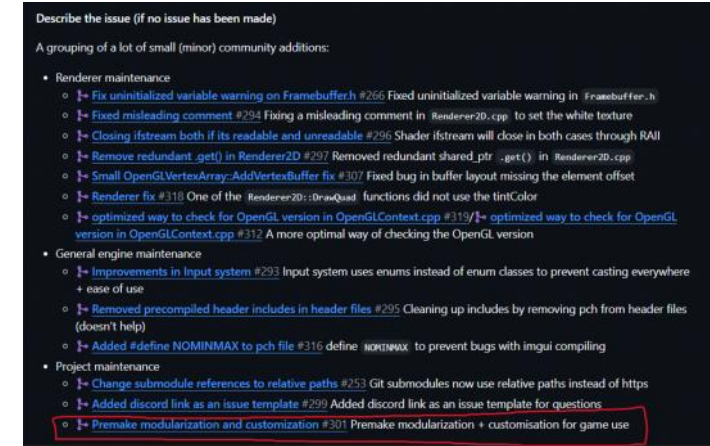


-----Saving & Loading scene-----

》》》更改 Premake 文件构架

这一集中 Cherno 对 premake 文件进行了操作，不过此时 Premake 文件的构架发生了改变（现在每个项目的 premake 被放置在项目的文件夹下，而不是集中放置在 Nut 根目录下的 Premake 文件中），这是因为之前的一次 pull request。
本来准备先完善引擎 UI，后面集中对引擎进行维护，现在看来就先提交一下这个更改吧。

具体可以参考：（<https://github.com/TheCherno/Hazel/pull/320>）

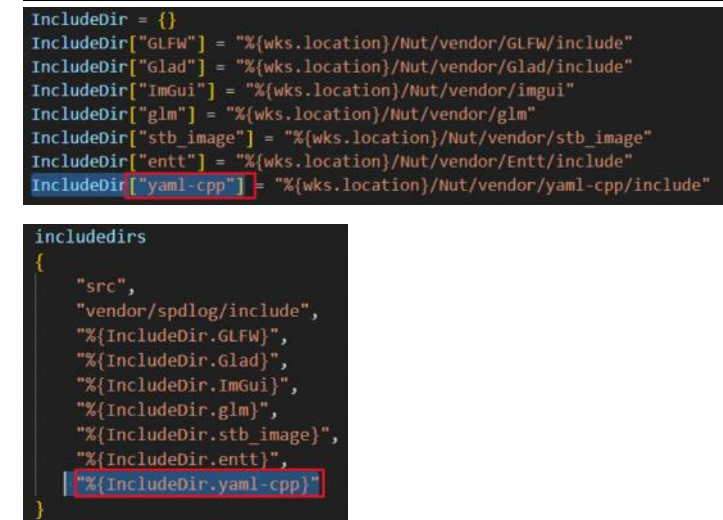


》》》一个问题：关于 premake 文件中的命名

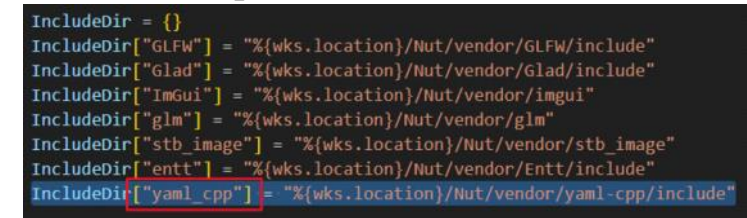
当我将 yaml-cpp 作为键（Key），并以此来索引存储的值（Value），此时会出现一个错误：

Error: [string "return IncludeDir.yaml-cpp"]:1: attempt to perform arithmetic on a nil value (field 'yaml') in token: IncludeDir.yaml-cpp	（错误：[string "return IncludeDir.yaml-cpp"] : 1: 尝试对令牌中的零值（字段 "yaml"）执行算术运算：IncludeDir.yaml-cpp）
-------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------

编译器似乎将 '-' 识别为算术运算符，而不是文本符号，这导致他尝试进行算术运算操作。



但是当我将 '-' 更改为 '_' 时，这样的问题便消失了。



```

includedirs
{
    "src",
    "vendor/spdlog/include",
    "%{IncludeDir.GLFW}",
    "%{IncludeDir.Glad}",
    "%{IncludeDir.ImGui}",
    "%{IncludeDir.glm}",
    "%{IncludeDir.stb_image}",
    "%{IncludeDir.entt}",
    "%{IncludeDir}.yaml_cpp"
}

```

》》》关于最新的 YAML 导致链接错误的解决方案

编译器疑似在以动态库的方式尝试运行 yaml-cpp 库，并发出了很多警告

```

yaml-cpp\include\yaml-cpp\parser.h(65,28): warning C4251: "YAML::Parser::m_scanner": class "std::unique_ptr<YAML::Scanner, std::default_delete<YAML::Scanner>>" 需要有 dll 接口由 class "YAML::Parser" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\parser.h(66,31): warning C4251: "YAML::Parser::m_directives": class "std::unique_ptr<YAML::Directives, std::default_delete<YAML::Directives>>" 需要有 dll 接口由 class "YAML::Parser" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\parser.h(66,31): warning C4251: "YAML::Parser::m_directives": class "std::unique_ptr<YAML::Directives, std::default_delete<YAML::Directives>>" 需要有 dll 接口由 class "YAML::Parser" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\binary.h(65,30): warning C4251: "YAML::Binary::m_data": class "std::vector<unsigned char, std::allocator<unsigned char>>" 需要有 dll 接口由 class "YAML::Binary" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\binary.h(18): message : 参见 "std::vector<unsigned char, std::allocator<unsigned char>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\ostream_wrapper.h(48,29): warning C4251: "YAML::ostream_wrapper::m_buffer": class "std::vector<char, std::allocator<char>>" 需要有 dll 接口由 class "YAML::ostream_wrapper" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\ostream_wrapper.h(49): message : 参见 "std::vector<char, std::allocator<char>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\emitter.h(132,33): warning C4251: "YAML::Emitter::m_state": class "std::unique_ptr<YAML::EmitterState, std::default_delete<YAML::EmitterState>>" 需要有 dll 接口由 class "YAML::Emitter" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\emitter.h(132): message : 参见 "std::unique_ptr<YAML::EmitterState, std::default_delete<YAML::EmitterState>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\exceptions.h(155,58): warning C4275: 非 dll 接口 class "std::runtime_error" 用作 dll 接口 class "YAML::Exception" 的基 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
36) Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\stdexcept(101): message : 参见 "std::runtime_error" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\exceptions.h(155): message : 参见 "YAML::Exception" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\exceptions.h(164,15): warning C4251: "YAML::Exception::msg": class "std::basic_string<char, std::char_traits<char>, std::allocator<char>>" 需要有 dll 接口由 class "YAML::Exception" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
38) Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\wstring(4871): message : 参见 "std::basic_string<char, std::char_traits<char>, std::allocator<char>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\node.h(135,15): warning C4251: "YAML::Node::m_invalid_key": class "std::basic_string<char, std::char_traits<char>, std::allocator<char>>" 需要有 dll 接口由 class "YAML::Node" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
38) Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\wstring(4871): message : 参见 "std::basic_string<char, std::char_traits<char>, std::allocator<char>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\node.h(136,40): warning C4251: "YAML::Node::m_memory": class "std::shared_ptr<YAML::detail::memory_holder>" 需要有 dll 接口由 class "YAML::Node" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)

```

初步解决方案:

@mjthebest7294 2年前

I had to define "YAML_CPP_STATIC_DEFINE" in the premake for the newer version of YAML, otherwise it will try to compile as a DLL.

我必须在新版本 YAML 的预编译中定义 "YAML_CPP_STATIC_DEFINE", 否则它将尝试编译为 DLL

12 回复

6 条回复

@p3rk4n27 2年前

It now build yaml project but cant link it to engine... there are errors like unresolved external dllimport...

它现在构建 yaml 项目，但无法将其链接到引擎... 存在诸如未解析的外部 dllimport 之类的错误...

2 回复

@mjthebest7294 2年前

@p3rk4n27 maybe the engine compiles as a .dll instead of a static .lib

@p3rk4n27 也许引擎编译为 .dll 而不是静态 .lib

》》AND..

@rio9415 1年前

With the new version of yaml-cpp, you need to change staticruntime to "on" in premake file of yaml-cpp project

使用新版本的yaml-cpp，需要在yaml-cpp项目的premake文件中将staticruntime更改为"on"

首先我已经在 yaml-cpp 的 premake 文件中声明了 "YAML_CPP_STATIC_DEFINE"，并且打开了 staticruntime，但我发现没有作用。

```

defines
{
    "YAML_CPP_STATIC_DEFINE"
}

filter "system:windows"
{
    systemversion "latest"
    cppdialect "C++17"
    staticruntime "on"
}

```

接着解决:

问题是，你还需要在你所使用项目的 premake 文件中再次声明 "YAML_CPP_STATIC_DEFINE"

```
project "Nut-Editor"
objdir {"${wks.location}/bin-int/" .. outputdir .. "${prj.name}"}

files
{
    "src/**/*.h",
    "src/**/*.cpp"
}

defines
{
    "YAML_CPP_STATIC_DEFINE"
}

includedirs
{
    "${wks.location}/Nut/vendor/spdlog/include",
    "${wks.location}/Nut/src",
    "${wks.location}/Nut/vendor",
    "${IncludeDir.glm}",
    "${IncludeDir.entt}",
    "${IncludeDir.yaml_cpp}"
}
```

@kingofspades9720 2周前

If you are having issues using YAML, one fix might be adding `#define YAML_CPP_STATIC_DEFINE` inside of the Hazel premake file not just inside of the yaml-cpp premake file.

