Technical Design Document for Pandor Engine

Summary

Nomenclature

To top of file

Naming convention :

- Fonctions : SnakeCase
- Variables : camelCase

- public : camelCase
- private : m_camelCase
- protected : p_camelCase
- static : s_SnakeCase

Commentary convention :

To comment the code we decided to always comment before the line or the function except if it's for a variable.

> *Example :*
>
> */* Comment this function */*
>
> *void FunctionName();*
>
> *int a; // This variable is used to ...*

Order in classes :

Private Variable :

Protected Variable :

Public Variable :

Private Fonction :

Protected Fonction :

Public Fonction :

Class variables and naming convention:

We will use all variables in private and protected unless it's really necessary to use public and we will add Getters and Setters to the public functions of the class to have control over the acess of every class variables

To name functions , files, folders and namespaces we will always use SnakeCase

Bracket norms :

Work for functions, if/else , for loop, classes, etc...

if()

> *one line of code only*

else

> *one line of code only*

if()

{

> *multiple lines of code*

}

else

{

> *one or more line of code*

}

Namespaces :

1 Namespace = 1 folder

There can also be some specific namespaces inside one of those and thus no specific directory will be created
We use "Using namespace" only in cpp files and only for the namespace of the file classe but we can also use it for the Maths or Physics namespace if needed

List of namespaces :

- Core : everything else
- Wrapper : The namespace Wrapper is in Core but allow more precision (then there is one namespace by wrapper in this)
- Component : All basics component for the objects
- Math : Math library
- Render : Everythink linked to the render, the shaders, openGl , etc..
- LowRenderer : For the lights
- Resources : Every resources or linked to the Resources Manager
- Utils : File loader, parsing, input handler
- Physic : Physics Library
- Debug : Logs, console, output, asserts
- EditorUI : The interface of the engine, like ImGui
- Scripting : Handle the function and classes needed for the scripting

Libraries used

To top of file

List of libraries :

- Graphics API : OpenGL -> Link : Wrapper (https://docs.google.com/spreadsheets/d/1fXyijKYIah5L2H-D8j5Tq8nMZ_qwe0nkNNSenTW3wBA/edit?usp=sharing)
- Interface : Dear ImGui
- FBX loader : ofbx
- Texture and image loading : STB Image
- Desktop : GLFW
- Physics : PhysX
- Sound : MiniAudio
- Scripting : Mono
- Others :
  - - Font Library : Freetype
  - - TextEditor : ImGuiColorTextEdit
  - - Hot Reload : FileWatch

STB_Image :

Link : stb_image (https://github.com/nothings/stb)

In order to parse the images and textures in the engine we decided to use the stb_image library.
It's a single file library which make it really easy to use and install in the engine.
We choose this library because it's a really easy to use library and we already used it a few times so we won't have to learn how to use it.
We also could have rewrite our own image parser but it would have been time consuming, and it wouldn't be as efficient as stb_image.

Difficulties :

There was no difficulty with this library since we already used it a lot

Dear ImGui :

Link : DearImGUI (https://github.com/ocornut/imgui/tree/docking)

For the management of the interface we chose to use the docker version of imgui which allows to have a clear and easy to use interface as well as the possibility to manage the windows as you want and not to be stuck in the main window.

Positives :

- ImGui is a library that we have used a lot, so we know how to use it and its different features.
- The dock version of ImGui will allow greater flexibility in window positioning.
- ImGui is easy to use and to implement
- It's a very used and documented library

Difficulties :

There was no difficulty with this library since we already used it a lot and we already explored a lot of functionalities

Physics :

Link : Physx (https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html)

Link : Physx GitHub (https://github.com/NVIDIAGameWorks/PhysX)

> *PhysX*

Positive points :

- PhysX is multi-platform : Physx and its development kit are available on Microsoft Windows, OS X, Linux, PlayStation 2, PlayStation 3, PlayStation 4, Xbox 360, Xbox One, Wii, Wii U, Android and Apple iOS.
- The development kit is free for commercial and non-commercial use on Microsoft Windows, OS X, Linux, Android and Apple iOS.
- It is an Open Source library, with a github.
- It is a widely used library: it is integrated in several game engines such as Unreal Engine or Unity.
- It is also possible to debug physics with a visual debugger: Physical Visual Debugger (PVD).
- The PhysX library can be static or dynamic depending on how it is used in a program.
- It also has many examples on the github
- Romain had the opportunity to use it for a previous project, so he have some experience on it contrary to the Bullet library.

Negative points:

- However, you have to register with Nvidia to download it.
- It is heavy in terms of file size (~300 MB of static library)

> *Bullet*

Link : Bullet (https://pybullet.org/wordpress/)

Link : Bullet GitHub (https://github.com/bulletphysics)

Positive points :

- Bullet is an OpenSource library which make it flexible
- Bullet works as good on Nvidia than on AMD GPU
- Bullet is really fast to do calculation

Negative points:

- The library hasn't been updated for a long time
- Doesn't work well in multithreading
- The documentation is smaller than PhysX
- The physics calculation isn't has precise than others physics library

> *Our choice : PhysX*

We decided to use the Physx library, the calculation may be slower but they will be more precise and we already

Difficulties :

The order of the calls is really important so we need to be very precise with the order of every call

We still have some issues with the library, the implementation of physx in the engine requires a lot of time

FBX Loader :

Link : OpenFBX (https://github.com/nem0/OpenFBX)

We decided to use OpenFBX library for loading 3D/animation/textures models in .fbx format

Positive points:

- It can load geometry (with UVs, normals, tangents, colors), skeletons, animations, blendshapes, materials and textures.
- Very easy to install.
- Static library, with only 1 ".h" to include.
- Very lightweight in terms of file size (474 kb).
- I (Romain) have had the opportunity to use it in a previous project, so I am familiar with the library.
- Fast loading of resources (0.00375 seconds. for the monkey model).
- Provides an example on Github.
- It is used in several game engines such as Lumix Engine and Flax Engine.

