres. Cela donne une scène semblable à la figure 1.

Les trois lignes suivantes remédient à cela et ajoutent une texture :

```
1 node->setMaterialFlag(irr::video::EMF_LIGHTING, false);
2 node->setMD2Animation(irr::scene::EMAT_STAND);
3 node->setMaterialTexture(0, driver->getTexture("data/blue_texture.pcx"));
```

Cela permet d'avoir une scène ressemblant à la figure 2.

FIGURE 2 – Notre personnage, texturé

Le code correpondant à ceci est dans `etape3.tar.gz`

Question 1 : *Parcourez la documentation de la classe* `IAnimatedMeshSceneNode` *et de ses classes parentes pour changer l'animation ou voir ce qu'il possible de modifier pour un tel nœud.*

5 Déplacement d'un personnage

En particulier, on peut trouver les fonctions `getPosition()`, `setPosition()`, `getRotation()` et `setRotation()`, qui permettent de récupérer et de modifier la position et l'orientation d'un nœud.

Question 2 : *Modifiez la classe `MyEventReceiver` afin de faire bouger notre personnage en utilisation ces fonctions dans la fonction `OnEvent`.*

Il est possible de demander au moteur de déplacer automatiquement des nœuds d'une scène. Par exemple, le code suivant permet de bouger le nœud `perso_cours` en ligne droite entre deux points :

```
1 is::ISceneNodeAnimator *anim = smgr->createFlyStraightAnimator(ic::vector3df(-100,0,60),
2                                                                ic::vector3df(100,0,60), 3500, true);
3 perso_cours->addAnimator(anim);
```

Question 3 : *Créez un deuxième personnage, et animez-le en ligne droite avec la fonction précédente. Ajustez la caméra au besoin. Consultez la documentation de `createFlyStraightAnimator` afin d'en comprendre les paramètres, puis celles de `createFlyCircleAnimator` et de `createFollowSplineAnimator` afin de faire bouger des personnages comme vous le souhaitez.*

FIGURE 3 – Notre personnage, accompagné d'un marcheur en ligne droite et d'un sauteur en rond

Le code correpondant à ceci est dans `etape4.tar.gz`

6 Chargement d'un décor complexe

Nous avons plusieurs personnages mais ils évoluent tous dans le vide pour l'instant.

L'un des rôles d'un moteur de jeu (ou même d'un moteur graphique) est de permettre de gérer facilement des éléments graphiques complexes. On a déjà vu qu'il était assez simple de manipuler des personnages animés. Manipuler un décor complet n'est pas vraiment plus complexe.

L'ajout des quelques lignes suivantes permet de charger un décor complet, et de demander au moteur de l'afficher autour de nos personnages, même si les interactions physiques (collisions) entre les différentes entités ne sont pas gérées (cela viendra dans le prochain TP) :

```
1 // Ajout de l'archive qui contient entre autres un niveau complet
2  device->getFileSystem()->addFileArchive("data/map-20kdm2.pk3");
3   // On charge un bsp (un niveau) en particulier :
4  is::IAnimatedMesh *mesh = smgr->getMesh("20kdm2.bsp");
5  is::ISceneNode *node;
6  node = smgr->addOctreeSceneNode(mesh->getMesh(0), nullptr, -1, 1024);
7  // Translation pour que nos personnages soient dans le décor
8  node->setPosition(core::vector3df(-1300,-104,-1249));
```

On commence par ajouter toute une archive à ce qui est disponible en lecture pour le moteur. Un fichier `.pk3` est en fait un `.zip` qui contient des textures, des scripts, des niveaux, etc. Après l'appel à la fonction `addFileArchive()`, les fichiers contenus dans cette archive seront vus comme s'ils étaient directement dans le répertoire courant.

Il suffit ensuite de charger un niveau (bsp) qui est vu comme un ensemble de maillages dont le premier (souvent le seul) est celui qui nous intéresse. À partir de ce maillage, on crée un nœud qui n'est pas tellement différent des nœuds que l'on a créé pour les personnages si ce n'est qu'il utilise un octree pour accélérer l'affichage. Le fait de déplacer le nœud résultant n'est vraiment là que pour regrouper les éléments de notre scène.