如果您在使用 YAML 时遇到问题，一种解决方法可能是在 Hazel 预编译文件内添加 `#define YAML_CPP_STATIC_DEFINE`，而不仅仅是在 yaml-cpp 预编译文件内。

👍 2 🗨 回复

@yu_a_v4427 11个月前

for new version of yaml-cpp just add a `#define YAML_CPP_STATIC_DEFINE` before including `<yaml-cpp/yaml.h>` in any file,


and turn on `staticruntime` in premakefile of yaml-cpp

对于新版本的 yaml-cpp，只需在任何文件中包含 `<yaml-cpp/yaml.h>` 之前添加 `#define YAML_CPP_STATIC_DEFINE`，

并在 yaml-cpp 的 premakefile 中打开 `staticruntime`

👍 6 🗨 回复

总结：



1分钟前

As of October 2024, you can correctly run yaml-cpp with the following requirements:

1. Define `YAML_CPP_STATIC_DEFINE` in the premake file of yaml-cpp, and define `YAML_CPP_STATIC_DEFINE` in the project configuration file that uses yaml-cpp (such as premake)
2. Define `staticruntime 'on'` in the yaml-cpp premake file

》》》什么是 .editorconfig 文件？有什么作用？

问题引入：在深入研究这次提交时，一个以 .editorconfig 署名的文件映入眼帘，这是什么文件？

文件介绍：

EditorConfig helps maintain consistent coding styles for multiple developers working on the same project across various editors and IDEs. The EditorConfig project consists of a file format for defining coding styles and a collection of text editor plugins that enable editors to read the file format and adhere to defined styles. EditorConfig files are easily readable and they work nicely with version control systems.

来自 <<https://editorconfig.org/>>

翻译：

EditorConfig 可帮助多个开发人员在不同的编辑器或 IDE 上维护同一个项目的编码风格，使其保持一致。EditorConfig 项目包含一个用于定义编码风格的文件格式和一组文本编辑器插件，这些插件可让编辑器读取文件格式并遵循定义的风格。EditorConfig 文件易于阅读，并且可与版本控制系统完美配合。

作用：

通过使用 EditorConfig 文件，团队中的每个成员可以确保他们的代码遵循相同的格式，降低因代码风格不一致而引起的问题。许多现代代码编辑器和 IDE（如 Visual Studio Code、Atom、JetBrains 系列等）都支持 EditorConfig，可以自动读取这些规则并应用到打开的文件中。

使用规范：

文件名：	文件名为 .editorconfig，通常放在项目根目录。
------	-------------------------------

键值对格式：	使用 <code>key = value</code> 的形式定义规则，每条规则占一行。 空行和以 <code>#</code> 开始的行会被视为注释。
范围选择器：	使用 <code>[*]</code> 表示应用于所有文件，也可以使用其他模式如 <code>*.js</code> 或 <code>*.py</code> 来指定特定文件类型。
支持的属性：（支持的键值对）	常用属性包括： <code>root</code> ：指示是否为顶层文件。 <code>end_of_line</code> ：指定行结束符（如 <code>lf</code> , <code>crlf</code> , <code>cr</code> ）。 <code>insert_final_newline</code> ：是否在文件末尾插入换行符。 <code>indent_style</code> ：设置缩进样式（如 <code>tab</code> 或 <code>space</code> ）。 <code>indent_size</code> ：指定缩进的大小，可以是数字或 <code>tab</code> 。 <code>charset</code> ：文件字符集（如 <code>utf-8</code> , <code>latin1</code> 等）。 <code>trim_trailing_whitespace</code> ：是否修剪行尾空白。

详情参考文档：(<https://spec.editorconfig.org/>)

Table of Contents

- [EditorConfig Specification](#)
 - [Introduction \(informative\)](#)
 - [Terminology](#)
 - [File Format](#)
 - [No inline comments](#)
 - [Parts of an EditorConfig file](#)
 - [Glob Expressions](#)
 - [File Processing](#)
 - [Supported Pairs](#)

代码理解：

```
...  ...  @@ -0,0 +1,8 @@

1  + # top-most EditorConfig file
2  + root = true
3  +
4  + # Unix-style newlines with a newline ending every file
5  + [*]
6  + end_of_line = lf
7  + insert_final_newline = true
8  + indent_style = tab
```

root = true:	指示这是一个顶层的 EditorConfig 文件，编辑器在找到此文件后不会再向上查找其他 EditorConfig 文件。
[*]:	表示应用于所有文件类型的规则。
end_of_line = lf:	指定行结束符为 Unix 风格的换行符（LF，Line Feed）。这通常在类 Unix 系统（如 Linux 和 macOS）中使用。
insert_final_newline = true:	指定在每个文件的末尾插入一个换行符。这是一种良好的编码习惯，许多项目标准要求这样做。
indent_style = tab:	指定缩进样式为制表符（tab），而不是空格。这会影响代码的缩进方式。

《《《拓展：什么是 Hard tabs? 什么是 Soft tabs?

Hard Tabs	是使用制表符进行缩进，具有灵活性但可能导致跨环境的不一致。
Soft Tabs	是使用空格进行缩进，保证了一致性但文件体积可能更大。

选择使用哪种方式通常取决于团队的编码标准或个人偏好。

》》》Y A M L U know what I'm saying

》》》YAML YAML YAML

》》》关于这次 premake 构架的维护，我只上传了一部分，剩下的留到之后维护时再做。现在我去了解一下 YAML。

》》》YAML, What is yaml ? What we can do by yaml ?

介绍：

YAML is a human-readable data serialization language that is often used for writing configuration files. Depending on whom you ask, YAML stands for yet another markup language or YAML ain't markup language (a recursive acronym), which emphasizes that YAML is for data, not documents. 来自 < https://www.redhat.com/en/topics/automation/what-is-yaml >	YAML 是一种人类可读的数据序列化语言，通常用于编写配置文件。根据使用的对象，YAML 可以代表另一种标记语言或者说 YAML 根本不是标记语言（递归缩写），这强调了 YAML 用于数据，而不是文档。
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------

理解：

在程序中，我们可以使用 yaml 对文件进行两种操作：序列化和反序列化（Serialize & Deserialize）。
序列化意味着我们可以将复杂的数据转变为字节流，进而可以将其轻易保存到文件或数据库中。
反序列化则意味着我们可以对已经序列化的数据进行逆处理，进而将数据转换回原始的数据结构或对象状态。

基础：

基本结构

映射（Map）：键值对的集合。	key: value
序列（Sequence）：有序的元素列表。	- item1 - item2

	- item3
--	---------

2. 嵌套结构

YAML 支持嵌套映射和序列，可以组合使用：	person: name: John Doe age: 30 hobbies: - reading - cycling
------------------------	----------------------------------------------------------------------------

3. 数据类型

YAML 支持多种数据类型， 包括：字符串，数字，布尔值，Null 值。	例如： string: "Hello, World!" number: 42 boolean: true null_value: null
-----------------------------------------	-----------------------------------------------------------------------------------

》》》yaml-cpp 的使用（详情请阅览：<https://github.com/ibeder/yaml-cpp/blob/master/docs/Tutorial.md>）

在 C++ 中使用 yaml-cpp 库，可以方便地处理 YAML 数据的读取和写入。（以下是读取 Yaml 文件和写入 Yaml 文件的示例）

读取 YAML	<pre>#include <iostream> #include <yaml-cpp/yaml.h> int main() { YAML::Node config = YAML::LoadFile("config.yaml"); std::string name = config["person"]["name"].as<std::string>(); std::cout << "Name: " << name << std::endl; return 0; }</pre>
写入 YAML： 使用 YAML::Emitter 可以生成 YAML 文件	<pre>#include <iostream> #include <yaml-cpp/yaml.h> int main() { YAML::Emitter out; out << YAML::BeginMap; out << YAML::Key << "name" << YAML::Value << "John Doe"; out << YAML::Key << "age" << YAML::Value << 30; out << YAML::EndMap; std::cout << out.str() << std::endl; // 输出生成的 YAML return 0; }</pre>

YAML::Node

定义：YAML::Node 是 YAML-CPP 中的一个核心类，表示 YAML 文档中的一个节点。一个节点可以是标量（单个值）、序列（列表）或映射（键值对）。通过 YAML::Node，你可以以编程方式访问和操作 YAML 数据结构。	创建和使用 YAML::Node： Eg. <pre>#include <yaml-cpp/yaml.h> YAML::Node node = YAML::Load("key: value"); std::string value = node["key"].as<std::string>();</pre>
------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Sequences 和 Maps

Sequences （序列）是一个有序列表，表示一组无命名的值。它们在 YAML 中用短横线表示：	fruits: - Apple - Banana - Cherry
在 YAML-CPP 中，你可以这样处理序列：	Eg. <pre>YAML::Node sequence = YAML::Load("[Apple, Banana, Cherry]"); for (const auto& item : sequence) { std::cout << item.as<std::string>() << std::endl; }</pre>
Maps （映射）是一组键值对，表示命名的值。它们在 YAML 中用冒号分隔表示：	person: name: John Doe age: 30
在 YAML-CPP 中，你可以这样处理映射：	Eg. <pre>YAML::Node map = YAML::Load("name: John Doe\nage: 30"); std::string name = map["name"].as<std::string>(); int age = map["age"].as<int>();</pre>

Sequences 和 Maps 的不同之处

序列和映射都是 YAML::Node 的一种。你可以在一个映射中嵌套序列，反之亦然。

不同之处：

序列：	没有键，每个项都有顺序。
映射：	每个项都有唯一的键，顺序不重要。

Converting To/From Native Data Types

YAML-CPP 提供了方便的方法来将 YAML::Node 转换为 C++ 的原生数据类型。你可以使用 as<T>() 方法进行转换。

示例：从 YAML::Node 转换到原生数据类型	<pre>YAML::Node node = YAML::Load("name: John Doe\nage: 30"); std::string name = node["name"].as<std::string>(); int age = node["age"].as<int>();</pre>
示例：从原生数据类型转换到 YAML::Node	<pre>YAML::Node newNode; newNode["name"] = "Jane Doe"; newNode["age"] = 28; // 序列 YAML::Node fruits; fruits.push_back("Apple"); fruits.push_back("Banana"); newNode["fruits"] = fruits; // 输出为 YAML 格式 std::cout << newNode << std::endl;</pre>

》》由此引出两个疑惑:

问题一：

查阅文档时，我发现当插入的索引超出当前序列的范围时，YAML-CPP 会将节点视为映射，而不是继续保持序列

Indexing a sequence node by an index that's not in its range will *usually* turn it into a map, but if the index is one past the end of the sequence, then the sequence will grow by one to accommodate it. (That's the only exception to this rule.) For example,

添加新元素:

node[3] = 4; 试图在索引 3 的位置插入 4。由于索引 3 在当前序列中不存在，所以 node 仍然被认为是序列，结果是 [1, 2, 3, 4]。

```
YAML::Node node = YAML::Load("[1, 2, 3]");
node[3] = 4; // still a sequence, [1, 2, 3, 4]
node[10] = 10; // now it's a map! {0: 1, 1: 2, 2: 3, 3: 4, 10: 10}
```

