Informatique Graphique

Moteurs 3D temps réels

Présentation de GobLim

Guillaume Gilet
Guillaume.Gilet@unilim.fr

2012-2013

master ISICG limoges

Moteurs 3D, en général

Programme *type*

- Initialisation
- Une boucle principale
  - Affichage de la scène
  - Détection et traitement des interactions de l'utilisateur
  - Mécanismes propres au jeu (I.A., physique,règles...)
  - Mise a jour des positions/etats des *entités*

Moteurs 3D, en général

Affichage de la scène

- Calcul des informations globales à la scène (lumières, ombres...)
- Pour chague objet 3D :
  - Préparation du pipeline (matrices, communication CPU-GPU...)
  - Envoi de la géométrie dans le pipeline
  - Remise en état du pipeline
- Calcul *post-processing*

GobLim

GobLim

- Moteur 3D développé au sein de l'équipe SIR
- OpenGL 4.+ et OpenGL 3.2
- Windows, Linux et (avec difficulté encore) MacOSX
- 2010 - Encore en développement

Principe

- Proposer une surcouche à la technique
- Automatiser chargements, managements des objets OpenGL
- Optimisation de mémoire (CPU et GPU)
- Permettre de réaliser facilement une idée/test/shader
- But : Développer une collection de méthodes de rendu indépendants

GobLim

Dépendances

GLFW (optionnel : interface)

- Interface et gestion des fenêtres
- Gestion des interactions

http ://www.glfw.org/

GLEW (optionnel : management OpenGL sous Windows)

- Initialisation des extensions/fonctions OpenGL

http ://glew.sourceforge.net/GLEW

GLM (Pour se simplifier les math)

- Utiliser les types et fcontions GLSL en C/C++

http ://glm.g-truc.net/

Interface (GLFW)

Principe

- Initialisation du programme (extensions openGL, création de fenêtre, contexte)
- Boucle principale
  - Gère l'affichage
  - Gère les interactions utilisateurs
  - Tant que la fenêtre est ouverte

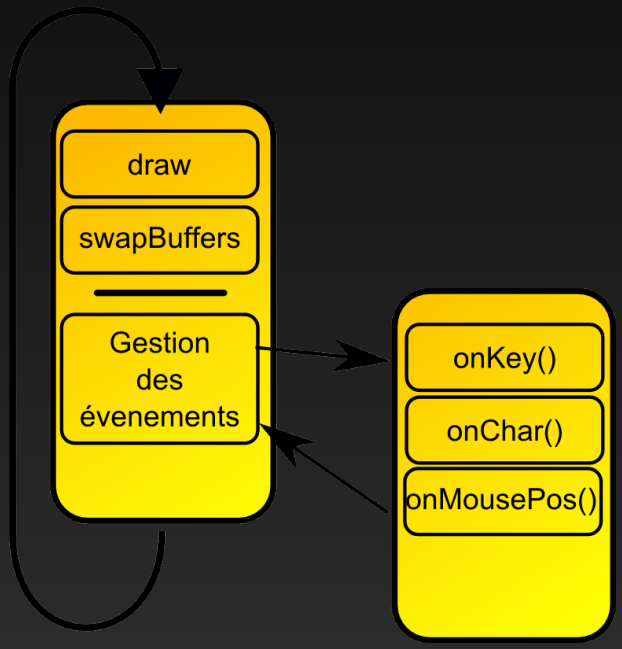
GLFW

- Initialisation de l'interface
  - glfwInit()
- Creation de la fenêtre
  - glfwOpenWindow()
  - Diverses options (titre, présence du curseur, clavier, buffers...)
- Définition des fonctions *callbacks*
  - Fonctions appelées par GLFW
  - En fonction des évènements (utilisateurs ou non)
  - glfwSetCharCallback() dans le cas de caractères
  - glfwSetKeyCallback() pour les touches spéciales