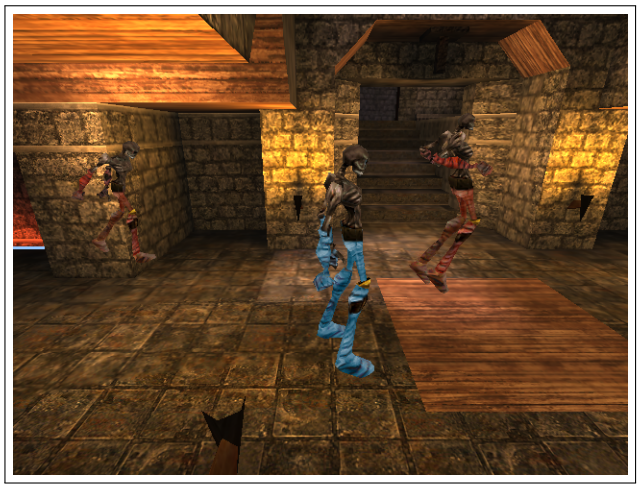

FIGURE 4 – Nos personnages dans un décor

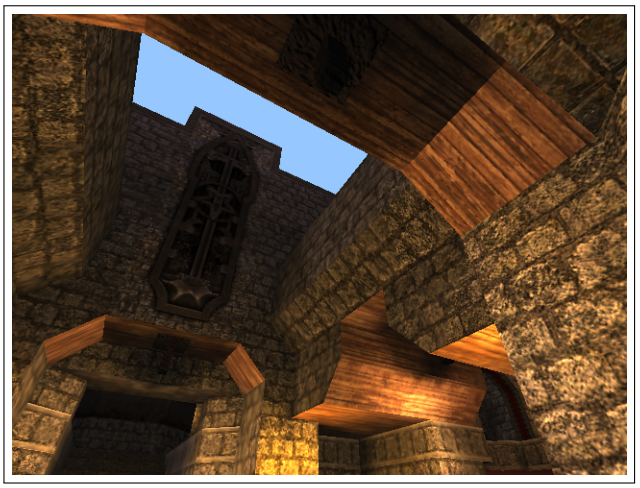

FIGURE 5 – Un autre point de vue du décor

Question 4 : *Implémentez cela et déplacez la caméra pour visiter ce décor*

Il est possible d'utiliser deux types de caméras « intelligentes » avec Irrlicht. La première mime un déplacement que l'on retrouve dans les jeux types *FPS*. Pour l'utiliser, on remplacera l'appel à `addCameraSceneNode()` par :

```
1 smgr->addCameraSceneNodeFPS();
```

Cette caméra permet de se déplacer dans la scène avec la souris et les touches flèchées du clavierr (attention si vous utilisez déjà ces touches pour autre chose !)

Une autre caméra prédéfinie est celle utilisée par les modelleurs et que l'on peut diriger à la souris. On peut la créer de la manière suivante :

```
1 smgr->addCameraSceneNodeMaya();
```

Une caméra est un nœud de la scène. On peut donc modifier sa position, sa rotation, ou plus simplement changer l'endroit qu'elle « regarde » avec la fonction `setTarget()`.

Question 5 : *Expérimentez avec les différentes caméras disponibles. Essayez de créer une caméra qui suivrait les déplacements d'un personnage.*

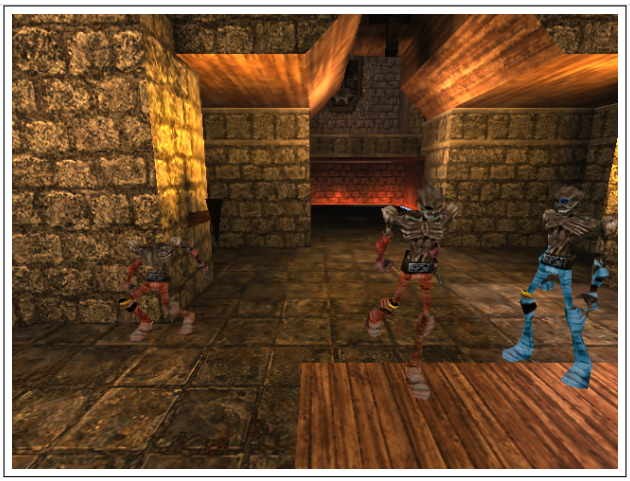

FIGURE 6 – Une vue des personnages après déplacement de la caméra

Le code correpondant à ceci est dans `etape5.tar.gz`