插入非连续索引:

node[10] = 10; 尝试将值 10 插入到索引 10 的位置，因为 10 远大于当前序列的最大索引 (3)，这导致 node 变成了一个映射。最终结果是 {0: 1, 1: 2, 2: 3, 3: 4, 10: 10}。

结论：动态类型：YAML::Node 的类型是动态的，可以在运行时根据操作的不同而变化。当你使用整数索引时，它保持序列。当你使用非连续的索引或字符串键时，它会转变为映射。

问题二：如何为Node添加一个映射？

在 YAML::Node node = YAML::Load("[1, 2, 3]"); 的情况下，使用 node[1] = 5 是不合适的。

如果你想让 node[1] 表示一个映射，node[1] = 5 会将序列中索引为 1 的元素（即第二个元素）设置为整数 5，而不是将其更改为一个映射。

如果你想在当前位置设置一个映射，你可以这样做：	<pre>YAML::Node node = YAML::Load("[1, 2, 3]"); node[1] = YAML::Node(YAML::NodeType::Map); // 创建一个新的映射 node[1]["key"] = "value"; // 向映射中添加键值对</pre>
结构：	<pre>- 1 - key: value - 3</pre>
或者：	<pre>YAML::Node node = YAML::Load("[1, 2, 3]"); // 将 node[1] 设置为一个新的映射 node[1] = YAML::Node(YAML::NodeType::Map); // 设置键为原来的值 2，并赋值为 5 node[1][2] = 5; // 这里的 2 是之前 index 1 的值</pre>
结构：	<pre>- 1 - 2: 5 - 3</pre>

注意：

如果你使用了 node["1"] = 5，由于 "1" 是一个字符串键，而不是数字索引，这将使程序尝试在 node 中以 "1" 为键插入值 5。node 原本是一个序列，但它会因此转变为一个映射。	最终结果会是 { 0: 1, 1: 2, 2: 3, "1": 5}, 其中 "1" 是一个新的字符串键。
-----------------------------------------------------------------------------------------------------	-------------------------------------------------------------------

》》》ifstream 和 ofstream 之间的关系

二者定义在 <fstream> 头文件中，管理文件流。

std::ifstream:	用于从文件中读取数据（输入文件流）。
Std::ofstream:	用于向文件中写入数据（输出文件流）。

易混淆: <iostream>和文件流没有关系，<iostream> 是提供输入或输出流的标准库，主要包括 std::cin, std::cout, std::cerr 等。

》》ofstream 的使用: std::ofstream 用于创建和写入文件

```
// Send file-stream
std::ofstream fout(filepath); // Create and open a file from the filepath
fout << out.c_str();          // Writing data
```

》》ifstream 的使用: std::ifstream 用于读入文件，进而对读入的文件进行一些处理。

(图例: 逐行读取文件内容到字符串中)

```
std::ifstream file(filepath);
std::string line;
// 逐行读取文件内容
while (std::getline(file, line)) {
    std::cout << line << std::endl;
}
```

或者 (比上述方法更加高效, 迅捷)

```
std::ifstream file(filepath);
std::stringstream fileContent;
fileContent << file.rdbuf();
```

《《《什么是 rdbuf();

在 C++ 中, rd 通常是 "read" 的简写, 意味着与读取操作相关的函数。

rdbuf()

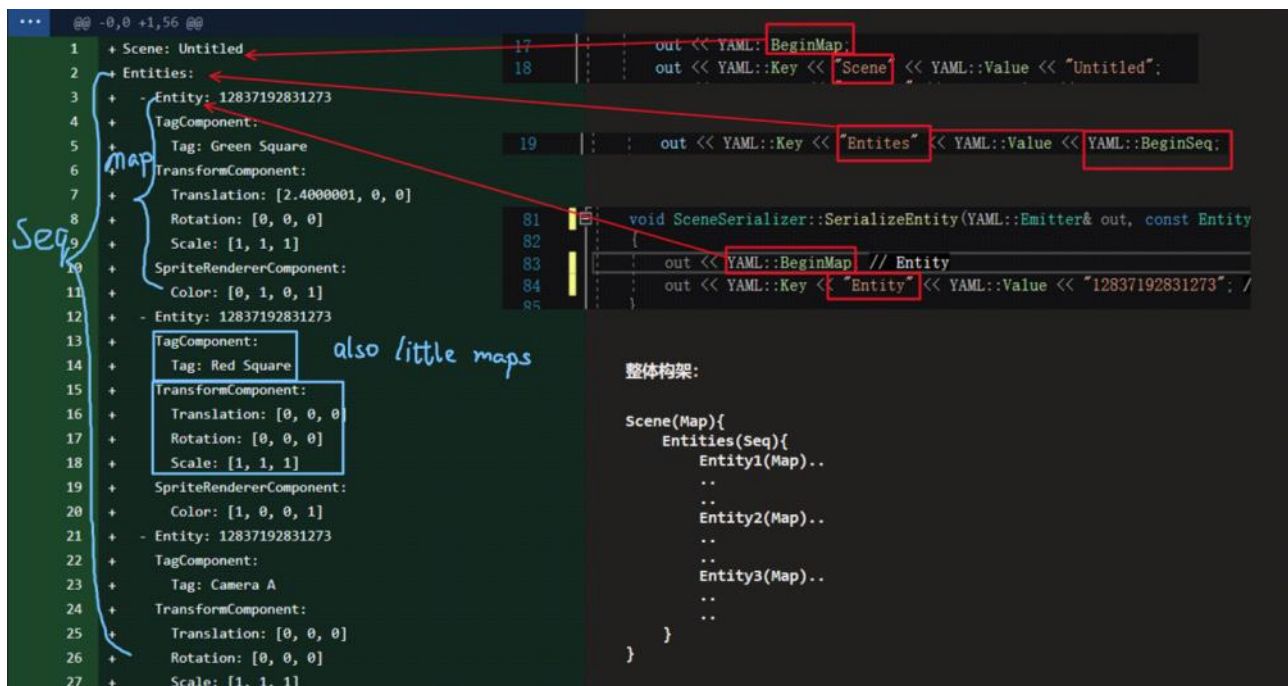
释义:	rdbuf() 是 C++ 中的一个成员函数, 可以直接访问流的底层缓冲区。它通常用于与输入输出流 (如 std::ifstream, std::ofstream, std::iostream 等) 交互。
返回类型:	std::streambuf* 返回指向与流关联的 std::streambuf 对象的指针。该指针可以用于直接进行低级别的输入输出操作。
优点: 直接访问缓冲区	rdbuf() 返回一个指向当前流缓冲区的指针 (即 std::streambuf 对象), 允许你直接从流中读取或写入数据。 这意味着, 你可以将整个文件的内容一次性读入, 而不需要逐行或逐字符地读取, 从而提高了效率。 当处理大型文件时, 逐行读取会涉及多次 I/O 操作, 这可能导致性能瓶颈。而使用 rdbuf() 可以减少这些 I/O 操作, 因为它一次性读取整个缓冲区的数据。

类似的 rd 开头的函数还有 rdstate()	含义: rdstate() 是一个成员函数, 用于获取流的状态标志。它返回一个整数, 表示流的当前状态, 包括是否已达到文件结束、是否发生了错误等。
--------------------------	----------------------------------------------------------------------------

》》》FIEL STRUCTURE U know what I'm saying

》》》YAML YAML YAML

》》》YAML 文件构架, YAML 文件设置思路



因此我们也可解释 Deserialize() 函数中做出的操作: 从 data(map) 中取出序列 entities(seq), 然后通过 For 循环对序列中的 entity(map) 进行读取, 随后根据读取的数据去复现场景。
 需要提醒的是: Map 中的元素不能重复, Seq 中的元素可以重复。

```
// According to data, we reproduce the scene
auto entities = data["Entity"];
if(entities)
{
    for (auto entity : entities)
    {
        // reproduce codes
    }
}

return true;
```

》》》》 YAML::Emitter out << YAML::Flow;

概念: out << YAML::Flow 是 C++ 中使用 YAML 库 (如 yaml-cpp) 时的一种语法, 它用来设置 YAML 输出的格式为“流式” (flow style)。

详解: YAML::Flow 是一个常量, 指示输出的 YAML 数据应采用流式表示形式。

流式表示形式将集合 (如数组和映射) 以更紧凑的方式表示, 例如使用方括号 [] 表示数组, 使用花括号 {} 表示映射。

使用场景:

当你想要以更紧凑的格式输出数据时, 可以使用流式表示形式。相比于块状表示 (block style), 流式表示在视觉上更简洁, 适用于小型数据结构或在单行内表示数据。

假设你有一个简单的 YAML 数据结构, 如果不使用 YAML::Flow, 输出可能是这样的 (块状表示):	items: - item1 - item2
而如果使用 YAML::Flow, 此时, 输出将是:	items: [item1, item2]

示例代码:

```
#include <yaml-cpp/yaml.h>
#include <iostream>
int main() {
    YAML::Emitter out;

    out << YAML::Flow; // 设置为流式格式
    out << YAML::BeginMap
    << YAML::Key << "items" << YAML::Value
    << YAML::BeginSeq
    << "item1" << "item2"
    << YAML::EndSeq
    << YAML::EndMap;
    std::cout << out.str() << std::endl;
}
```

》》》》 设计上的理解

1.Serialize


```

if (entity.HasComponent<CameraComponent>())
{
    out << YAML::Key << "CameraComponent";
    out << YAML::BeginMap;

    auto& cc = entity.GetComponent<CameraComponent>();
    auto& camera = cc.Camera;

    out << YAML::Key << "Camera" << YAML::Value;
    out << YAML::BeginMap;{
        out << YAML::Key << "ProjectionType" << YAML::Value << (int)camera.GetProjectionType();
        out << YAML::Key << "PerspectiveFOV" << YAML::Value << camera.GetPerspectiveVerticalFOV();
        out << YAML::Key << "PerspectiveNear" << YAML::Value << camera.GetPerspectiveNearClip();
        out << YAML::Key << "PerspectiveFar" << YAML::Value << camera.GetPerspectiveFarClip();
        out << YAML::Key << "OrthographicSize" << YAML::Value << camera.GetOrthographicSize();
        out << YAML::Key << "OrthographicNear" << YAML::Value << camera.GetOrthographicNearClip();
        out << YAML::Key << "OrthographicFar" << YAML::Value << camera.GetOrthographicFarClip();
    }
    out << YAML::EndMap;

    out << YAML::Key << "Primary" << YAML::Value << cc.Primary;
    out << YAML::Key << "Fixed Aspect Ratio" << YAML::Value << cc.FixedAspectRatio;

    out << YAML::EndMap;
}

