Alexandre Blaquière
909 206 137

Infographie
IFT-3100

Document de design

Travail présenté à
Philippe Voyer

Département d'informatique
Université Laval
2 mars 2014

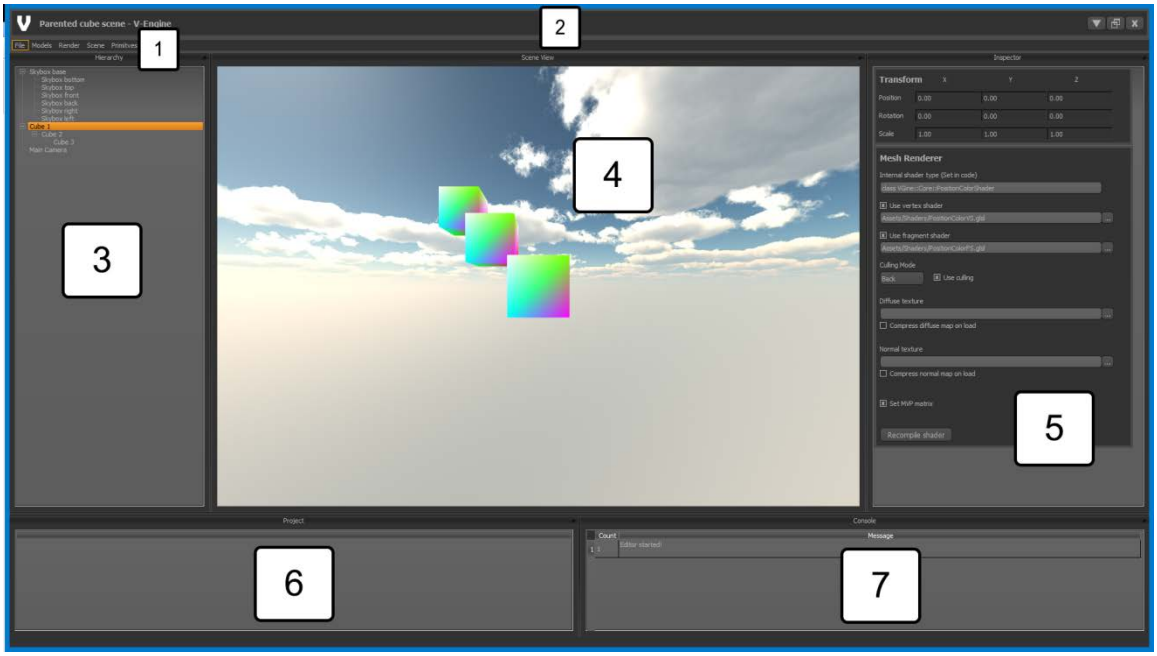
Table des matières

Sommaire

L'application jointe à ce document est un éditeur de scène 3D fait à l'aide de Qt. Cette application permettra à un usager d'importer des modèles 3D, de les positionner dans une scène qui se met à jour en temps réel pour finalement pouvoir en faire le rendu. Le design de l'application s'inspire fortement de Unity3D pour le fonctionnement au niveau du code et de Substance Designer pour l'apparence.

Liste de fonctionnalités

Avant tout, voici une image globale de l'interface et une explication rapide de chaque élément de l'interface.



1. L'éditeur offre une barre de menu permettant de faire certaines opérations telles que d'enregistrer une image de la scène ou charger un modèle 3D dans la scène.

2. La barre de titre contient à gauche le nom de la scène et le nom de l'éditeur. Elle sert aussi à déplacer l'éditeur dans l'écran en enfonçant et en tenant la souris enfoncée.

3. La hiérarchie d'objet permet de sélectionner un ou plusieurs objets dans la scène.

4 La vue de la scène en 3D. Il est possible d'interagir avec le monde en 3D en cliquant sur la fenêtre. Cette vue est en temps réel, et Qt essaie d'en faire le rendu à des intervalles de 18 millisecondes.

5. L'inspecteur d'objets. Dans cette région de l'éditeur, il est possible de modifier des paramètres appartenant aux objets de la scène tels que leur position ou bien leur paramètre de rendus.

