

Irrlicht

Étude d'un moteur de jeu

David Odin

CPE Lyon

2015

- 1 MOTEURS 3D ET MOTEURS DE JEU
- 2 LE CHOIX DE IRRLICHT
- 3 UTILISATION DE IRRLICHT
- 4 ANNEXES

- **Définition** : ensemble de bibliothèques permettant de développer rapidement des afficheurs 3D ou des jeux

- **Définition** : ensemble de bibliothèques permettant de développer rapidement des afficheurs 3D ou des jeux
- Fonctions de haut niveaux manipulant des objets graphiques (!= primitives)

- **Définition** : ensemble de bibliothèques permettant de développer rapidement des afficheurs 3D ou des jeux
- Fonctions de haut niveaux manipulant des objets graphiques (!= primitives)
- Fonctions permettant de simplifier plein de trucs : math, portabilité, gestion de scène, optimisations, etc.

DIFFÉRENTES BIBLIOTHÈQUES

Moteur de jeu = ensemble de bibliothèques.

On trouve souvent :

- Mémoire, logs, portabilité

DIFFÉRENTES BIBLIOTHÈQUES

Moteur de jeu = ensemble de bibliothèques.

On trouve souvent :

- Mémoire, logs, portabilité
- Math

DIFFÉRENTES BIBLIOTHÈQUES

Moteur de jeu = ensemble de bibliothèques.

On trouve souvent :

- Mémoire, logs, portabilité
- Math
- Input

DIFFÉRENTES BIBLIOTHÈQUES

Moteur de jeu = ensemble de bibliothèques.

On trouve souvent :

- Mémoire, logs, portabilité
- Math
- Input
- Graphique

DIFFÉRENTES BIBLIOTHÈQUES

Moteur de jeu = ensemble de bibliothèques.

On trouve souvent :

- Mémoire, logs, portabilité
- Math
- Input
- Graphique
- Importation de données : images, modèles, décor, etc.

DIFFÉRENTES BIBLIOTHÈQUES

Moteur de jeu = ensemble de bibliothèques.

On trouve souvent :

- Mémoire, logs, portabilité
- Math
- Input
- Graphique
- Importation de données : images, modèles, décor, etc.
- GUI

DIFFÉRENTES BIBLIOTHÈQUES

Moteur de jeu = ensemble de bibliothèques.

On trouve souvent :

- Mémoire, logs, portabilité
- Math
- Input
- Graphique
- Importation de données : images, modèles, décor, etc.
- GUI
- Physique

Moteur de jeu = ensemble de bibliothèques.

On trouve souvent :

- Mémoire, logs, portabilité
- Math
- Input
- Graphique
- Importation de données : images, modèles, décor, etc.
- GUI
- Physique
- Musique / effets sonores

- Logger,

- Logger,
- Gestion mémoire,

- Logger,
- Gestion mémoire,
- Design patterns

- vec2 , vec3 , vec4

- `vec2`, `vec3`, `vec4`
- matrices (perspective, rotation, translation)

- `vec2`, `vec3`, `vec4`
- matrices (perspective, rotation, translation)
- quaternion

- Clavier

- Clavier
- Souris

- Clavier
- Souris
- Joystick

- Clavier
- Souris
- Joystick
- Temps

- Image / texture, beaucoup de formats

- Image / texture, beaucoup de formats
- Modèles 3D

- Image / texture, beaucoup de formats
- Modèles 3D
- Décor / environnement

- Image / texture, beaucoup de formats
- Modèles 3D
- Décor / environnement
- Sons

Manipulation d'un graphe où chaque noeud peut-être :

- Un personnage,

Manipulation d'un graphe où chaque noeud peut-être :

- Un personnage,
- Un objet décoratif,

Manipulation d'un graphe où chaque noeud peut-être :

- Un personnage,
- Un objet décoratif,
- Un objet scénarique,

Manipulation d'un graphe où chaque noeud peut-être :

- Un personnage,
- Un objet décoratif,
- Un objet scénarique,
- Un élément de décor (une pièce),

Manipulation d'un graphe où chaque noeud peut-être :

- Un personnage,
- Un objet décoratif,
- Un objet scénarique,
- Un élément de décor (une pièce),
- Un décor entier,

Manipulation d'un graphe où chaque noeud peut-être :

- Un personnage,
- Un objet décoratif,
- Un objet scénarique,
- Un élément de décor (une pièce),
- Un décor entier,
- Une caméra,

Manipulation d'un graphe où chaque noeud peut-être :

- Un personnage,
- Un objet décoratif,
- Un objet scénarique,
- Un élément de décor (une pièce),
- Un décor entier,
- Une caméra,
- Déplacé,

Manipulation d'un graphe où chaque noeud peut-être :

- Un personnage,
- Un objet décoratif,
- Un objet scénarique,
- Un élément de décor (une pièce),
- Un décor entier,
- Une caméra,
- Déplacé,
- Testé,

Manipulation d'un graphe où chaque noeud peut-être :

- Un personnage,
- Un objet décoratif,
- Un objet scénarique,
- Un élément de décor (une pièce),
- Un décor entier,
- Une caméra,
- Déplacé,
- Testé,
- etc.