```

2.Yaml data

```

Scene: Untitled
Entities:
- Entity: 256257383941
  TagComponent:
    Tag: Clip-Camera
  TransformComponent:
    Translation: [0, 0, 0]
    Rotation: [0, 0, 0]
    Scale: [1, 1, 1]
  CameraComponent:
    Camera:
      ProjectionType: 1
      PerspectiveFOV: 0.785398185
      PerspectiveNear: 0.00999999978
      PerspectiveFar: 1000
      OrthographicSize: 5
      OrthographicNear: -1
      OrthographicFar: 1
    Primary: false
    Fixed Aspect Ratio: false

```

3.Deserialize

```

auto cameraComponent = data["CameraComponent"];
if(cameraComponent)
{
    auto& cameraProps = cameraComponent["Camera"];
    auto& cc = entity.AddComponent<CameraComponent>();

    int projectionType = cameraProps["ProjectionType"].as<int>();
    cc.Camera.SetProjectionType((SceneCamera::ProjectionType)projectionType);
    cc.Camera.SetPerspectiveVerticalFOV(cameraProps["PerspectiveFOV"].as<float>());
    cc.Camera.SetPerspectiveNearClip(cameraProps["PerspectiveNear"].as<float>());
    cc.Camera.SetPerspectiveFarClip(cameraProps["PerspectiveFar"].as<float>());
    cc.Camera.SetOrthographicSize(cameraProps["OrthographicSize"].as<float>());
    cc.Camera.SetOrthographicNearClip(cameraProps["OrthographicNear"].as<float>());
    cc.Camera.SetOrthographicFarClip(cameraProps["OrthographicFar"].as<float>());
    // Unlike Camera, Primary is a separate key-value mapping,
    // while Camera is a map that requires further access.
    cc.Primary = cameraComponent["Primary"].as<bool>();
    cc.FixedAspectRatio = cameraComponent["Fixed Aspect Ratio"].as<bool>();
}

```

》》 There are few issues you need to know:

- 1.字符匹配: 查找value所用的key需要正确无误, 比如你想查找yaml文件中的 Fixed Aspect Ratio, 你就必须用 Fixed Aspect Ratio 作为索引, 而不是 FixedAspectRatio.
- 2.Map访问: 如果你在map中查找其中存储的元素, 你只能访问顶层的元素, 而不能访问靠底层的元素。比如 CameraComponent

```

CameraComponent: ← map
  Camera: ← element 1 (also a map)
    ProjectionType: 1
    PerspectiveFOV: 0.785398185
    PerspectiveNear: 0.00999999978
    PerspectiveFar: 1000
    OrthographicSize: 5
    OrthographicNear: -1
    OrthographicFar: 1
  Primary: false ← element 2 (key-value)

```

图例此时处于 cameraComponent, 你通过这两个 key 访问其中的 value: Camera 和 Primary (比如调用 cameraComponent["Camera"])。可是如果你想通过 CameraComponent 访问

projectionType, 就不能使用 cameraComponent["ProjectionType"] 这样的语句, 而必须使用 cameraComponent["Camera"]["ProjectionType"], 否则会报错。

》》》Cherno 将文件保存在 .hazel 后缀的文件中, 这可以吗? 为什么? 只有Hazel 才能处理这种文件吗? ?

```
+         if (ImGui::MenuItem("Serialize"))
+         {
+             SceneSerializer serializer(m_ActiveScene);
+
+             serializer.Serialize("assets/scenes/Example.hazel");
+         }
+
+         if (ImGui::MenuItem("Deserialize"))
+         {
+             SceneSerializer serializer(m_ActiveScene);
+
+             serializer.Deserialize("assets/scenes/Example.hazel");
+         }
```

这取决于你的打开方式, 现在 Hazel (或者 Nut) 有能力接受这种文件, 通过我们定义的函数, Hazel (或者 Nut) 通过文件流合理读入文件, 然后识别文件内容并且做出了相应操作。如果你使用 word 打开这种文件, 应用程序就会通过文本格式打开这个文件, 这样也是被允许的, 因为我们就是以文本的形式去设置了 yaml 文件。不过使用其他打开方式可能就会出错。

》》》注意事项/可改进事项

@KarimInordinata 3年前

Is there any reason you've implemented this without reflection? Or is it just simplicity? For my engine I've added simple metaprogramming to allow for reflection, which means I don't have to write deserializers for every member variable, or write code to show it in the editor.

您是否有任何理由不加反思就实施了这一点? 或者只是简单? 对于我的引擎, 我添加了简单的元编程以允许反射, 这意味着我不必为每个成员变量编写反序列化器。或编写代码以在编辑器中显示它。

25 7 条回复

@qx-jd9mh 3年前

@mattmurphy7030 "game devs" are allergic to template metaprogramming

@mattmurphy7030 "game devs"对模板元编程过敏

2 2 回复

@ipatrick6686 3年前

how?

如何?

1 1 回复

@erniegutierrez410 3年前

He doesn't have a clue

他不知道

1 1 回复

@johnjackson9767 3年前

Yes. Reflection information makes this a breeze.

是的。反射信息使这变得轻而易举。

1 1 回复

@utkarshja9547 3年前

Is there a place where I can go to learn about this?!

有没有地方可以去学习这方面的知识? !

1 1 回复

@utkarshja9547 3年前

@johnjackson9767 Is there a place I can learn about this???? I'm fairly annoyed by having to type out the statements for each member variable....

@johnjackson9767有地方可以了解这个吗? ? 我对必须为每个成员变量键入语句感到相当恼火.....

1 1 回复

@NikiGity 2年前 (修改过)

I have the same question. This seems like needless error prone work - I'm sure at least 10 times a year you're going to add a member to a class and forget to add it to the serializer, or you add it to the serializer but forget it in the deserializer. Why do it this way when you can just enumerate over the reflected fields in the class and just use their variable/member name as the key? Then you only write code once per data type, not once per member.

我有同样的问题。这似乎是不必要的容易出错的工作——我确信每年至少有 10 次你要向类中添加成员但忘记将其添加到序列化器中, 或者你将其添加到序列化器中但忘记了解串器。当您以枚举类中的反射字段并使用它们的变量/成员名称作为键时, 为什么要这样做呢? 然后, 您只需为每个数据类型编写一次代码, 而不是为每个成员编写一次代码。

AND

@wakeupthesun3 1年前 (修改过)

Another thing to note (I'm not sure if this is covered later) is the scene is being deserialized in inverse order in which the original entities were added to the scene. You can see this by the original scene has the red square on top, covering the green square. When deserialized, the green square is on top. You can either serialize your entities backwards, or deserialize them backwards. I think deserializing backwards is better, because then the serialized file will match the order of your hierarchy panel. To deserialize backwards, you can make a vector of the entity nodes and then get a reverse iterator to that vector:

auto entitiesNode = data["Entities"];

// reverse it to add the entities in the order they were
// originally added
std::vector<YAML::Node> entitiesRev(entitiesNode.begin(),
 entitiesNode.end());

for (auto it = entitiesRev.rbegin(); it != entitiesRev.rend(); ++it)
{
 s_deserializeEntity(*it, mp_scene.get());
}

Have fun!

另一件需要注意的事情（我不确定稍后是否会介绍这一点）是场景正在以与原始实体添加到场景相反的顺序进行反序列化。您可以通过原始场景看到顶部有红色方块，覆盖了绿色方块。反序列化时，绿色方块位于顶部。您可以向后序列化实体，也可以向后反序列化它们。我认为向后反序列化更好，因为这样序列化的文件将与层次结构面板的顺序匹配。要向后反序列化，您可以创建实体节点的向量，然后获取该向量的反向迭代器：

自动实体节点=数据["实体"];

// 反转它以按实体的顺序添加实体
// 最初添加的
std::vector<YAML::Node>EntityRev(entitiesNode.begin(),
 实体节点.end());

for (auto it =EntityRev.rbegin();it!=entitiesRev.rend();++it)
{
 s_deserializeEntity(*it, mp_scene.get());
}

玩得开心！

----- Save/Open file dialog -----

》》》什么是 commdlg 库？

```
1 #include <winapifamily.h>
2
3
4 *
5 * commdlg.h -- This module defines the 32-Bit Common Dialog APIs
6 *
7 * Copyright (c) Microsoft Corporation. All rights reserved.
8 *
9 *****/
10
```

概念： commdlg.h 是 Windows API 中的一个头文件，它用于实现标准对话框功能，如文件打开和文件保存对话框。这个头文件定义了与这些对话框相关的结构、常量和函数，使开发者能够方便地在应用程序中集成文件选择功能。

在使用 commdlg.h 时，通常会涉及到以下几个函数：

GetOpenFileName：	用于显示“打开文件”对话框。
GetSaveFileName：	用于显示“保存文件”对话框。

》》》 glfw3.h 和 glfw3native.h 这两个库之间的不同

不同： glfw3.h 和 glfw3native.h 之间的区别在于： glfw3.h 用于创建 glfw 类型的窗口， glfw3native.h 用于获取原生操作系统中的窗口的句柄，以此来进行更底层的操作。

1. <GLFW/glfw3.h> 目的： 这是 GLFW 的主要头文件，提供了创建窗口、处理输入、管理 OpenGL 上下文、以及其他与窗口和输入相关的功能。 内容： 包含了 GLFW 的所有核心功能，比如：创建和管理窗口和上下文、处理键盘、鼠标等输入事件、管理 OpenGL 扩展、定时器等功能	2. <GLFW/glfw3native.h> 目的： 这个头文件提供了平台特定的功能，通常用于访问底层原生窗口句柄或其他系统级别的功能。 内容： 包含了一些函数，这些函数允许你获取与操作系统相关的窗口句柄。例如，在 Windows 系统上，你可以通过它获得 HWND 句柄；在 X11 上，你可以获得相应的 X11 窗口 ID。 使用场景： 如果你需要与操作系统的原生 API 进行交互（例如，在窗口中嵌入第三方控件，或与其他库集成），则可能需要使用这个头文件。
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