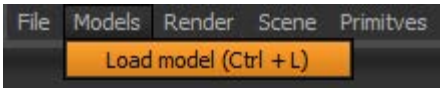
6. Cette zone de l'inspecteur n'est malheureusement pas fonctionnelle pour cette remise. L'intention originale était d'avoir un explorateur de ressources dans ce panneau.

7. La console. Cette zone de l'inspecteur permet de faire remonter à l'interface des messages générés par le code. Si tout se passe bien, seul le message «Editor started ! » devrait apparaitre.

Tous les panneaux de l'interface sont redimensionnables et au début de l'application, ceux-ci seront probablement trop petits. Il est aussi possible de sélectionner la barre de titre des panneaux pour en créer une fenêtre séparée. Cette fonctionnalité est particulièrement utile pour la vue de scène.

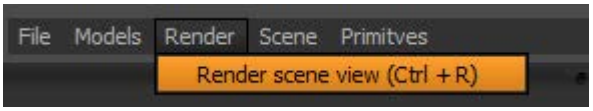
Chargement de modèles 3D

Il est possible de charger un modèle 3D en utilisant le menu « Models » de la barre de menu principal. Seuls des modèles de types .dae et .obj seront acceptés par le moteur.



Rendu de la vue de la scène

Il est possible de prendre une image de la vue actuellement fournie par la caméra principale. Il suffit d'utiliser la fonctionnalité « Render scene view » dans le menu « Render ». Après avoir choisi un nom de fichier, le moteur créera une image en format png de 1024*1024 pixels.



Changement de scène

Bien qu'il ne soit présentement pas possible de créer une scène vierge, il est possible de charger quatre scènes contenant déjà des objets. La première scène nommée Parented cube scene contient une skybox et trois cubes qui ont une relation enfant/parent. Il sera donc possible de modifier la transformation du cube1 et cela affectera la transformation du cube 2 et du cube3.

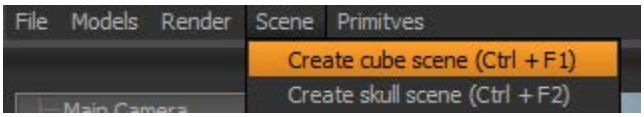
La deuxième scène nommée Skull Scene contient un modèle 3D représentant un crâne et une autre skybox différente de la première scène.

La troisième scène nommée lighting demo scene démontre les quatre types de lumières disponibles dans le moteur (Ambient, ponctuelle, directionnelle et projecteur). On peut aussi apercevoir sur le crâne un matériau utilisant du normal mapping.

La quatrième scène nommée displacement shader scene démontre trois objets ayant un shader de tesselation utilisant une height map pour déplacer les points des modèles ainsi qu'une normal map. On y retrouve aussi une seule lumière ponctuelle.

La cinquième scène nommée terrain scene démontre une autre utilisation du shader de tesselation de la scène précédente. Dans cette scène, un plan est fortement tesselé et la hauteur des sommets et modulé en fonction d'une texture de déplacement. Le plan est texturé avec trois textures différentes, soit du gazon, de la roche ou de la neige et c'est la hauteur des sommets qui décide de la texture qui sera appliqué.

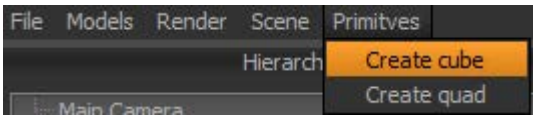
Il est possible de changer de scène en utilisant le menu nommé Scene et ensuite en sélectionnant la scène désirée.



Ajout de primitives 3D

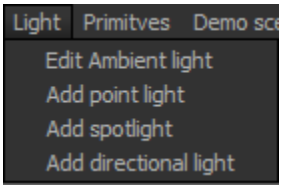
Il est possible d'ajouter 2 types de modèles 3D très simple à la scène, soit un cube et un quad. Les primitives auront automatiquement trois composantes d'attachés soit un « MeshRenderer », un « MeshFilter » et un « Transform ».

Faites attention, si vous ajoutez une nouvelle primitive, elle sera ajoutée en position (0,0,0) et elle risque d'être cachée par un autre objet! Il se peut aussi que la rotation de la primitive fasse en sorte qu'il ne soit pas visible à cause culling.