http ://www.glfw.org/

Interface

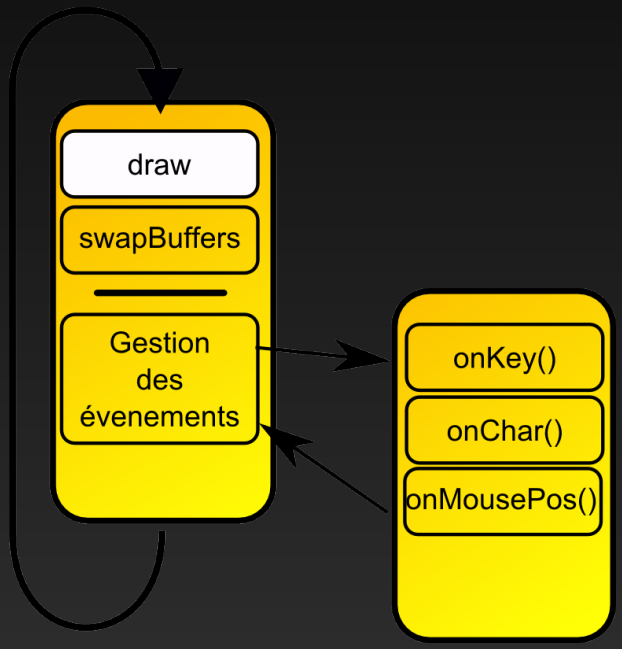
Boucle principale



Interface

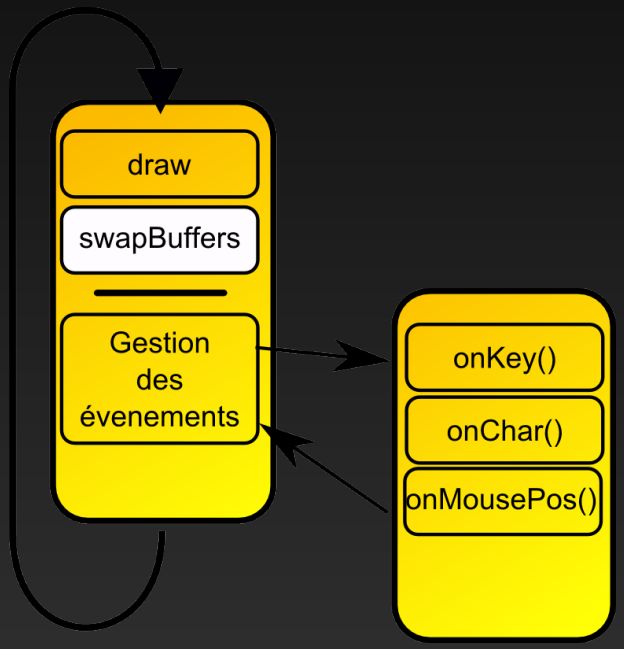
Boucle principale

- **Draw()** : Affichage de l'écran
  - Appel à render() de la classe EngineGL



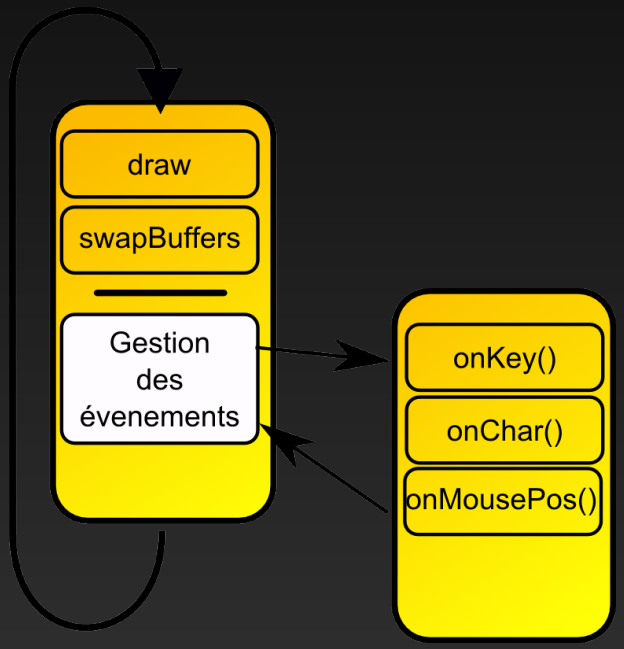
Boucle principale

- **Draw()** : Affichage de l'écran
  - Appel à render() de la classe EngineGL
- **glfwSwapBuffers()** : Echange les buffers d'affichage