Certains moteurs proposent des outils de productions de données liées :

- Un modelleur plus ou moins complet

Certains moteurs proposent des outils de productions de données liées :

- Un modelleur plus ou moins complet
- Un éditeur de scénario / d'éléments scénariques

- 1 MOTEURS 3D ET MOTEURS DE JEU
- 2 LE CHOIX DE IRRLICHT**
- 3 UTILISATION DE IRRLICHT
- 4 ANNEXES

- Unity

UN CHOIX TRÈS VASTE

- Unity
- Ogre3D

UN CHOIX TRÈS VASTE

- Unity
- Ogre3D
- Id2, Id3, Id4

UN CHOIX TRÈS VASTE

- Unity
- Ogre3D
- Id2, Id3, Id4
- Crytek

UN CHOIX TRÈS VASTE

- Unity
- Ogre3D
- Id2, Id3, Id4
- Crytek
- Unigine

- Unity
- Ogre3D
- Id2, Id3, Id4
- Crytek
- Unigine
- Des dizaines d'autres sur https://fr.wikipedia.org/wiki/Liste_de_moteurs_de_jeu

- C++ (connu des étudiants)

LES AVANTAGES DE IRRLICHT

- C++ (connu des étudiants)
- Facile à installer

LES AVANTAGES DE IRRLICHT

- C++ (connu des étudiants)
- Facile à installer
- Open Source

LES AVANTAGES DE IRRLICHT

- C++ (connu des étudiants)
- Facile à installer
- Open Source
- Assez représentatif

LES AVANTAGES DE IRRLICHT

- C++ (connu des étudiants)
- Facile à installer
- Open Source
- Assez représentatif
- Bien documenté

- C++ (connu des étudiants)
- Facile à installer
- Open Source
- Assez représentatif
- Bien documenté
- Didacticiels

- Plus vraiment développé (peu de développeurs actifs)

LES INCONVÉNIENTS DE IRRLICHT

- Plus vraiment développé (peu de développeurs actifs)
- Code un peu veillot

LES INCONVÉNIENTS DE IRRLICHT

- Plus vraiment développé (peu de développeurs actifs)
- Code un peu vieillot
- Limité à OpenGL 3.0 / GLSL 1.3

- 1 MOTEURS 3D ET MOTEURS DE JEU
- 2 LE CHOIX DE IRRLICHT
- 3 UTILISATION DE IRRLICHT**
- 4 ANNEXES

```
CPPFLAGS = -I/usr/include/irrlicht  
CXXFLAGS = -Wall -Wextra -O2 -g
```



```
CPPFLAGS = -I/usr/include/irrlicht
```

```
CXXFLAGS = -Wall -Wextra -O2 -g
```

```
irrlicht-exemple1: main.o
```

```
    g++ $^ -o $@ -lIrrlicht
```

- `irr::` : Tout est dans cet espace de noms

LES ESPACES DE NOMS

- `irr::` : Tout est dans cet espace de noms
- `irr::core` Structures de maths, containers, etc.

LES ESPACES DE NOMS

- `irr::` : Tout est dans cet espace de noms
- `irr::core` Structures de maths, containers, etc.
- `irr::video` Les classes liées au driver video, qui réalisent l'affichage 2D et 3D.

- `irr::` : Tout est dans cet espace de noms
- `irr::core` Structures de maths, containers, etc.
- `irr::video` Les classes liées au driver video, qui réalisent l'affichage 2D et 3D.
- `irr::scene` manipulation de maillages (chargement, transformations, animation), de caméra, regroupement en un graphe, etc.

- `irr::` : Tout est dans cet espace de noms
- `irr::core` Structures de maths, containers, etc.
- `irr::video` Les classes liées au driver video, qui réalisent l'affichage 2D et 3D.
- `irr::scene` manipulation de maillages (chargement, transformations, animation), de caméra, regroupement en un graphe, etc.
- `irr::io` Chargement de fichiers, utilisation transparente de fichiers compressés, etc.