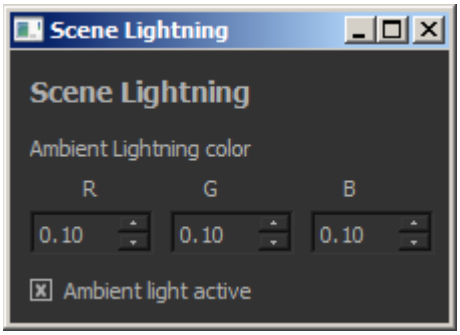
Ajout de lumière

Il est possible d'ajouter à une scène des lumières ou de modifier la lumière ambiante en utilisant le menu « Light ». Tout l'éclairage dans le moteur est implémenté avec les équations de Blinn-Phong. Voir le fichier Assets/Shaders/Lighting.glsl



Édition de la lumière ambiante

Il est possible de modifier les paramètres de la lumière ambiante en cliquant sur Edit Ambient light. Une nouvelle fenêtre apparaitra et il sera possible de changer la couleur de la lumière ou bien de changer la couleur de la lumière ambiante.



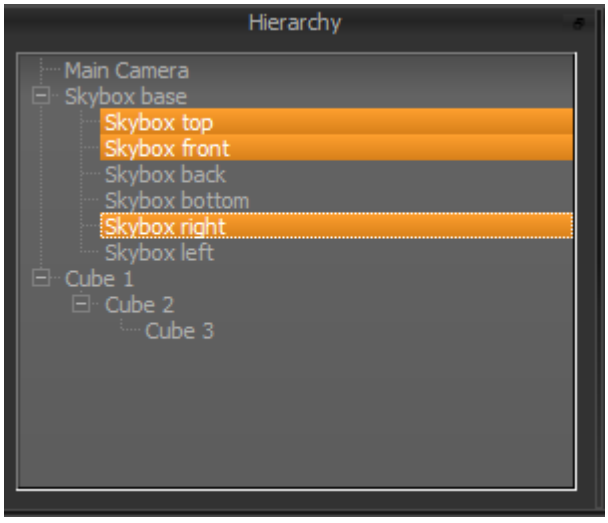
Éditions des autres types de lumières

Tous les autres types de lumières dans le moteur peuvent être édités. Si un objet dans la hiérarchie contient une composante de type lumière, il sera possible de changer les paramètres de cette lumière. L'exemple suivant montre l'édition d'une lumière de point.

Notez bien qu'il y a un nombre maximal de lumières dans la scène. En effet, il peut y avoir un maximum de 8 lumières du même type dans la même scène.

Sélection d'objets

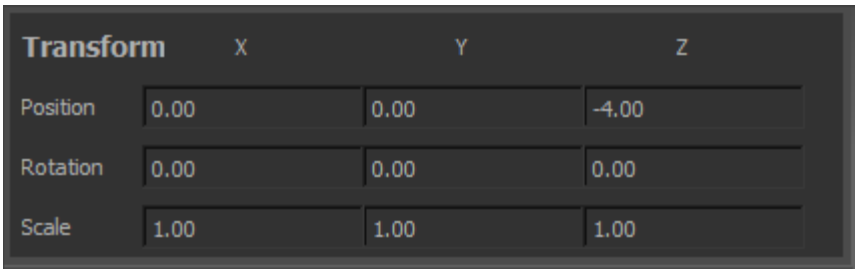
Il est possible de sélectionner un ou plusieurs objets dans la scène à l'aide du panneau de hiérarchie. Il suffit de cliquer sur un objet pour en sélectionner un seul ou cliquer et glisser pour en sélectionner plusieurs. Il est aussi possible d'ajouter des objets à la sélection en cliquant sur un autre objet en appuyant sur shift.



Modification des transformations d'un objet

Tout objet de scène à un transform qui est paramétrable. Lorsqu'un ou plusieurs objets sont sélectionnés, il est possible de modifier la position, la rotation et la taille d'un objet dans l'inspecteur. La rotation est exprimée en degré.

