Contents

[Assimp - Open Asset Import Library 2](#_Toc128139532)

[Bullet 3](#_Toc128139533)

[**Fonctionnalités** 3](#_Toc128139534)

[GLFW 4](#_Toc128139535)

[**Fonctionnalités** 4](#_Toc128139536)

[QT6 5](#_Toc128139537)

[**Essentiels Qt** 6](#_Toc128139538)

[QtAdvencedDocking 7](#_Toc128139539)

[SpirV 8](#_Toc128139540)

[**Fonctionnalités** 9](#_Toc128139541)

[Stb\_image 9](#_Toc128139542)

[**Fonctionnalités** 10](#_Toc128139543)

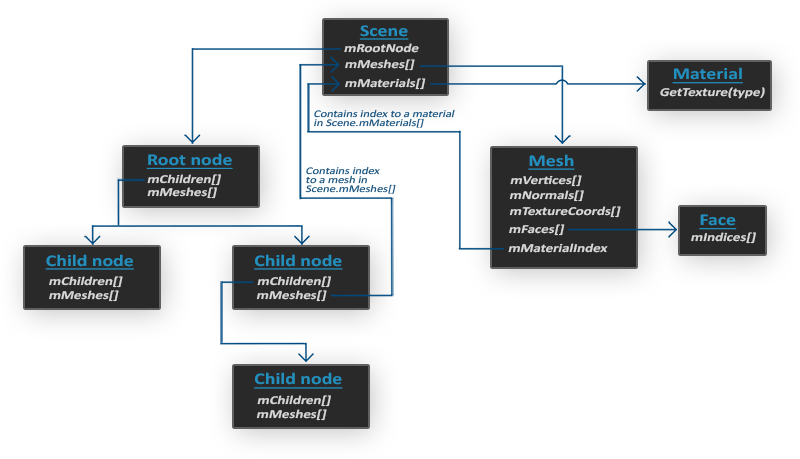
[VkBootstrap 10](#_Toc128139544)

[Vulkan memory allocator 11](#_Toc128139545)

# Assimp - Open Asset Import Library

Assimp peut importer des dizaines de formats de fichiers modèles (et peut aussi les exporter), en archivant les données du modèle dans des structures de données générales. Dès qu’Assimp charge le modèle, on peut retrouver toutes les données du modèle dans des structures de données propres à Assimp. Ces structures de données restant toujours les mêmes, quel que soit le format du fichier importé, cela nous permet de s’affranchir des formats utilisés pour les fichiers importés.

Assimp charge le modèle entier dans un objet Scene qui contient toutes les données du modèle importé. Assimp contient alors un ensemble de nœuds, chaque nœud contenant des indices pointant vers les données de l’objet Scene, un nœud pouvant avoir un nombre quelconque de nœuds fils. Un modèle (simplifié) de la structure des données d’Assimp est figuré ci-dessous :

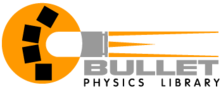


# Bullet

Bullet est un moteur physique simulant la détection de collisions ainsi que la mécanique des corps rigides et déformables.

### **Fonctionnalités**

* Détection de collision pour les formes primitives : sphère, parallélépipède rectangle, cylindre, cône, coque convexe et maillage de triangles
* Calcul de distance entre objets convexes par l'algorithme GJK
* Détection de collisions par balayage
* Détection de collision continue (CCD)
* Contraintes



# GLFW

Une bibliothèque légère, libre, multi-plateforme et open source pour le développement d'applications desktop OpenGL, OpenGL ES et Vulkan. Elle fournit aux programmeurs la possibilité de créer et de gérer des fenêtres et des contextes pour ces interfaces graphiques, et le support des joystick, clavier et souris.