Interface

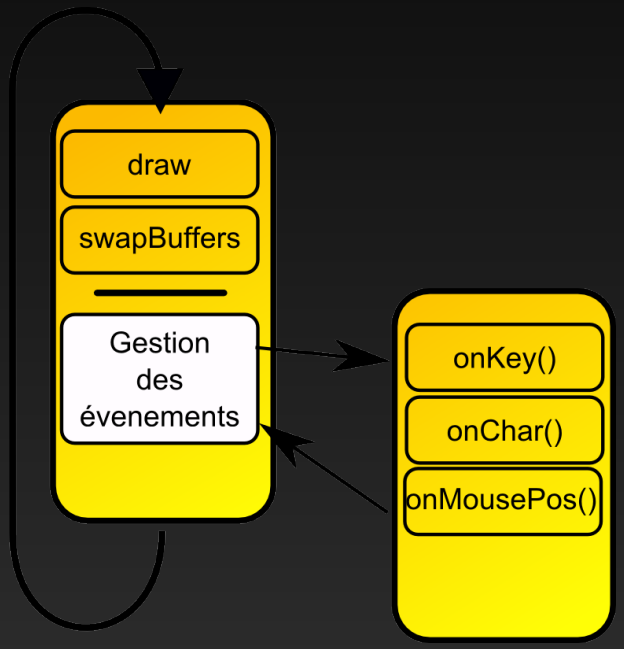
Boucle principale



- **Draw()** : Affichage de l'écran
  - Appel à render() de la classe EngineGL
- **glfwSwapBuffers()** : Echange les buffers d'affichage
- Evènement : Action utilisateur
  - Fonctions *callback*
  - void *GLFWCALL* **OnChar**(int c,int action)
  - avec **c** le caractère
  - **action** $\in$ [GLFW_PRESS,GLFW_RELEASE]
  - Mécanisme identique pour la souris, la fenêtre...

Interface

Boucle principale



- **Draw()** : Affichage de l'écran
  - Appel à render() de la classe EngineGL
- **glfwSwapBuffers()** : Echange les buffers d'affichage
- Evènement : Action utilisateur
  - Fonctions *callback*
  - void *GLFWCALL* **OnChar**(int c,int action)
  - avec **c** le caractère
  - **action** $\in$ [**GLFW_PRESS**,**GLFW_RELEASE**]
  - Mécanisme identique pour la souris, la fenêtre...

Pas de commande OpenGL dans l'interface

Classe **Engine**

- Le **coeur** du moteur 3D
- Gère l'affichage, la création de la scène, l'interaction avec le GPU...
- Une fonction initScene()
  - **Création** de la scène (Modèles 3D (**ModelGL**), entités (**Node**), méthodes de rendu...)
  - Paramètres OpenGL (couleur de fond, tests de profondeur...)
- Une fonction render()
  - Nettoyage des *buffers*
  - Collecte des **Node** du graphe
  - Appel de la fonction render() des **Node**
- C'est ici qu'on va travailler

Les scène 3D

classe **Scene**

- Gère le graphe de scène
- Graphe de **Node**
- Une racine (*root*)
- Un ensemble de manager de ressources
- Gère liste des modèles 3D, Nodes...
- Contient une **Camera**

Utilisation de **Scene**

- C'est un singleton
- Des constructeurs/destructeurs privés
- Utiliser Scene : :getInstance()

Gestion des Node

- Chaque **Node** identifié par un nom
- Creation des **Node** au vol
- Dans **Scene** : Un manager de **Node**
- Dans **Scene** : Node* getNode(name)