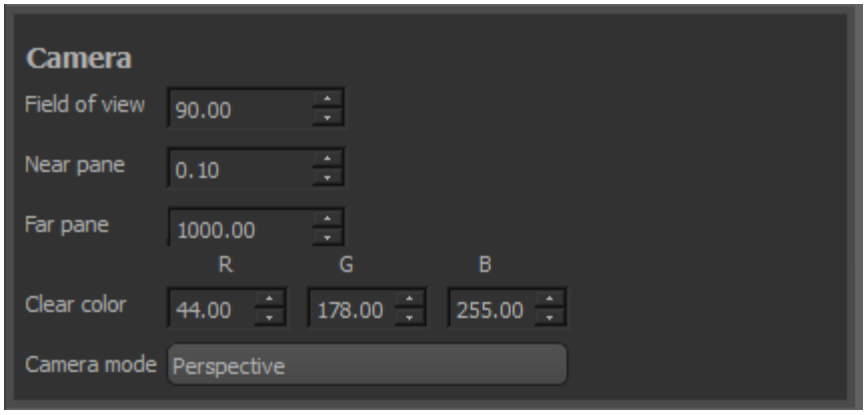
Faites attention, si vous sélectionnez plusieurs objets, tous les champs seront mis à 0 et ne seront pas mis à jour. Toutesfois, il est possible de modifier ces champs et la transformation de tous les objets sera modifiée.



Modification des paramètres d'une caméra

Il est possible de modifier les paramètres de tout objet de scène sur lequel est attachée une caméra.

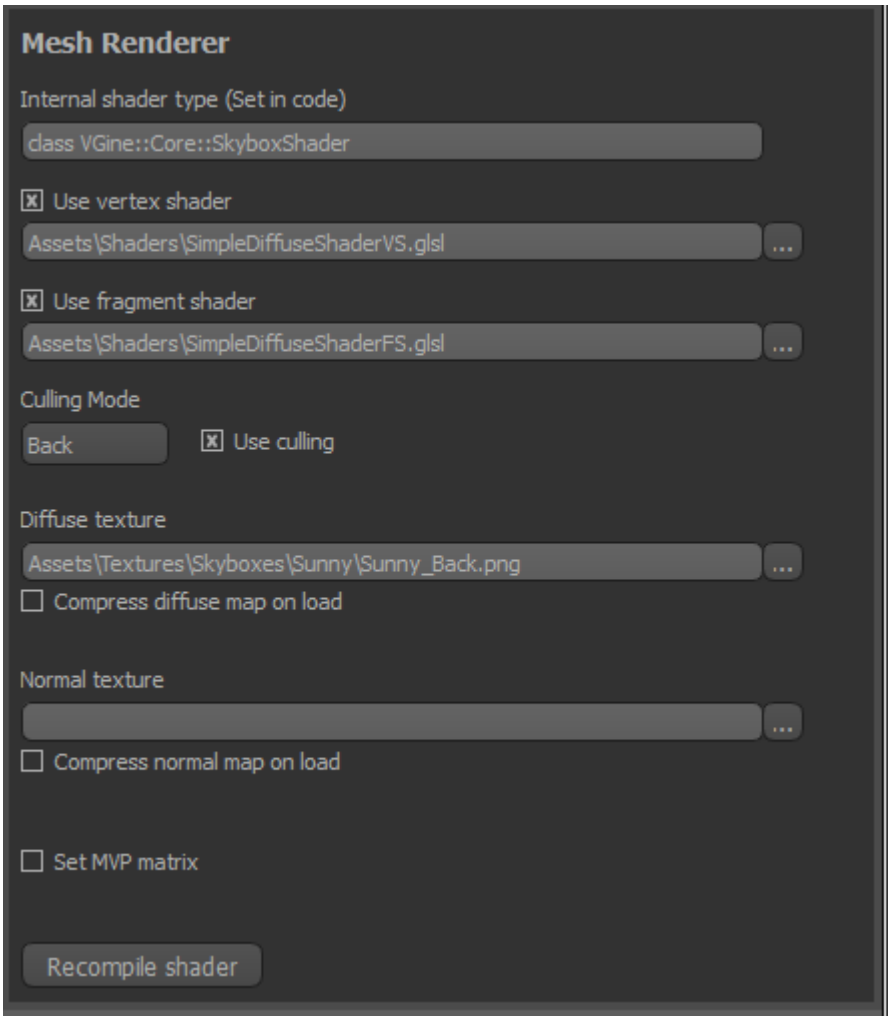
Faites attention, seul l'éditeur de transformation apparaitra si vous sélectionnez plusieurs objets, celui de la caméra n'apparaitra pas.



