Tech stack :

* Glfw : windowing
* IMGUI : User Interface
* GLAD : OpenGL abstraction layer
* OpenGL : Render API
* PhysX : Physique
* SoLoud : audio
* Assimp : 3D model importer
* Stb\_image : Texture importer

Required Features :

* Buffers
* Shaders
* Camera
* Input
* Textures
* Models

Documentation du Projet C++ - Survivant

Introduction

Le but du projet Survivant est de créé un moteur de jeux. En équipe 3, nous devrons rechercher, structurer, projeter et annoter notre plan pour s’attaquer à cette charge. Après ça complétion, nous utiliserons notre moteur pour développer un FPS puzzle-game plutôt simple.

Configuration du Projet

Fichiers Sources

Bibliothèques Externes

Dans cette sous-section, ils s’y trouvent la liste des bibliothèques dont nous allons utiliser dans ce projet ainsi une courte explication de leurs rôle et importance de celui-ci (Fonctionnement, Fichier Source, Type de Bibliothèque, Chargement).

- Explication de son fonctionnement

- Justification de la librairie

- Chargement de la bibliothèque

- Intégration des fichiers sources (.lib et ou .dll, ou généré avec fetchContent)

- Bibliothèque statique(.lib)/dynamique(.dll)

1. GLFW (Windowing)

(Fonctionnement) Cette librairie « Open Source » est crucial pour la création et gestion de fenêtre compatible avec multiple API de rastérisation. GLFW fournit une abstraction multi-platform pour des applications graphiques. (Justification) GLFW est l’API de fenêtrage standard de OpenGL. De plus, l’équipe a de l’expérience avec la librairie.

Étapes de chargement :

1. Initialisation avec [glfwInit](https://www.glfw.org/docs/3.3/group__init.html#ga317aac130a235ab08c6db0834907d85e)().
2. Définir la version 4.6 avec [glfwWindowHint](https://www.glfw.org/docs/3.3/group__window.html#ga7d9c8c62384b1e2821c4dc48952d2033)([GLFW\_CONTEXT\_VERSION\_MAJOR](https://www.glfw.org/docs/3.3/group__window.html#gafe5e4922de1f9932d7e9849bb053b0c0), 4) et [glfwWindowHint](https://www.glfw.org/docs/3.3/group__window.html#ga7d9c8c62384b1e2821c4dc48952d2033)([GLFW\_CONTEXT\_VERSION\_MINOR](https://www.glfw.org/docs/3.3/group__window.html#ga31aca791e4b538c4e4a771eb95cc2d07), 6).
3. Créé une fenêtre par default avec [glfwCreateWindow](https://www.glfw.org/docs/3.3/group__window.html#ga3555a418df92ad53f917597fe2f64aeb)(hauteur, largeur, titre).
4. Utilisé la librairie dans la loupe principale.
5. Sélectioner la fenêtre avec [glfwMakeContextCurrent](https://www.glfw.org/docs/3.3/group__context.html#ga1c04dc242268f827290fe40aa1c91157)(fenêtre).
6. Finir en détruisant les fenêtre avec [glfwDestroyWindow](https://www.glfw.org/docs/3.3/group__window.html#gacdf43e51376051d2c091662e9fe3d7b2)(fenêtre) et ensuite [glfwTerminate](https://www.glfw.org/docs/3.3/group__init.html#gaaae48c0a18607ea4a4ba951d939f0901)().

Fichier Source :

Type de Bibliothèque :

Sources : <https://www.glfw.org/docs/3.3/quick.html>

2. ImGui (User Interface)

(Fonctionnement) ImGui est une librairie d’interface graphique développer pour assister au développement d’engin de jeux. (Justification) ImGui est un API très légère qui ne contient aucune dépendance qui prioritise la factibilité d’itération et la visualisation simple.

Étapes de chargement :

1. Initialiser le contexte avec IMGUI\_CHECKVERSION() et ImGui::CreateContext().
2. Configuré les flags selon les types d’input (ex : io.ConfigFlags |= ImGuiConfigFlags\_NavEnableKeyboard).
3. Initialiser le ‘’backend’’ avec ImGui\_ImplWin32\_Init(YOUR\_HWND) et

ImGui\_ImplDX11\_Init(YOUR\_D3D\_DEVICE, YOUR\_D3D\_DEVICE\_CONTEXT).

1. Initialisé au début et à la fin de la loupe principale.
2. Finir avec ImGui\_ImplDX11\_Shutdown() et ImGui\_ImplWin32\_Shutdown() pour ensuite détruire avec ImGui::DestroyContext();

Fichier Source :

Type de Bibliothèque :

Source : <https://github.com/ocornut/imgui?tab=readme-ov-file>

3 GLAD (OpenGL Abstraction Layer)

[...]