### **Fonctionnalités**

* Opengl contexte en 2 appelle de fonction
* Support OpenGL, OpenGL ES et Vulkan
* Support multiple fenetre
* Support clavier, manette, sourie

Logo GLFW

# QT6

Qt est programmé en C++, le langage de programmation étant complété avec le préprocesseur MOC (Meta-Object Compiler) qui apporte des fonctionnalités telles que le mécanisme élémentaire signaux/slots (permettant la communication déclenchée par un événement entre les objets du programme). Pour ce faire, le préprocesseur génère avant même la compilation un C++ conforme aux normes à partir du code source Qt. Par conséquent, les applications Qt peuvent être traduites avec des compilateurs C++ courants tels que GCC, ICC, MinGW ou MSVC. Les versions les plus récentes d’infrastructures offrent d’autre part un accès au langage de balisage QML propre à Qt qui apporte des simplifications en particulier dans le cadre du développement d’IGU. En dehors de ces solutions de langage internes, il prenden charged’autres langages de programmation mis à dispositions par des prestataires tiers tels que Python, Ruby, Go, Java ou PHP.

### **Essentiels Qt**

Les Essentiels Qt constituent **la base de Qt**, quelle que soit la plateforme. Ils sont disponibles sur toutes les plateformes de développement supportées ainsi que sur toutes les plateformes cibles testées et sont pertinents et utiles pour une grande partie des applications développées. Les modules suivants font partie des composants essentiels de l’infrastructure :

| Modules | Description |
| --- | --- |
| Qt Core | Classes clés non graphiques utilisées par tous les autres modules |
| Qt GUI | Classes de base pour la conception d’interfaces graphiques utilisateur ; comprend OpenGL |
| Qt Multimedia | Ensemble de types QML et de classes C++ pour le traitement de contenus multimédias (audio/vidéo) ; IPA pour l’accès aux fonctionnalités de caméra et de radio |
| Qt Multimedia Widgets | Classes basées sur les widgets pour l’implémentation de fonctionnalités multimédias |
| Qt Network | IPA pour les applications accédant à des réseaux TCP/IP |
| Qt QML | Infrastructure et types de langage de balisage Qt QML |
| Qt Quick | Infrastructure déclarative pour le développement d’applications hautement dynamiques avec des interfaces utilisateur QML taillées sur mesure |
| Qt Quick Controls 2 | Extension Qt Quick : types QML légers, très efficaces simplifiant la création d’interfaces |
| Qt Quick Dialogs | Extension Qt Quick : types pour la création et l’interaction avec les dialogues système |
| Qt Quick Layouts | Extension Qt Quick : types QML pour l’agencement d’objets dans l’interface développée |
| Qt Quick Test | Infrastructure test pour les applications QML ; les cas de test sont décrits comme fonctionnalités JavaScript |
| Qt SQL | Classes pour l’intégration de banques de données SQL |
| Qt Test | Classes pour les tests détaillés des applications et les bibliothèques Qt |
| Qt Widgets | Ensemble d’éléments d’IU permettant de créer des interfaces utilisateur classiques basées sur des widgets avec le Qt Designer |