Modifications des paramètres de rendu

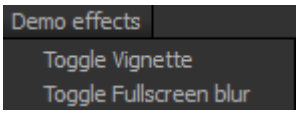
Il est possible de modifier plusieurs paramètres de rendus d'un objet si un « MeshRenderer » est attaché à l'objet. Par exemple, il est possible de changer les shaders en temps réel, le mode de culling et même les textures utilisés par le modèle. Il est aussi possible de compresser les textures en DXT5 en cochant la case « Compress on load ».

Faites attention, plusieurs changements demande de recompiler le shader en appuyant sur le bouton « Recompile Shader »



Effet en plein écran

Il existe deux types d'effets en plein écran. Il est possible d'ajouter un effet de vignettage ainsi qu'un effet de flou à la scène en entier. Pour les activer, il suffit d'utiliser le menu « Demo effects ».



Interactivité

Il est possible d'interagir avec la scène avec le clavier et la souris. Lorsque la vue de scène est sélectionnée, il est possible de déplacer la caméra principale de la scène en utilisant les touches du clavier W-A-S-D. Il est aussi possible de faire pivoter la caméra principale en enfonçant le bouton gauche de la souris. Un mouvement de bas en haut fera pivoter la caméra selon l'axe des X multiplié par la rotation de la caméra alors qu'un mouvement gauche à droite fera pivoter la caméra selon l'axe des Y.

Ressources

Les ressources utilisées par l'éditeur se trouvent dans le fichier /Assets

Nom	Type	Auteur
DawnDusk_back.png	Skybox, couché/levé soleil	Viens de base avec Unity3D
DawnDusk_down.png	Skybox, couché/levé soleil	Viens de base avec Unity3D
DawnDusk_front.png	Skybox, couché/levé soleil	Viens de base avec Unity3D
DawnDusk_left.png	Skybox, couché/levé soleil	Viens de base avec Unity3D
DawnDusk_right.png	Skybox, couché/levé soleil	Viens de base avec Unity3D
DawnDusk_up.png	Skybox, couché/levé soleil	Viens de base avec Unity3D
Sunny_Back.png	Skybox journée ensoleillée	Viens de base avec Unity3D
Sunny_Down.png	Skybox journée ensoleillée	Viens de base avec Unity3D
Sunny_Front.png	Skybox journée ensoleillée	Viens de base avec Unity3D
Sunny_Left.png	Skybox journée ensoleillée	Viens de base avec Unity3D
Sunny_Right.png	Skybox journée ensoleillée	Viens de base avec Unity3D
Sunny_Up.png	Skybox journée ensoleillée	Viens de base avec Unity3D
Skull.obj	Modèle 3D d'un crâne	Alexandre Blaquière
SkullLightningMap.png	Texture diffuse d'un crâne	Alexandre Blaquière
SkullNormalMap.png	Texture normale d'un crâne	Alexandre Blaquière
TeethLightningMap.png	Texture diffuse des dents d'un crâne	Alexandre Blaquière
TeethNormalMap.png	Texture normale des dents d'un crâne	Alexandre Blaquière
Brick0Diffuse.png	Texture diffuse de briques	www.rendertextures.com
Brick0Normal.png	Texture normale de briques	www.rendertextures.com
Brick0Height.png	Texture de hauteur de briques	www.rendertextures.com
ConcereteDiffuse.png	Texture diffuse de béton	www.rendertextures.com
ConcereteNormal.png	Texture normale de béton	www.rendertextures.com
ConcereteDisplacement.png	Texture de hauteur de béton	www.rendertextures.com
Pecan Oak_DIFFUSE.png	Texture diffuse de bois	www.rendertextures.com
Pecan Oak_NORMAL.png	Texture normale de bois	www.rendertextures.com
Pecan Oak_DISP.png	Texture de hauteur de bois	www.rendertextures.com