4 OpenGL (Render API)

[...]

5 PhysX (Physique)

(Fonctionnement) La librairie PhysX permet de définir des acteurs et les déplacer en suivant les lois de mécaniques classiques dans un environnement trois-dimensionnel. Elles supportent aussi les collision, jointures, actionneur et des tests physique de rayons, de balayage et de chevauchement.

(Justification) PhysX est une librairie de bas niveau très puissante et complexe qui nous permettras de créer un environnement réaliste. Elle peut même être configuré pour utiliser la GPU pour certaine tache demandant (CUDA).

Étapes de chargement :

1. Initialiser avec PxCreateFoundation(PX\_PHYSICS\_VERSION, callback, error\_callback)
2. Connecter la foundation a un socket avec mPvd = PxCreatePvd(\*gFoundation), PxPvdTransport\* transport = PxDefaultPvdSocketTransportCreate(PVD\_HOST, 5425, 10), mPvd->connect(\*transport,PxPvdInstrumentationFlag::eALL). (Cooking?)
3. Créé un objet physique avec mPhysics = PxCreatePhysics(PX\_PHYSICS\_VERSION, \*mFoundation,

PxTolerancesScale(), recordMemoryAllocations, mPvd).

1. Utilisé des ‘’callback‘’ dans la loupe principale.
2. Finir avec mPhysics->release() et mFoundation->release().

Fichier Source :

Type de Bibliothèque :

Source : <https://gameworksdocs.nvidia.com/PhysX/4.0/documentation/PhysXGuide/Manual/Introduction.html#a-brief-overview-of-physx>

<https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/apireference/files/group__foundation.html>

6 SoLoud (Audio)

(Fonctionnement) SoLoud est un engin créé pour simplifier la tâche de la gestion audio. (Justification) Malgré la simplicité d’utilisation de cette librairie, SoLoud contient une multitude de fontinalité tel que l’ajustabilité du volume et de la vitesse, modifié l’audio avec des filtres, ajouté des sons a une queue, du support pour différent format de fichier et de la loupe de musique sans écart.

Étapes de chargement :

1. Initialiser un ‘’core’’ avec SoLoud::Soloud gSoloud et gSoloud.init().
2. Importer un sons SoLoud::Wav gWave avec gWave.load("pew\_pew.wav").
3. Modifier le sons si désiré, pour ensuite le jouer avec gSoloud.play(gWave).
4. Finir avec gSoloud.deinit()

Fichier Source :

Type de Bibliothèque :

Source : <https://solhsa.com/soloud/index.html>

7 ***Assimp (3D Model Importer)***

La bibliothèque Open Asset Import Library (Assimp) permet le chargement et la manipulation de modèles 3D provenant de divers formats de fichiers. Elle offre une solution puissante et flexible pour traiter des données de modèles 3D dans une application. Assimp prend en charge une variété de formats, facilitant l'importation de modèles provenant de différentes sources.

***Initialisation :***

* Utilisez la fonction **aiCreateDefaultIOSystem** pour créer un système d'E/S par défaut.
* Initialisez la bibliothèque avec **aiImportFile** en spécifiant le chemin du fichier et les drapeaux d'importation nécessaires.

***Manipulation de la scène :***

Accédez aux données de la scène via la structure aiScene.

Parcourez les nœuds (aiNode) pour obtenir des informations sur la hiérarchie du modèle.

***Libération des ressources :***

Libérez la mémoire utilisée par la scène avec aiReleaseImport.

Source: https://www.assimp.org/

8 ***Stb\_image (Texture Importer)***

La bibliothèque stb\_image est une bibliothèque simple et légère permettant le chargement d'images dans divers formats. Elle offre une solution rapide et facile à utiliser pour incorporer des images dans des applications graphiques.

***Téléchargement et Intégration :***

. Téléchargez le fichier d'en-tête stb\_image.h depuis le dépôt officiel.

Incluez le fichier d'en-tête dans votre projet.

***Chargement de l'Image :***

Utilisez la fonction **stbi\_load** pour charger une image depuis un fichier.

Accédez aux pixels de l'image et aux informations telles que la largeur, la hauteur et le nombre de canaux.

***Libération de la mémoire :***

Libérez la mémoire allouée pour l'image avec la fonction stbi\_image\_free une fois que vous avez fini de l'utiliser.

Source : https://github.com/nothings/stb/blob/master/stb\_image.h

Conclusion

Résumez brièvement les points clés de votre documentation.