LES ESPACES DE NOMS

- `irr::` Tout est dans cet espace de noms
- `irr::core` Structures de maths, containers, etc.
- `irr::video` Les classes liées au driver video, qui réalisent l'affichage 2D et 3D.
- `irr::scene` manipulation de maillages (chargement, transformations, animation), de caméra, regroupement en un graphe, etc.
- `irr::io` Chargement de fichiers, utilisation transparente de fichiers compressés, etc.
- `irr::gui` Les classes permettant de créer une interface utilisateur (boutons, textes, images 2D, menus, etc.)

REDÉFINITION DES TYPES DE BASES

Pour la portabilité, les types classiques sont redéfinis :

- `s8`, `u8`, `c8`

REDÉFINITION DES TYPES DE BASES

Pour la portabilité, les types classiques sont redéfinis :

- `s8`, `u8`, `c8`
- `s16`, `u16`

REDÉFINITION DES TYPES DE BASES

Pour la portabilité, les types classiques sont redéfinis :

- **s8, u8, c8**
- **s16, u16**
- **s32, u32**

Pour la portabilité, les types classiques sont redéfinis :

- **s8, u8, c8**
- **s16, u16**
- **s32, u32**
- **s64, u64**

Pour la portabilité, les types classiques sont redéfinis :

- **s8, u8, c8**
- **s16, u16**
- **s32, u32**
- **s64, u64**
- **f32, f64**

Pour la portabilité, les types classiques sont redéfinis :

- **s8, u8, c8**
- **s16, u16**
- **s32, u32**
- **s64, u64**
- **f32, f64**

REDÉFINITION DES TYPES DE BASES

Pour la portabilité, les types classiques sont redéfinis :

- **s8**, **u8**, **c8**
- **s16**, **u16**
- **s32**, **u32**
- **s64**, **u64**
- **f32**, **f64**

Exemples :

```
s32 a; // équivalent à int a en général  
f64 b; // équivalent à double
```

Pour les structures simples (vecteurs, dimensions, rectangles, etc.) Irrlicht a fait le choix de code généré.

- `irr::core::vector3d<float>` ou `irr::core::vector3df` pour un `vec3`

Pour les structures simples (vecteurs, dimensions, rectangles, etc.) Irrlicht a fait le choix de code généré.

- `irr::core::vector3d<float>` ou `irr::core::vector3df` pour un `vec3`
- `irr::core::dimension2d<u32>` pour quelque chose ayant une largeur et une hauteur

Pour les structures simples (vecteurs, dimensions, rectangles, etc.) Irrlicht a fait le choix de code généré.

- `irr::core::vector3d<float>` ou `irr::core::vector3df` pour un `vec3`
- `irr::core::dimension2d<u32>` pour quelque chose ayant une largeur et une hauteur
- `irr::core::rect<T>`, `rectf`, `recti`, pour un rectangle

Pour les structures simples (vecteurs, dimensions, rectangles, etc.) Irrlicht a fait le choix de code généré.

- `irr::core::vector3d<float>` ou `irr::core::vector3df` pour un `vec3`
- `irr::core::dimension2d<u32>` pour quelque chose ayant une largeur et une hauteur
- `irr::core::rect<T>`, `rectf`, `recti`, pour un rectangle
- etc.

L'initialisation du moteur Irrlicht peut-être aussi simple que :

```
irr::IrrlichtDevice *device = irr::createDevice ();
```

INITIALISATION DU MOTEUR DE RENDU

L'initialisation du moteur Irrlicht peut-être aussi simple que :

```
irr::IrrlichtDevice *device = irr::createDevice();
```

Équivalent à :

```
irr::IrrlichtDevice *device =  
    irr::createDevice(irr::video::EDT_SOFTWARE,  
                      irr::core::dimension2d<u32>(640, 480)),  
                      16, false, false, false, nullptr);
```

INITIALISATION DU MOTEUR DE RENDU

L'initialisation du moteur Irrlicht peut-être aussi simple que :

```
irr::IrrlichtDevice *device = irr::createDevice();
```

Équivalent à :

```
irr::IrrlichtDevice *device =  
    irr::createDevice(irr::video::EDT_SOFTWARE,  
                      irr::core::dimension2d<u32>(640, 480)),  
                      16, false, false, false, nullptr);
```

Comme premier paramètre, on préférera `irr::video::EDT_OPENGL` pour les TP.