Grass.png	Texture utilisé par le terrain dans la scène 5.	www.rendertextures.com
Rock.png	Texture utilisé par le terrain dans la scène 5.	www.rendertextures.com
Snow.png	Texture utilisé par le terrain dans la scène 5.	www.rendertextures.com
TerrainHeight.png	Texture utilisé par le terrain dans la scène 5.	http://images.google.ca/ (J'ai perdu la source, désolé)
BlurVS.glsl	Vertex shader. Produit un effet de flou en plein écran	Alexandre Blaquière
BlurFS.glsl	Fragment shader. Produit un effet de flou en plein écran	Alexandre Blaquière
DisplacementVS.glsl	Vertex shader. Tesselation, normal mapping, texture, height mapping.	Alexandre Blaquière
DisplacementTC.glsl	Tesselation control shader. Tesselation, normal mapping, texture, height mapping.	Alexandre Blaquière
DisplacementTE.glsl	Tesselation evaluation shader. Tesselation, normal mapping, texture, height mapping.	Alexandre Blaquière
DisplacementFS.glsl	Fragment shader. Tesselation, normal mapping, texture, height mapping.	Alexandre Blaquière
TerrainVS.glsl	Vertex shader. Tesselation, normal mapping, texture, height mapping.	Alexandre Blaquière
TerrainTC.glsl	Tesselation control shader. Tesselation, normal mapping, texture, height mapping.	Alexandre Blaquière
TerrainTE.glsl	Tesselation evaluation shader. Tesselation, normal mapping, texture, height mapping.	Alexandre Blaquière
TerrainFS.glsl	Fragment shader. Tesselation, normal mapping, texture, height mapping. Texture selon la hauteur des sommets.	Alexandre Blaquière

FullVertexDefinition.glsl	Fichier inclus dans tous les Vertex shader. Contiens le format d'entrée des vertex shaders.	Alexandre Blaquière
Lighting.glsl	Contiens toutes les équations servant à faire l'éclairage. Toutes les équations fonctionnent avec Blinn-phong.	Alexandre Blaquière
LightingStructures.glsl	Contiens les structures avec lesquelles l'information des lumières est envoyée du programme vers la carte graphique.	Alexandre Blaquière
NoLight.VS	Vertex Shader. Dessine une texture sans aucune lumière.	Alexandre Blaquière
NoLight.FS	Fragment Shader. Dessine une texture sans aucune lumière	Alexandre Blaquière
PositionColorVS.glsl	Vertex shader qui affiche la couleur assignée aux vertices	Alexandre Blaquière
PositionColorFS.glsl	Fragment shader qui affiche la couleur assignée aux vertices	Alexandre Blaquière
RenderVS.glsl	Vertex shader qui dessine sur un quad une texture. Utilisé par le moteur de rendu pour afficher la scène au final.	Alexandre Blaquière
RenderFS.glsl	Fragment shader qui dessine sur un quad une texture. Utilisé par le moteur de rendu pour afficher la scène au final.	Alexandre Blaquière
TexturedBumpedSpecVS.glsl	Vertex Shader. Lumière altérée à l'aide de normal mapping. Texture.	Alexandre Blaquière
TexturedBumpedSpecFS.glsl	Fragment Shader. Lumière altérée à l'aide de normal mapping. Texture.	Alexandre Blaquière
TexturedSpecularVS.glsl	Vertex Shader. Lumière spéculaire. Texture	Alexandre Blaquière
TexturedSpecularFS.glsl	Fragment Shader. Lumière spéculaire. Texture	Alexandre Blaquière

VignetteVS.glsl	Vertex Shader. Effet en plein écran de vignettage.	Alexandre Blaquière
VignetteFS.glsl	Fragment shader. Effet en plein écran de vignettage.	Alexandre Blaquière

Architecture logicielle

Division des projets

L'éditeur de scène est divisé en trois projets différents. Le premier projet, nommé CommonToolboxCPP est une libraire qui contient du code utilitaire, par exemple du code pour faire du déverminage.

Le deuxième projet, nommé VGineCore est le cœur de l'application. Cette partie du logiciel ne se préoccupe que de garder en mémoire les objets de la scène, les mètres à jour et aussi de dessiner sur un contexte OpenGL.