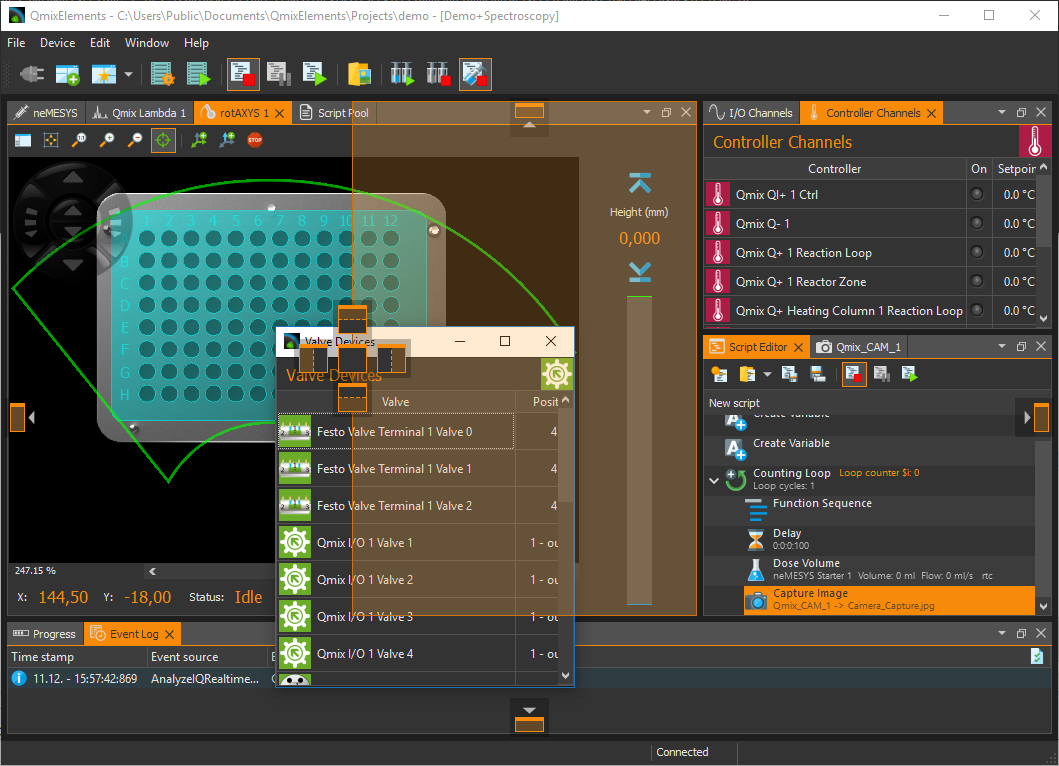
A green and white sign

Description automatically generated with low confidence

# QtAdvencedDocking

Qt Advanced Docking System lets you create customizable layouts using a full featured window docking system similar to what is found in many popular integrated development environments (IDEs) such as Visual Studio.

Il utilise le systeme de docking de Qt, et rajoute une couche plus avancer





# SpirV

SPIRV-Reflect is a lightweight library that provides a C/C++ reflection API for SPIR-V shader bytecode in Vulkan applications.

SPIRV-Reflect has been tested on Linux and Windows.

### **Fonctionnalités**

* Extract descriptor bindings from SPIR-V bytecode, to assist in the generation of Vulkan descriptor set and pipeline layouts.
* Extract push constant block size from SPIR-V bytecode to assist in the generation of pipeline layouts.
* Extract full layout data for uniform buffer and push constant blocks from SPIR-V bytecode, to assist in application updates of these structures.
* Extract input/output variables from SPIR-V bytecode (including semantic decorations for HLSL shaders), to assist in validation of pipeline input/output settings.
* Remap descriptor bindings at runtime, and update the source SPIR-V bytecode accordingly.
* Log all reflection data as human-readable text.



# Stb\_image

bibliothèque de chargement d’image. Single-file public domain (or MIT licensed) libraries for C/C++

### **Fonctionnalités**

* image loader: stb\_image.h
* image writer: stb\_image\_write.h
* image resizer: stb\_image\_resize.h
* font text rasterizer: stb\_truetype.h
* typesafe containers: stb\_ds.h

# VkBootstrap

A utility library that jump starts initialization of Vulkan

This library simplifies the tedious process of:

* Instance creation
* Physical Device selection
* Device creation
* Getting queues
* Swapchain creation

It also adds several conveniences for:

* Enabling validation layers
* Adding a debug callback messenger
* Enabling extensions on a physical device
* Select a gpu based on a set of criteria like features, extensions, memory, etc

# Vulkan memory allocator

Memory allocation and resource (buffer and image) creation in Vulkan is difficult (comparing to older graphics APIs, like D3D11 or OpenGL) for several reasons:

It requires a lot of boilerplate code, just like everything else in Vulkan, because it is a low-level and high-performance API.

There is additional level of indirection: VkDeviceMemory is allocated separately from creating VkBuffer/VkImage and they must be bound together.

Driver must be queried for supported memory heaps and memory types. Different GPU vendors provide different types of it.

It is recommended to allocate bigger chunks of memory and assign parts of them to particular resources, as there is a limit on maximum number of memory blocks that can be allocated.

Features

This library can help game developers to manage memory allocations and resource creation by offering some higher-level functions:

* Functions that help to choose correct and optimal memory type based on intended usage of the memory.
* Functions that allocate memory blocks, reserve and return parts of them (VkDeviceMemory + offset + size) to the user.
* Functions that can create an image/buffer, allocate memory for it and bind them together - all in one call.