Le *device* crée trois entités importantes :

- Le *driver* qui gère l'affichage :

```
irr::video::IVideoDriver *driver =  
device->getVideoDriver();
```

Le *device* crée trois entités importantes :

- Le *driver* qui gère l'affichage :

```
irr::video::IVideoDriver *driver =  
    device->getVideoDriver();
```

- Le *scene manager* les objets de la scène et la caméra :

```
irr::scene::ISceneManager *s_m =  
    device->getSceneManager();
```

Le *device* crée trois entités importantes :

- Le *driver* qui gère l'affichage :

```
irr::video::IVideoDriver *driver =  
    device->getVideoDriver();
```

- Le *scene manager* les objets de la scène et la caméra :

```
irr::scene::ISceneManager *s_m =  
    device->getSceneManager();
```

- Le *GUI Environment* qui gère tous les éléments de l'interface utilisateur 2D :

```
irr::gui::IGUIEnvironment *gui =  
    device->getGUIEnvironment();
```


Le *scene manager* gère un graphe composé de nœuds qui peuvent être :

- Un maillage (animé ou non) :

```
mesh = smgr->getMesh("data/tris.md2");  
node = smgr->addAnimatedMeshSceneNode(mesh);
```

Chaque nœud peut ensuite être manipulé (voir documentation).

Le *scene manager* gère un graphe composé de nœuds qui peuvent être :

- Un maillage (animé ou non) :

```
mesh = smgr->getMesh("data/tris.md2");  
node = smgr->addAnimatedMeshSceneNode(mesh);
```

- Une camera :

```
smgr->addCameraSceneNode();
```

Chaque nœud peut ensuite être manipulé (voir documentation).

Le *scene manager* gère un graphe composé de nœuds qui peuvent être :

- Un maillage (animé ou non) :

```
mesh = smgr->getMesh("data/tris.md2");  
node = smgr->addAnimatedMeshSceneNode(mesh);
```

- Une camera :

```
smgr->addCameraSceneNode();
```

- Un décor

Chaque nœud peut ensuite être manipulé (voir documentation).

Le *scene manager* gère un graphe composé de nœuds qui peuvent être :

- Un maillage (animé ou non) :

```
mesh = smgr->getMesh("data/tris.md2");  
node = smgr->addAnimatedMeshSceneNode(mesh);
```

- Une camera :

```
smgr->addCameraSceneNode();
```

- Un décor
- Un autre élément graphique...

Chaque nœud peut ensuite être manipulé (voir documentation).

Le *GUI environment* permet l'affichage d'éléments graphiques 2D comme :

- Un label :

```
gui->addStaticText (L"Bonjour",  
                    rect<s32>(50, 50, 150, 100));
```

Le *GUI environment* permet l'affichage d'éléments graphiques 2D comme :

- Un label :

```
gui->addStaticText (L"Bonjour",  
                    rect<s32>(50, 50, 150, 100));
```

- Une image :

```
texture = driver->getTexture ("data/logo.tga");  
gui->addImage (texture, position2d<s32>(10,20));
```

Mais aussi des éléments interactifs comme :

- Un texte éditable :

```
gui->addEditBox(L"Texte_éritable",  
                rect<s32>(50, 50, 250, 100));
```

Mais aussi des éléments interactifs comme :

- Un texte éditable :

```
gui->addEditBox(L"Texte_éritable",  
               rect<s32>(50, 50, 250, 100));
```

- Un bouton clickable :

```
gui->addButton(rect<s32>(50, 50, 150, 100),  
              nullptr, id,  
              L"Coucou", L"Dit_coucou");
```


Mais aussi des éléments interactifs comme :

- Un texte éditable :

```
gui->addEditBox(L"Texte_éditable",  
               rect<s32>(50, 50, 250, 100));
```

- Un bouton clickable :

```
gui->addButton(rect<s32>(50, 50, 150, 100),  
              nullptr, id,  
              L"Coucou", L"Dit_coucou");
```

- Une barre de défilement :

```
gui->addScrollBar(true, rect<s32>(50, 50, 150, 100));
```

Une fois le tout mis en place, on passe la main au moteur avec la boucle d'affichage (qui peut être plus complète que cela) :

```
while (device->run())
{
    driver->beginScene();
    // Dessin de la scène :
    smgr->drawAll();
    // Dessin de l'interface utilisateur :
    gui->drawAll();
    //
    driver->endScene();
}
```

- 1 MOTEURS 3D ET MOTEURS DE JEU
- 2 LE CHOIX DE IRRLICHT
- 3 UTILISATION DE IRRLICHT
- 4 ANNEXES**

- La documentation en ligne de irrlicht (avec les didacticiels) :
`http://irrlicht.sourceforge.net/docu/index.html`

- La documentation en ligne de irrlicht (avec les didacticiels) :
`http://irrlicht.sourceforge.net/docu/index.html`
- Real-Time Rendering (ISBN-13: 978-1568814247) A K Peters/CRC Press