》》》关于 glfw3native.h 和 #define GLFW_EXPOSE_NATIVE_WIN32

```

161  B/*****
162      * Functions
163      *****/
164
165  B#if defined(GLFW_EXPOSE_NATIVE_WIN32) 非活动预处理器块
166      #endif
167
168
169
170
171
172  /*! @brief Returns the `HWND` of the specified window.
173   *
174   * @return The `HWND` of the specified window, or `NULL` if an
175   * [error](#error_handling) occurred.
176   *
177   * @errors Possible errors include @ref GLFW_NOT_INITIALIZED and @ref
178   * GLFW_PLATFORM_UNAVAILABLE.
179   *
180   * @remark The `HDC` associated with the window can be queried with the
181   * [GetDC](https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getdc)
182   * function.
183   * @code
184   * HDC dc = GetDC glfwGetWin32Window(window);
185   * @endcode
186   * This DC is private and does not need to be released.
187   *
188   * @thread_safety This function may be called from any thread. Access is not
189   * synchronized.
190   *
191   * @since Added in version 3.0.
192   *
193   * @ingroup native
194   */
195  GLFWAPI HWND glfwGetWin32Window(GLFWwindow* window);
196  #endif
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226

```

```
std::string FileDialogs::openFile(const char* filter)
{
    OPENFILENAMEA ofn;
    CHAR szFile[260] = { 0 };
    ZeroMemory(&ofn, sizeof(OPENFILENAME));
    ofn.lStructSize = sizeof(OPENFILENAME);
    ofn.hwndOwner = glfwGetWin32Window((GLFWwindow*)Application::Get().GetWindow().GetNativeWindow());
    ofn.lpstrFile = szFile;
    ofn.nMaxFile = sizeof(szFile);
    ofn.lpstrFilter = filter;
    ofn.nFilterIndex = 1;
    ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST | OFN_NOCHANGEDIR;
    if (GetOpenFileNameA(&ofn) == TRUE)
    {
        return ofn.lpstrFile;
    }
    return std::string();
}
```

<code>std::string FileDialogs::OpenFile(const char* filter)</code>	定义一个名为 <code>OpenFile</code> 的静态成员函数，接受一个字符串参数 <code>filter</code> ，该参数用于指定文件类型过滤器（例如仅显示文本文件或图像文件 / 或者 <code>Cherno</code> 填入的：" <code>Hazel Scene (*.hazel)\0*.hazel\0</code> "）
<code>{</code>	创建一个 <code>OPENFILENAMEA</code> 结构体实例 <code>ofn</code> ，该结构体用于存储文件对话框的各种信息。
<code>OPENFILENAMEA ofn;</code>	
<code>CHAR szFile[260] = { 0 };</code>	声明一个字符数组 <code>szFile</code> ，用于存储用户选择的文件路径，大小为 260 字节（这是 Windows 系统中路径的最大长度限制）。
<code>ZeroMemory(&ofn, sizeof(OPENFILENAME));</code>	将 <code>ofn</code> 结构体的内存清零，以确保它的所有字段都被初始化为零。
<code>ofn.lStructSize = sizeof(OPENFILENAME);</code>	设置 <code>ofn</code> 结构体的大小，以便 Windows 知道使用哪个版本的结构体
<code>ofn.hwndOwner = glfwGetWin32Window((GLFWwindow*) Application::Get().GetWindow().GetNativeWindow());</code>	获取当前窗口的句柄并赋值给 <code>ofn.hwndOwner</code> ，这样文件对话框会相对于这个窗口显示。
<code>ofn.lpstrFile = szFile;</code>	将 <code>szFile</code> 的地址赋值给 <code>ofn.lpstrFile</code> ，以便在用户选择文件后可以将文件路径写入这个数组中。
<code>ofn.nMaxFile = sizeof(szFile);</code>	设置 <code>ofn.nMaxFile</code> 为 <code>szFile</code> 的大小，以告诉对话框可以存储的最大路径长度。
<code>ofn.lpstrFilter = filter;</code>	设置 <code>ofn.lpstrFilter</code> 为传入的 <code>filter</code> 参数，以定义可见的文件类型（例如 <code>".txt;.cpp"</code> ）。
<code>ofn.nFilterIndex = 1;</code>	设定过滤器的索引，通常设为 1 表示使用第一个过滤器。
<code>ofn.Flags = OFN_PATHMUSTEXIST OFN_FILEMUSTEXIST OFN_NOCHANGEDIR;</code>	设置对话框的标志： <ul style="list-style-type: none"><code>OFN_PATHMUSTEXIST</code>：用户选择的路径必须存在。<code>OFN_FILEMUSTEXIST</code>：用户选择的文件必须存在。

	• OFN_NOCHANGEDIR: 打开对话框时, 不更改当前工作目录。
if (GetOpenFileNameA(&ofn) == TRUE)	调用 Windows API 函数 GetOpenFileNameA 显示文件对话框。如果用户选择了一个文件, 返回值为 TRUE。
{ return ofn.lpstrFile; }	如果文件选择成功, 返回 ofn.lpstrFile 中存储的文件路径。
return std::string(); }	如果用户取消了操作或发生错误, 返回一个空的字符串。

《《关于 OPENFILENAMEA 的定义

```
typedef struct tagOFNA {
    DWORD      lStructSize;
    HWND        hwndOwner;
    HINSTANCE    hInstance;
    LPCSTR      lpstrFilter;
    LPSTR       lpstrCustomFilter;
    DWORD       nMaxCustFilter;
    DWORD       nFilterIndex;
    LPSTR       lpstrFile;
    DWORD       nMaxFile;
    LPSTR       lpstrFileName;
    DWORD       nMaxFileName;
    LPCSTR      lpstrInitialDir;
    LPCSTR      lpstrTitle;
    DWORD       Flags;
    WORD        nFileOffset;
    WORD        nFileExtension;
    LPCSTR      lpstrDefExt;
    LPARAM      lCustData;
    LPOFNHOOKPROC lpfnHook;
    LPCSTR      lpTemplateName;
#ifdef _MAC
    LPEDITMENU  lpEditInfo;
    LPCSTR      lpstrPrompt;
#endif
#ifdef (_WIN32_WINNT >= 0x0500)
    void *      pvReserved;
    DWORD       dwReserved;
    DWORD       FlagsEx;
#endif // (_WIN32_WINNT >= 0x0500)
} OPENFILENAMEA, *LPOPENFILENAMEA;
```

《《关于 ZeroMemory 函数的定义

在 minwinbase.h 函数中:

```
39  |#define ZeroMemory RtlZeroMemory
20266 |#define RtlZeroMemory(Destination,Length) memset((Destination),0,(Length))
```

《《ofn.lpstrFilter = filter; 之中, filter 为什么是 const char* ? 填入的时候有什么规范?

lpstrFilter 格式

示例:

```
const char *filter = "Hazel Files (*.hazel)\0*.hazel\0All Files (*.*)\0*.*\0\0";
```

格式:

每一组文件类型描述由两部分组成 -> 描述字符串和扩展名字符串:

描述字符串是用户在对话框中看到的文件类型名称, 扩展名字符串指定了可以被选择的文件扩展名。各组之间用 \0 分隔, 最后以两个 \0 结束。

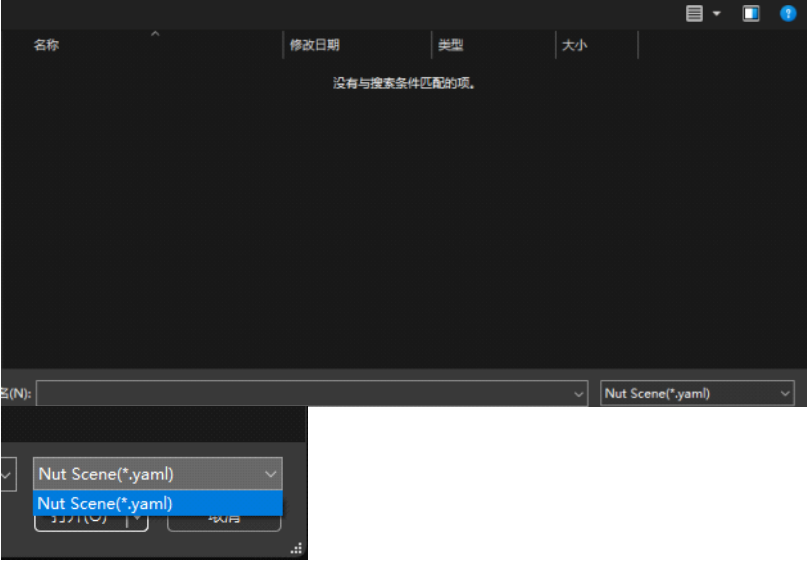
对话框如何识别和表示:

Hazel Files (*.hazel)\0*.hazel\0 -> 在文件对话框中, 用户会看到 "Hazel Files (*.hazel)" 作为文件类型的选项。当选择这个选项后, 对话框会过滤出所有以 .hazel 结尾的文件。

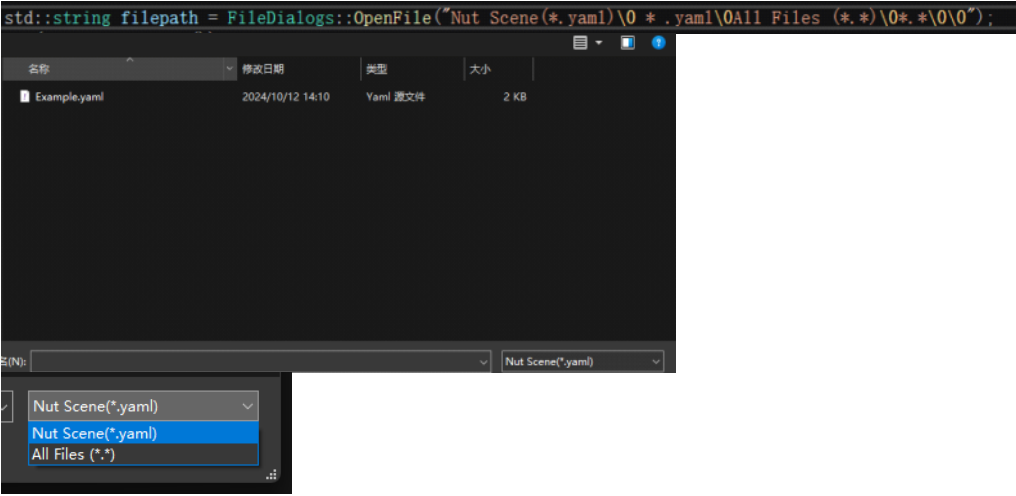
All Files (*.*)\0*.*\0 -> 如果用户选择 "所有文件 (.)", 则会显示所有文件, 包括 .hazel 文件。