Gestion des ModelGL

- Un manager de **ModelGL**
- Chaque **ModelGL** identifié par un nom
- Creation des **ModelGL** au vol
- Dans **Scene** : Un manager de **ModelGL**
- Dans **Scene** : ModelGL* getModel<ModelGL>(string name)

Les entités 3D

class Node

- Entité 3D : Les *objets* de la scène
- Regroupe une géometrie (**Model**), une méthode de rendu(**Material**) et un repère 3D(**Frame**)
- Des fonctions setModel(Model*) et setMaterial(Material*)

Classe **ModelGL** (hérite de **Model**)

- Gestion d'un modèle géométrique (stocké au format .obj)
- Création de *Vertex Buffer Object* (VBO) sur le GPU
- Une fonction drawGeometry()
  - Envoie la géométrie dans le pipeline graphique

Méthode de rendu

Un MaterialGL

- Correspond à une méthode de rendu
- Charge des **GLProgram** et un **GLPipelineProgram** (*m_ProgramPipeline*)
- Deux pointeurs *vp* et *fp* sur des **GLProgram**
- Une fonction render()
  - Envoyer les informations nécessaires aux **GLProgram**
  - Activer notre objet pipeline (bind() de **GLPipelineProgram**
  - Envoyer la géometrie (drawGeometry() de **Node**
  - Desactiver le pipeline (release() de **GLPipelineProgram**

Un *Pipeline Program*

Un **GLPipelineProgram**

- Les étapes programmables du pipeline graphique
- Pointe sur les **GLProgram** utilisés pour une exécution du pipeline
- Nécessite au moins un *Vertex program* et un *Fragment program*
- Changement dynamique des **GLProgram** utilisés à chaque étape (*vertex*,*fragment*...)
- *Lié* en mémoire graphique
- Une fonction d'activation (bind()) et de désactivation (release())

Les *programs (shaders)*

Plusieurs **GLProgram**

- Peut être de plusieurs type (*Vertex*, *Fragment*...)
- Un *shader* lu dans un fichier
- Compilé et lié (inscrit en mémoire graphique) séparement
- Possède un espace en mémoire graphique pour ses variables globales
- Contient un manager de variables
- Convention de nom : NomFichier-VS.glsl : Vertex Shader
- Convention de nom : NomFichier-FS.glsl : Fragment Shader

Communication avec le GPU

En OpenGL < 3.0

- Mode direct
- A chaque frame :
  - Activer le *program*
  - Envoyer les informations au GPU (glUniform...)
- Contraintes et envoi d'information

GobLim (OpenGL 4+)

- Accès direct à la mémoire graphique (DSA)
- Chargement arbitraire
- Pointeurs sur la mémoire graphique
- Pour chaque **GLProgram** : Un *manager* de variables GPU : GLUniformManager* uniforms()

Communication avec le GPU

Principe

- Demander au manager un pointeur sur la mémoire graphique
- Un type GPUVariable : GPUvec3,GPUmat4,GPUfloat...
- Des fonctions GPUvec3* getGPUvec3(name), GPUmat4* getGPUmat4(name)
- avec *name* le nom de la variable dans le shader correspondant
- Une fonction Set() pour charger en mémoire graphique

Exemple : Matrice de transformation/projection (MVP)

- Dans le **Material** *PremierTP* :
  - Une GPUVariable* : GPUmat4* ModelViewProj ;
- Lors de l'initialisation, récupération du pointeur mémoire graphique :
  - ModelViewProj = vp->uniforms()->get("MVP") ;
- Lors de l'affichage, envoi des informations :
  - ModelViewProj->Set(...) ;

La caméra

Classe Camera

- Le **point de vue** du rendu
- Permet d'effectuer la projection
- Contient une matrice de projection et un repère (Frame)
- Définition d'une projection
  - Orthographique : setOrthographicProjection()
  - Perspective : setPerspectiveProjection() et setFrustum()
- Une fonction de placement (lookAt())
- Des fonctions de déplacement spécifiques (translate(),rotate())
- Une caméra doit être "'liée"' è la scène (link())