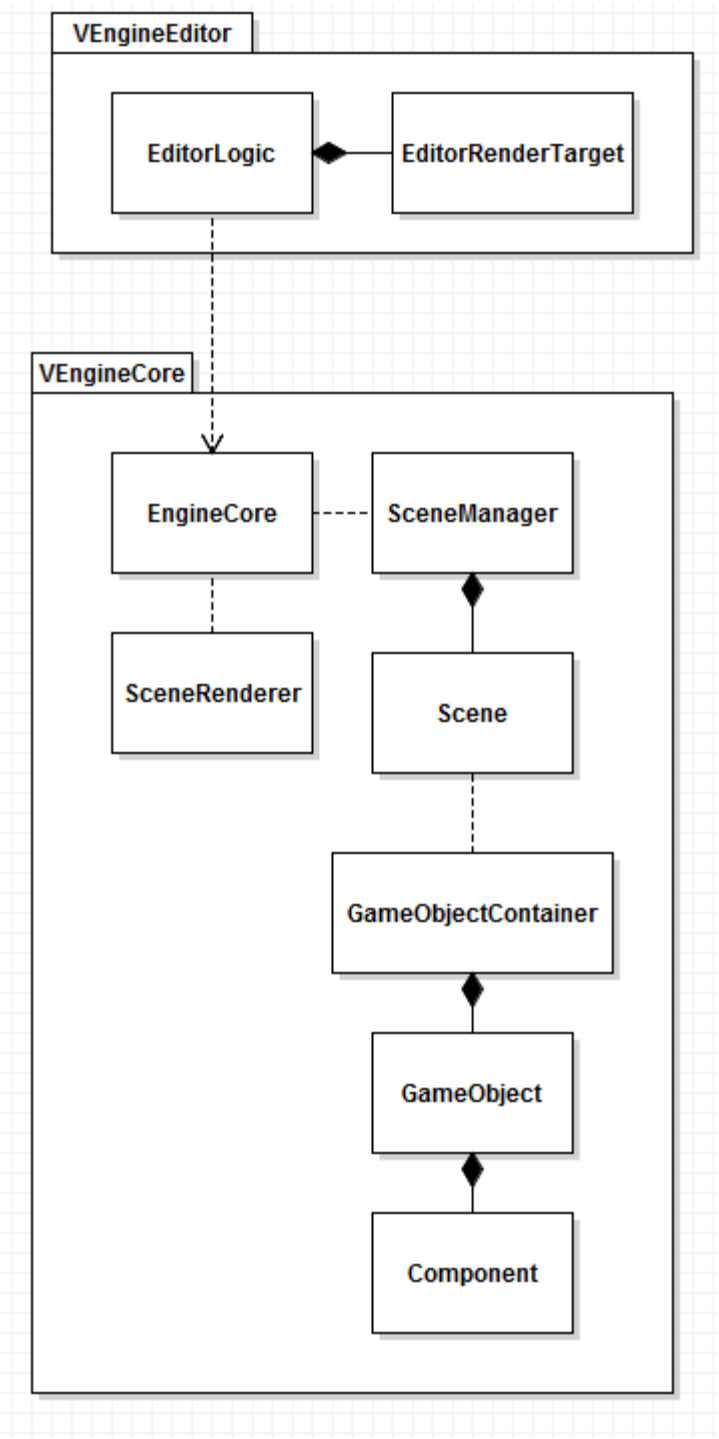
Le dernier projet, nommé VGineEditor est un projet Qt permettant de modifier des paramètres du projet VGineCore et d'indiquer au cœur quelles opérations effectuer (Chargers des modèles, charger des textures, etc.). VGineCore ne connait donc pas VGineEditor et VGineCore s'attend donc à se faire passer un contexte OpenGL valide sur lequel il peut dessiner.

Objets de scène

Pour ce qui est des objets de scène, VGineCore utilise un système semblable à Unity3D, soit un système où un « GameObject » est créé, ajouté à la scène et où il est possible par la suite d'y ajouter des « Component ». La classe abstraite « Component » contient des fonctions abstraites comme Update()qui sera appelé à chaque fois que la boucle Update du moteur sera appelée et OnDraw() qui permettra de faire certaines manipulations lorsqu'un objet est pour être rendu.

Il existe aussi plusieurs composantes prédéfinies par le moteur, soit « Transform » , «MeshFilter », « MeshRenderer » et « Camera ». En ordre, le transform est une composante qui garde en mémoire la position, la rotation et la grosseur d'un objet. Un MeshFilter peut contenir une « Mesh » qui à son tour peux contenir des vertices, des normales, des indices, des tangentes et des bitangentes. Un MeshRenderer peut contenir un programme de shader OpenGL ainsi que des textures. Et finalement, la composante « Camera » permet d'attacher à un objet une caméra pour pouvoir ensuite contenant des paramètres tel que le champ de vision.