例如:

```
std::string filepath = FileDialogs::OpenFile("Nut Scene(*.yaml)\0*.*\0*.hazel\0");
```

Nut Scene(*.yaml) 为显示的文本提示，通常设置为你可以选择的过滤器，通过文本提示，代码会索引到合适的过滤器。hazel 则会根据你选择的文本索引到你设置的过滤器，然后对所有文件进行过滤，如果符合 .hazel 后缀，便显示在对话框窗口中。



《《 GetOpenFileNameA 函数的作用：

GetOpenFileNameA 是 Windows API 中的一个函数，用于显示一个标准的“打开文件”对话框，让用户选择一个文件。

函数原型	BOOL GetOpenFileNameA(LPOPENFILENAMEA lpofn);
参数	lpofn: 指向 OPENFILENAMEA 结构体的指针，该结构体包含了对对话框的配置信息和用户选择的文件路径。
返回值	如果用户成功选择了一个文件并点击“确定”，函数返回非零值（通常是 TRUE）。 如果用户取消对话框或发生错误，返回值为零（FALSE）。可以通过调用 GetLastError 来获取更多错误信息。

《《 Flags 详细解释：

在 OPENFILENAME 结构体中，可以设置多个标志（Flags）来控制对话框的行为。

OFN_PATHMUSTEXIST:	含义：用户输入的路径必须存在。如果用户在对话框中输入了一个路径（而不是从浏览器中选择），这个路径必须是有效的。 触发情况：当用户输入一个不存在的路径并尝试打开文件时，会出现错误提示，说明路径无效。
OFN_FILEMUSTEXIST:	含义：用户选择的文件必须存在。即使用户在对话框中选择了文件，如果该文件不存在，就会阻止选择。 触发情况：当用户选择的文件实际上在磁盘上不存在时，系统会显示错误提示，说明所选文件不存在。
OFN_NOCHANGEDIR:	含义：打开对话框时不改变当前工作目录。默认情况下，打开文件对话框可能会改变程序的当前工作目录以便于访问文件。 触发情况：这个标志的作用是确保选择文件后，程序的当前工作目录保持不变，即使用户选择了不同的文件路径。

《关于 OFN_NOCHANGEDIR 的详细说明

背景：在 Windows 应用程序中，当前工作目录是指程序在文件系统中默认访问的位置。当应用程序启动时，它会有一个初始的工作目录，通常是可执行文件所在的位置。

使用场景：

当用户打开文件对话框并选择一个文件时，默认情况下，Windows 会将当前工作目录更改为用户选择的文件所在的路径。这意味着如果用户选择了一个不同位置的文件，之后程序的所有文件访问操作都会基于这个新的工作目录。

然而，在某些情况下，这种行为可能会导致问题。例如：

- 依赖于相对路径：如果程序中的文件操作使用相对路径，工作目录的变化可能会导致文件访问失败。
- 多次调用：如果你的应用程序需要频繁打开文件，改变工作目录可能会使管理变得复杂，特别是在需要回到原始路径时。

假设你有一个文本编辑器应用程序，用户可以打开和编辑多个文件。在这个程序中，用户最开始可能在 C:\Documents 中打开一个文件。
OPENFILENAME ofn; // common dialog box structure

```

char szFile[260];      // buffer for file name

// Initialize OPENFILENAME
ZeroMemory(&ofn, sizeof(ofn));
ofn.lStructSize = sizeof(ofn);
ofn.hwndOwner = hwnd;
ofn.lpstrFile = szFile;
ofn.lpstrFile[0] = '\\0';
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFilter = "Text Files\\0*.TXT\\0All Files\\0*.*\\0";
ofn.nFilterIndex = 1;
ofn.lpstrFileTitle = NULL;
ofn.nMaxFileTitle = 0;
ofn.lpstrInitialDir = NULL;
ofn.lpstrTitle = "Open File";
ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST | OFN_NOCHANGEDIR;

// Display the Open dialog box
if (GetOpenFileName(&ofn)) {
    // 用户选择了文件
    // 这里可以进行文件读取等操作
}

```

在此示例中:

如果没有设置 `OFN_NOCHANGEDIR`, 选择一个位于 `D:\OtherFiles` 的文件可能会把当前工作目录从 `C:\Documents` 改为 `D:\OtherFiles`, 这意味着接下来如果程序尝试以相对路径访问文件 (例如, 读取 `data.txt`), 它会在 `D:\OtherFiles` 查找, 而不是在用户原本的路径 `C:\Documents`.

加入 `OFN_NOCHANGEDIR` 后的效果:

通过加入 `OFN_NOCHANGEDIR`, 即使用户选择了位于不同文件夹的文件, 程序的当前工作目录仍然保持在 `C:\Documents`。这样, 任何依赖于该目录的文件操作都不会受到影响。

》》》问题: 触发断点 -> 为什么将 `CreateRef` 改为 `Ref` 时会触发断点异常?

```

void EditorLayer::OpenScene()
{
    std::string filepath = FileDialogs::OpenFile("Nut Scene(*.yaml)\\0 *.yaml\\0All Files (*.*)\\0*.*\\0\\0");
    if (!filepath.empty())
    {
        m_ActiveScene = CreateRef<Scene>();
        m_ActiveScene->OnViewportResize((uint32_t)m_ViewportSize.x, (uint32_t)m_ViewportSize.y); // We use
        m_SceneHierarchyPanel.SetContext(m_ActiveScene); // We use

        SceneSerializer serializer(m_ActiveScene);
        serializer.Deserialize(filepath);
    }
}

```

因为 `Ref (std::shared_ptr<T>())` 创建了一个智能指针, 但未分配内存;

而 `CreateRef (std::make_shared<T>())` 创建并初始化一个智能指针, 同时分配内存并构造对象。

》》》关于快捷键触发这个事件函数的设计:

```

bool EditorLayer::OnKeyPressed(KeyPressedEvent& event)
{
    // You can triggering this event only once,
    // because if you try to trigger this event again,
    // the event.GetRepeatCount will bigger than 0.
    // And function will return false
    if (event.GetRepeatCount() > 0)
        return false;

    bool ctrl = Input::IsKeyPressed(NUT_KEY_LEFT_CONTROL) || Input::IsKeyPressed(NUT_KEY_RIGHT_CONTROL);
    bool shift = Input::IsKeyPressed(NUT_KEY_LEFT_SHIFT) || Input::IsKeyPressed(NUT_KEY_RIGHT_SHIFT);
    switch (event.GetKeyCode())
    {
        case NUT_KEY_N: {
            if (ctrl)
                NewScene();

            break;
        }
        case NUT_KEY_O: {
            if (ctrl)
                OpenScene();

            break;
        }
        case NUT_KEY_S: {
            if (ctrl && shift)
                SaveSceneAs();

            break;
        }
    }
}

```

1. GetRepeat() 在这里的作用：防止处理重复按键事件。

e.GetRepeatCount() 函数用于获取按键的重复次数。当一个按键被按下并保持时（比如长按），操作系统会不断生成按键按下事件，这样会导致该事件被多次触发。所以当检测到重复按键时，函数直接返回 false，意味着后续的代码（如处理快捷键的逻辑）将不会被执行。这可以防止同一个快捷键被多次触发，从而避免造成意外的重复操作。

2. 这个函数为什么只在事件重复时返回一个布尔量，其他条件下不返回值呢。为什么 GetRepeat() 需要设置在函数最前面？

在事件触发之后，我们运行实际逻辑代码（OpenScene/NewScene），但是我们不能返回 true，因为 true 不会阻止事件运行，这将会导致事件会被连续触发。其次，如果在逻辑代码之后返回 false，而不是在函数开头返回 false，都会导致事件被触发第二次，因为在判断是否重复触发时依旧会先运行逻辑代码，随后判断出来结果是 false（应该阻塞），不过此时已经没有用了。

```

bool ctrl = Input::IsKeyPressed(NUT_KEY_LEFT_CONTROL) || Input::IsKeyPressed(NUT_KEY_RIGHT_CONTROL);
bool shift = Input::IsKeyPressed(NUT_KEY_LEFT_SHIFT) || Input::IsKeyPressed(NUT_KEY_RIGHT_SHIFT);
switch (event.GetKeyCode())
{
    case NUT_KEY_N: {
        if (ctrl)
            NewScene();

        break;
    }
    case NUT_KEY_O: {
        if (ctrl)
            OpenScene();

        break;
    }
    case NUT_KEY_S: {
        if (ctrl && shift)
            SaveSceneAs();

        break;
    }
}

if (event.GetRepeatCount() > 0)
    return false;
return true;

```

第一次触发事件

不满足，故返回 true，不阻塞

```

bool ctrl = Input::IsKeyPressed(NUT_KEY_LEFT_CONTROL) || Input::IsKeyPressed(NUT_KEY_RIGHT_CONTROL);
bool shift = Input::IsKeyPressed(NUT_KEY_LEFT_SHIFT) || Input::IsKeyPressed(NUT_KEY_RIGHT_SHIFT);
switch (event.GetKeyCode())
{
    case NUT_KEY_N: {
        if (ctrl)
            NewScene();

        break;
    }
    case NUT_KEY_O: {
        if (ctrl)
            OpenScene();

        break;
    }
    case NUT_KEY_S: {
        if (ctrl && shift)
            SaveSceneAs();

        break;
    }
}

if (event.GetRepeatCount() > 0)
    return false;
return true;