Negative points :

- Library not well-documented and is not widely used.
- It can be complex to understand and use if you have never used an FBX loading library before.

Difficulties :

The parsing of the fbx even using a library is really hard and with OpenFBX the parsing of animations and textures was really difficult

Graphic API:

OpenGL 4.5 vs Vulkan vs DirectX 12

> *DirectX 12*

Link : DirectX (https://microsoft.github.io/DirectX-Specs/)

First we decided to remove DirectX from the equation due to our lack of experience with it. (We had all chosen Vulkan at the last project).

> *Vulkan*

Link : Vulkan (https://www.vulkan.org/)

Positives:

- Developer has great control over all settings and features
- Better performance (if setup in an optimized way)

Negative points:

- Little experience (only one project of about 2 weeks)
- Long to setup (900 lines just to display a triangle)
- Difficult to handle, there are a lot of parameters (this great control implies great responsibilities)

> *OpenGL*

Link : OpenGL (https://www.opengl.org/)

Positives:

- Lots of experience (about 1 year of practice through various projects)
- Simple and fast to use

Negative points:

- Less control and customization which can be very disabling (for optimization for example)

- Nowadays OpenGL is not used anymore for big games

> *OUR CHOICE: OpenGL*

With more experience on Vulkan our choice would have most likely been it.
This API is much more modern than OpenGL and offers better performance and much more customization. However, with so little practice, we would have had to spend a lot of time to fully master this API in order to really take advantage of its benefits. So we choose OpenGL to save precious time that will allow us to focus on the main features of the Pandor Engine.

Difficulties :

We already use this library a lot so we didn't have a lot of issues to report, for every problem that we had we only had to check in our last projects to find the solution

Window Library :

We decided to use GLFW for Main Window Rendering instead of SDL2.
Link : GLFW (https://www.glfw.org/)

Positive points :

- We have had the opportunity to use it many times, unlike SDL2.
- The GLFW library is designed specifically for the creation of OpenGL applications, while SDL2 is a more general multimedia development library. So since it uses OpenGL, GLFW has more optimized and easier features.
- It is lighter than SDL2. While SDL2 offers a wide range of features, knowing that we only need the creation of a window, GLFW focuses on the features necessary for the creation of OpenGL applications, which makes its size smaller.
- It also is very easy to install and configure, with static or dynamic links to libraries. SDL2 can be a bit more complicated to install and configure, with additional dependencies, which can make it less intuitive to use.
- GLFW offers a very efficient cross-platform compatibility. It supports all major platforms, including Windows, macOS and Linux, as well as most mobile platforms.
- It also offers excellent online documentation and numerous online examples to facilitate development.

Negative Points :

- GLFW does not offer high level event management like SDL2 (mouse and keyboard management).
- GLFW does not support reading and writing files, which is a feature that SDL2 offers.

Difficulties :

The library is easy to use and we already used it a lot so no real problem with this one either

Font Library :

We decided to Use the library freetype to load .ttf files

Link : Font (https://freetype.org/)

Positive points :

- It's very simple to use.
- Very well explain in LearnOpenGL : https://learnopengl.com/In-Practice/Text-Rendering.
- Is Multiplatform.
- Still update Frequently.

Negative Points :

- Large file size for font loading only

Difficulties :

There wasn't a lot of problems because learnOpenGL explain this very well. but i still cannot find how to change font Size at runtime

Text Editor Library :

We choose ImGuiColorTextEdit to do a text Editor in the engine

This is a header only library to edit script file like '.glsl' file in runtime with syntax highlighting

Edited by Romain a bit to fit correctly with the engine

Link : TextEditor (https://github.com/BalazsJako/ImGuiColorTextEdit)

Sound Library :

We decided to use Miniaudio for the sound of our Engine

Link : MiniAudio (https://miniaud.io/index.html)

Positive points :

- The documentation is really big and there is a lot of explanation on how to do the multiple features
- The library provide a lot of different functionalities such as spatialization or fading
- We know people that already used it so we could get help if needed

Negative Points :

- Very few tutorials on the web
- Not much of example of specific features

Difficulties :

There wasn't a lot of problems but due to the lack of tutorials every little problem took a while to fix because we had to find everything and solve everything from 0 with almost no help

Scripting Library :

For the C# scripting, we knew that it would take a lot of time because of the limited documentation.
Luckly, we found a recent and well explained tutorial by TheCherno on that subject using the Mono Library. So in order not to waste too much time, we used this library.
Link : Mono (https://github.com/mono/mono)

HotRealod Library:

We used filewatch to implement an easier hot reload feature in the engine This is a header only library that sends events whenever a given file is modified (used for C# scripts hot reload)
Link : filewatch (https://github.com/ThomasMonkman/filewatch)

Engine Core features :

To top of file

Pandor Engine Core feature choice

We wanted to implement a special feature in our engine that would have works like an instant scene switch to be able to develop level on multiple scene and used those scenes at the same time so the user could have created a game with different style and try a lot of interesting mechanics, but due to a lack of time we didn't have the time to implement this feature in our engine for now.

Folders architecture