Voici un diagramme UML de haut niveau qui met les parties les plus importantes classes de l'éditeur en valeur.



Compilation des shaders

Avant d'être compilé par OpenGL, les shaders sont soumis à un petit préprocesseur permettant de faire des inclusions d'autres fichiers. Il faut donc faire attention si l'on désirerait utiliser ces shaders dans une autre application. Le préprocesseur se situe dans la classe nommée « ShaderPreprocessor».

Boucle de rendu

La boucle de rendu commence lorsqu'un contexte OpenGL est crée du bord de l'éditeur. Après un intervalle de temps fixe, un QTimer (une classe faisant un appel de fonction à un intervalle fixe), itère au travers de la liste de tous les contextes valides OpenGL destiné à être dessiné. Si ce contexte(appelé EditorRenderTarget dans le code) contient un « Viewport » et une « Camera », alors il sera envoyé à la classe « SceneRenderer ». C'est dans cette classe que la majorité de la logique du dessinage se passe. Cette classe s'attend à recevoir une « Camera », un « Viewport » et une « Scene ». Avant de faire quoi que ce soit, une petite classe appelée « RenderInfo » sera crée contenant des informations telles que la matrice de vue et de projection calculée une seule fois pour pouvoir ne pas a avoir à les calculer plusieurs fois, toutefois, les shaders sont libres d'outrepasser tous les paramètres dans la classe « RenderInfo ». Par la suite, le « SceneRenderer » itèrera au travers de tous les « GameObject » dans la scène, attachera à OpenGL tous les shaders, les textures et les modèles dans les composantes rattachées aux « GameObjects » pour en faire le rendu.

Pour améliorer la performance et permettre de faire des effets en plein écran, tout dessin est d'abord fait dans un framebuffer. C'est la classe nommée « RenderBuffer » qui encapsule ce concept, et qui permet d'attacher en alternance deux framebuffers pour pouvoir accumuler des effets en plein écran.

Plateforme technologique

Libraires externes

Ce projet utilise 6 librairies externes.

Assimp : Assimp est une libraire permettant l'importation de modèles 3D. Les modèles importés à l'aide de Assimp sont transformés à l'interne et stockés à l'intérieur de la classe Mesh.

Glew : Glew permet de découvrir les fonctions et les extensions OpenGL disponibles.

Glm : Une librairie mathématique. Les positions, la rotation ainsi que la taille de tous les éléments sont calculées à l'aide de cette libraire. Les matrices issues de la caméra sont aussi calculées à l'aide de glm.

irrKlang : Une libraire permettant de faire jouer du son. Cette librairie n'a malheureusement pas été utilisée pour le travail pratique.

lodePNG : Une librairie permettant de charger des images en mémoire. Typiquement, les images sont chargées à l'intérieur de la classe Texture2D pour être envoyées vers OpenGL.

Qt : Les fenêtres de l'application ont été faites à l'aide de Qt. L'outil QtDesigner à été utilisé pour faire le design des fenêtres.

Visual studio : Tout le code a été développé sur Windows à l'aide de visual studio. Pour faciliter l'intégration avec Qt, le « add-in » de Qt a été utilisé.

Compilation

La seule dépendance non fournie pour la compilation du projet est Qt, car cette dépendance pèse 568MB compressé. Si une recompilation serait nécessaire, il faudrait télécharger la version fournie de Qt(le lien est dans le readme situé dans le dossier source) et le add-in de visual studio de Qt pour indiquer où la dépendance se situe. Sinon, toutes les « property sheets » utilisent des chemins relatifs et les ressources seront copiées lors du processus de build. La version de visual studio utilisé pour ce TP était visual studio 2012 professional. Attention aux espaces dans les noms de dossier, cela fera échouer certaines parties du build.

Présentation de l’équipe

Présentation de seul membre de l’équipe : Alexandre Blaquière

Je suis présentement étudiant en 2e année au baccalauréat en informatique et je suis un passionné de la programmation et de l’informatique en général. J’ai commencé à programmer il y a maintenant 6 ans des petits jeux en XNA avec C# et je désire faire carrière avec dans un relié aux jeux ou à l’animation 3D.