```

第二次触发事件（重复触发）

会再次处理

条件满足，返回 false 表示阻塞（但此时运行了两次业务）

3. GetKeyCode 和 GetRepeatCount 中返回的 keycode 和 count 是在哪里自动获取的，所以我们才能使用其返回值来进行判断

在之前设置的回调函数中：WindowsWindow.cpp 中

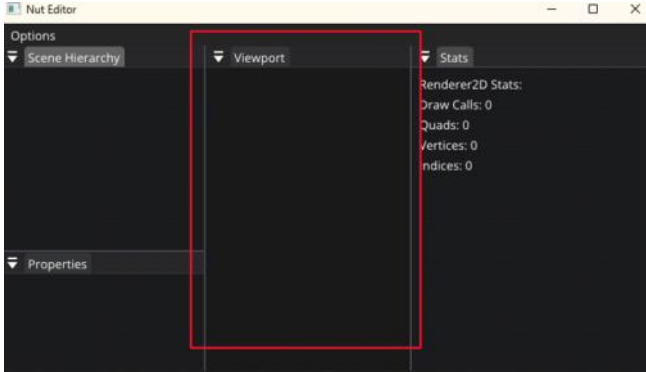
```

glfwSetKeyCallback(m_Window, [](GLFWwindow* window, int key, int scancode, int action, int mods)
{
    WindowData& data = *(WindowData*)glfwGetWindowUserPointer(window);

    switch (action)
    {
        case GLFW_PRESS: KeyPressedEvent event(key, 0); data.EventCallback(event); break; }
        case GLFW_RELEASE: KeyReleasedEvent event(key); data.EventCallback(event); break; }
        case GLFW_REPEAT: KeyPressedEvent event(key, 1); data.EventCallback(event); break; }
    }
});

```

4.为什么只有在鼠标单击了视口这个区域时（聚焦在 Viewport 窗口时），快捷键才能使用？



因为是在 EditorLayer::OnEvent() 函数中设置的事件分发。

```

void EditorLayer::OnEvent(Event& event)
{
    m_CameraController.OnEvent(event);

    EventDispatcher dispatcher(event);
    dispatcher.Dispatch<KeyPressedEvent>(NUT_BIND_EVENT_FN(EditorLayer::OnKeyPressed));
}

```

----- ImGui -----

》》》这段 premake 代码什么意思？

```

57
58     filter "files:vendor/ImGui/**.cpp"
59     flags {"NoPCH"}
60

```

》》》为什么Nut-editor 中没有包含 ImGui 库目录却可以使用 ImGui，但是没有包含 yaml-cpp 库目录的时候却不能使用 yaml-cpp？

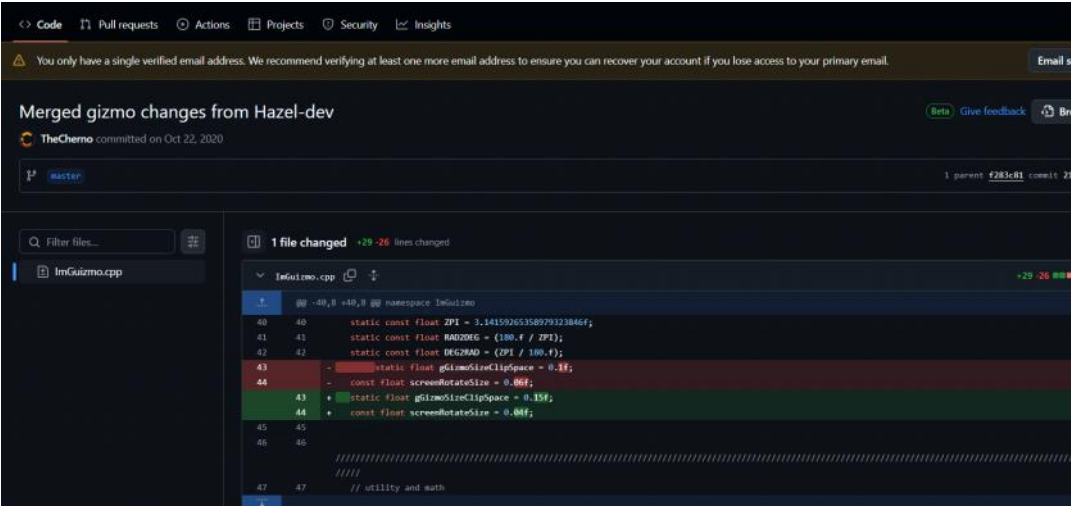
```

includedirs
{
    "${wks.location}/Nut/vendor/spdlog/include",
    "${wks.location}/Nut/src",
    "${wks.location}/Nut/vendor",
    "${IncludeDir.glm}",
    "${IncludeDir.entt}",
    "${IncludeDir.yaml_cpp}",
    "${IncludeDir.ImGui}"
}

```

》》》代码设计：gizmo库中对于 ImGui.cpp 的更改：

我猜想Cherno做的是一些对于 ImGui 样式的更改，并且由于时间原因，最新的 ImGui.cpp 结构发生了很多改变，这我不容易同步这个更改。所以我决定先保持默认值，后续再查看是否需要同步此更改。



》》》什么是万向锁，什么情况下会触发万向锁？什么是四元数，为什么四元数可以解决万向锁？ glm/gtx/quaternion.hpp 中处理四元数的函数怎么使用？

万向锁 (Gimbal Lock)

万向锁是一种在三维空间中旋转时遇到的现象，通常发生在使用欧拉角表示旋转的情况下。它指的是在某些特定的旋转配置下，系统的自由度减少，使得无法进行预期的旋转。

万向锁通常在以下情况下发生：

旋转轴对齐：	当一个旋转轴与另一个旋转轴对齐时，例如在俯仰角为 ±90° 时，两个旋转轴重合，这会导致系统失去一个自由度。
极限位置：	当物体达到某个极限位置（如直立或水平），就可能会导致无法实现某些方向的旋转。

万向锁的影响：

当发生万向锁时，物体可能会无法按预期方向旋转，或者某些旋转会变得不可用。这在动画、飞行控制和机器人运动等应用中会造成问题。

参考文档：（ <https://medium.com/@lalesena/euler-angles-rotations-and-gimbal-lock-brief-explanation-de1d4764170> ）

参考图片：

https://miro.medium.com/v2/resize:fit:498/format:webp/1*7JT7g5dLeZ-Q5uOYnX8hCw.gif
https://miro.medium.com/v2/resize:fit:400/format:webp/1*OckqKWmTtmDtzrukAX4hSA.gif

四元数 (Quaternion) 是一种扩展了复数的数学结构，通常用于表示三维空间中的旋转。它由一个实数部分和三个虚数部分组成，形式为：

$$q = w + xi + yj + zk$$

其中：w 是实数部分，x,y,z 是虚数部分（向量部分），i,j,k 是虚单位。

四元数通过以下方式解决万向锁问题：

四维表示：	四元数使用四个参数（一个实数和三个虚数）来表示旋转，而不是依赖于三个独立的角度（如俯仰、偏航和滚转）。这种表示方法避免了两条旋转轴重合的情况。
组合旋转：	四元数之间的乘法可以直接组合多个旋转，而不受单一旋转轴限制。这使得在任何方向上进行旋转都不会遭遇自由度的丧失。
插值平滑：	四元数支持球形线性插值 (Slerp)，这使得在动画中能够平滑地过渡旋转，从而避免因插值引起的万向锁问题。

参考文档---涉及到数理知识：（ <https://www.cnblogs.com/HDDDDDD/p/15067619.html> ）

》》》代码设计： TransformComponent 中的更改：

这可以防止通过 imGuizmo 调整物体旋转时图像极速发生变化的问题（通过 ImGuizmo 可视化工具轻轻更改欧拉角，角度却会极速增加，这会导致图像迅速旋转并闪烁）并且防止可能出现的万向锁问题（可以查看上述的： [万向锁 \(Gimbal Lock\)](#) ）。


```
Hazel/src/Hazel/Scene/Components.h
1  @@ -3,6 +3,9 @@
2
3  3      #include <glm/glm.hpp>
4  4      #include <glm/gtc/matrix_transform.hpp>
5  5
6  6  + #define GLM_ENABLE_EXPERIMENTAL
7  7  + #include <glm/gtx/quaternion.hpp>
8  8  +
9  9
10 6      #include "SceneCamera.h"
11 7      #include "ScriptableEntity.h"
12 8
13
14  @@ -31,9 +34,7 @@ namespace Hazel {
15
16  31 34
17  32 35          glm::mat4 GetTransform() const
18  33 36          {
19  34 -          glm::mat4 rotation = glm::rotate(glm::mat4(1.0f), Rotation.x, { 1, 0, 0 })
20  35 -          * glm::rotate(glm::mat4(1.0f), Rotation.y, { 0, 1, 0 })
21  36 -          * glm::rotate(glm::mat4(1.0f), Rotation.z, { 0, 0, 1 });
22  37 +          glm::mat4 rotation = glm::toMat4(glm::quat(Rotation));
23  37 38
24  38 39          return glm::translate(glm::mat4(1.0f), Translation)
25  39 40          * rotation
26  40
27  41
28  42
29  43
30  44
31  45
32  46
33  47
34  48
35  49
36  50
37  51
38  52
39  53
40  54
41  55
42  56
43  57
44  58
45  59
46  60
47  61
48  62
49  63
50  64
51  65
52  66
53  67
54  68
55  69
56  70
57  71
58  72
59  73
60  74
61  75
62  76
63  77
64  78
65  79
66  80
67  81
68  82
69  83
70  84
71  85
72  86
73  87
74  88
75  89
76  90
77  91
78  92
79  93
80  94
81  95
82  96
83  97
84  98
85  99
86  100
87  101
88  102
89  103
90  104
91  105
92  106
93  107
94  108
95  109
96  110
97  111
98  112
99  113
100 114
101 115
102 116
103 117
104 118
105 119
106 120
107 121
108 122
109 123
110 124
111 125
112 126
113 127
114 128
115 129
116 130
117 131
118 132
119 133
120 134
121 135
122 136
123 137
124 138
125 139
126 140
127 141
128 142
129 143
130 144
131 145
132 146
133 147
134 148
135 149
136 150
137 151
138 152
139 153
140 154
141 155
142 156
143 157
144 158
145 159
146 160
147 161
148 162
149 163
150 164
151 165
152 166
153 167
154 168
155 169
156 170
157 171
158 172
159 173
160 174
161 175
162 176
163 177
164 178
165 179
166 180
167 181
168 182
169 183
170 184
171 185
172 186
173 187
174 188
175 189
176 190
177 191
178 192
179 193
180 194
181 195
182 196
183 197
184 198
185 199
186 200
187 201
188 202
189 203
190 204
191 205
192 206
193 207
194 208
195 209
196 210
197 211
198 212
199 213
200 214
201 215
202 216
203 217
204 218
205 219
206 220
207 221
208 222
209 223
210 224
211 225
212 226
213 227
214 228
215 229
216 230
217 231
218 232
219 233
220 234
221 235
222 236
223 237
224 238
225 239
226 240
227 241
228 242
229 243
230 244
231 245
232 246
233 247
234 248
235 249
236 250
237 251
238 252
239 253
240 254
241 255
242 256
243 257
244 258
245 259
246 260
247 261
248 262
249 263
250 264
251 265
252 266
253 267
254 268
255 269
256 270
257 271
258 272
259 273
260 274
261 275
262 276
263 277
264 278
265 279
266 280
267 281
268 282
269 283
270 284
271 285
272 286
273 287
274 288
275 289
276 290
277 291
278 292
279 293
280 294
281 295
282 296
283 297
284 298
285 299
286 300
287 301
288 302
289 303
290 304
291 305
292 306
293 307
294 308
295 309
296 310
297 311
298 312
299 313
300 314
301 315
302 316
303 317
304 318
305 319
306 320
307 321
308 322
309 323
310 324
311 325
312 326
313 327
314 328
315 329
316 330
317 331
318 332
319 333
320 334
321 335
322 336
323 337
324 338
325 339
326 340
327 341
328 342
329 343
330 344
331 345
332 346
333 347
334 348
335 349
336 350
337 351
338 352
339 353
340 354
341 355
342 356
343 357
344 358
345 359
346 360
347 361
348 362
349 363
350 364
351 365
352 366
353 367
354 368
355 369
356 370
357 371
358 372
359 373
360 374
361 375
362 376
363 377
364 378
365 379
366 380
367 381
368 382
369 383
370 384
371 385
372 386
373 387
374 388
375 389
376 390
377 391
378 392
379 393
380 394
381 395
382 396
383 397
384 398
385 399
386 400
387 401
388 402
389 403
390 404
391 405
392 406
393 407
394 408
395 409
396 410
397 411
398 412
399 413
400 414
401 415
402 416
403 417
404 418
405 419
406 420
407 421
408 422
409 423
410 424
411 425
412 426
413 427
414 428
415 429
416 430
417 431
418 432
419 433
420 434
421 435
422 436
423 437
424 438
425 439
426 440
427 441
428 442
429 443
430 444
431 445
432 446
433 447
434 448
435 449
436 450
437 451
438 452
439 453
440 454
441 455
442 456
443 457
444 458
445 459
446 460
447 461
448 462
449 463
450 464
451 465
452 466
453 467
454 468
455 469
456 470
457 471
458 472
459 473
460 474
461 475
462 476
463 477
464 478
465 479
466 480
467 481
468 482
469 483
470 484
471 485
472 486
473 487
474 488
475 489
476 490
477 491
478 492
479 493
480 494
481 495
482 496
483 497
484 498
485 499
486 500
487 501
488 502
489 503
490 504
491 505
492 506
493 507
494 508
495 509
496 510
497 511
498 512
499 513
500 514
501 515
502 516
503 517
504 518
505 519
506 520
507 521
508 522
509 523
510 524
511 525
512 526
513 527
514 528
515 529
516 530
517 531
518 532
519 533
520 534
521 535
522 536
523 537
524 538
525 539
526 540
527 541
528 542
529 543
530 544
531 545
532 546
533 547
534 548
535 549
536 550
537 551
538 552
539 553
540 554
541 555
542 556
543 557
544 558
545 559
546 560
547 561
548 562
549 563
550 564
551 565
552 566
553 567
554 568
555 569
556 570
557 571
558 572
559 573
560 574
561 575
562 576
563 577
564 578
565 579
566 580
567 581
568 582
569 583
570 584
571 585
572 586
573 587
574 588
575 589
576 590
577 591
578 592
579 593
580 594
581 595
582 596
583 597
584 598
585 599
586 600
587 601
588 602
589 603
590 604
591 605
592 606
593 607
594 608
595 609
596 610
597 611
598 612
599 613
600 614
601 615
602 616
603 617
604 618
605 619
606 620
607 621
608 622
609 623
610 624
611 625
612 626
613 627
614 628
615 629
616 630
617 631
618 632
619 633
620 634
621 635
622 636
623 637
624 638
625 639
626 640
627 641
628 642
629 643
630 644
631 645
632 646
633 647
634 648
635 649
636 650
637 651
638 652
639 653
640 654
641 655
642 656
643 657
644 658
645 659
646 660
647 661
648 662
649 663
650 664
651 665
652 666
653 667
654 668
655 669
656 670
657 671
658 672
659 673
660 674
661 675
662 676
663 677
664 678
665 679
666 680
667 681
668 682
669 683
670 684
671 685
672 686
673 687
674 688
675 689
676 690
677 691
678 692
679 693
680 694
681 695
682 696
683 697
684 698
685 699
686 700
687 701
688 702
689 703
690 704
691 705
692 706
693 707
694 708
695 709
696 710
697 711
698 712
699 713
700 714
701 715
702 716
703 717
704 718
705 719
706 720
707 721
708 722
709 723
710 724
711 725
712 726
713 727
714 728
715 729
716 730
717 731
718 732
719 733
720 734
721 735
722 736
723 737
724 738
725 739
726 740
727 741
728 742
729 743
730 744
731 745
732 746
733 747
734 748
735 749
736 750
737 751
738 752
739 753
740 754
741 755
742 756
743 757
744 758
745 759
746 760
747 761
748 762
749 763
750 764
751 765
752 766
753 767
754 768
755 769
756 770
757 771
758 772
759 773
760 774
761 775
762 776
763 777
764 778
765 779
766 780
767 781
768 782
769 783
770 784
771 785
772 786
773 787
774 788
775 789
776 790
777 791
778 792
779 793
780 794
781 795
782 796
783 797
784 798
785 799
786 800
787 801
788 802
789 803
790 804
791 805
792 806
793 807
794 808
795 809
796 810
797 811
798 812
799 813
800 814
801 815
802 816
803 817
804 818
805 819
806 820
807 821
808 822
809 823
810 824
811 825
812 826
813 827
814 828
815 829
816 830
817 831
818 832
819 833
820 834
821 835
822 836
823 837
824 838
825 839
826 840
827 841
828 842
829 843
830 844
831 845
832 846
833 847
834 848
835 849
836 850
837 851
838 852
839 853
840 854
841 855
842 856
843 857
844 858
845 859
846 860
847 861
848 862
849 863
850 864
851 865
852 866
853 867
854 868
855 869
856 870
857 871
858 872
859 873
860 874
861 875
862 876
863 877
864 878
865 879
866 880
867 881
868 882
869 883
870 884
871 885
872 886
873 887
874 888
875 889
876 890
877 891
878 892
879 893
880 894
881 895
882 896
883 897
884 898
885 899
886 900
887 901
888 902
889 903
890 904
891 905
892 906
893 907
894 908
895 909
896 910
897 911
898 912
899 913
900 914
901 915
902 916
903 917
904 918
905 919
906 920
907 921
908 922
909 923
910 924
911 925
912 926
913 927
914 928
915 929
916 930
917 931
918 932
919 933
920 934
921 935
922 936
923 937
924 938
925 939
926 940
927 941
928 942
929 943
930 944
931 945
932 946
933 947
934 948
935 949
936 950
937 951
938 952
939 953
940 954
941 955
942 956
943 957
944 958
945 959
946 960
947 961
948 962
949 963
950 964
951 965
952 966
953 967
954 968
955 969
956 970
957 971
958 972
959 973
960 974
961 975
962 976
963 977
964 978
965 979
966 980
967 981
968 982
969 983
970 984
971 985
972 986
973 987
974 988
975 989
976 990
977 991
978 992
979 993
980 994
981 995
982 996
983 997
984 998
985 999
986 1000
987 1001
988 1002
989 1003
990 1004
991 1005
992 1006
993 1007
994 1008
995 1009
996 1010
997 1011
998 1012
999 1013
1000 1014
1001 1015
1002 1016
1003 1017
1004 1018
1005 1019
1006 1020
1007 1021
1008 1022
1009 1023
1010 1024
1011 1025
1012 1026
1013 1027
1014 1028
1015 1029
1016 1030
1017 1031
1018 1032
1019 1033
1020 1034
1021 1035
1022 1036
1023 1037
1024 1038
1025 1039
1026 1040
1027 1041
1028 1042
1029 1043
1030 1044
1031 1045
1032 1046
1033 1047
1034 1048
1035 1049
1036 1050
1037 1051
1038 1052
1039 1053
1040 1054
1041 1055
1042 1056
1043 1057
1044 1058
1045 1059
1046 1060
1047 1061
1048 1062
1049 1063
1050 1064
1051 1065
1052 1066
1053 1067
1054 1068
1055 1069
1056 1070
1057 1071
1058 1072
1059 1073
1060 1074
1061 1075
1062 1076
1063 1077
1064 1078
1065 1079
1066 1080
1067 1081
1068 1082
1069 1083
1070 1084
1071 1085
1072 1086
1073 1087
1074 1088
1075 1089
1076 1090
1077 1091
1078 1092
1079 1093
1080 1094
1081 1095
1082 1096
1083 1097
1084 1098
1085 1099
1086 1100
1087 1101
1088 1102
1089 1103
1090 1104
1091 1105
1092 1106
1093 1107
1094 1108
1095 1109
1096 1110
1097 1111
1098 1112
1099 1113
1100 1114
1101 1115
1102 1116
1103 1117
1104 1118
1105 1119
1106 1120
1107 1121
1108 1122
1109 1123
1110 1124
1111 1125
1112 1126
1113 1127
1114 1128
1115 1129
1116 1130
1117 1131
1118 1132
1119 1133
1120 1134
1121 1135
1122 1136
1123 1137
1124 1138
1125 1139
1126 1140
1127 1141
1128 1142
1129 1143
1130 1144
1131 1145
1132 1146
1133 1147
1134 1148
1135 1149
1136 1150
1137 1151
1138 1152
1139 1153
1140 1154
1141 1155
1142 1156
1143 1157
1144 1158
1145 1159
1146 1160
1147 1161
1148 1162
1149 1163
1150 1164
1151 1165
1152 1166
1153 1167
1154 1168
1155 1169
1156 1170
1157 1171
1158 1172
1159 1173
1160 1174
1161 1175
1162 1176
1163 1177
1164 1178
1165 1179
1166 1180
1167 1181
1168 1182
1169 1183
1170 1184
1171 1185
1172 1186
1173 1187
1174 1188
1175 1189
1176 1190
1177 1191
1178 1192
1179 1193
1180 1194
1181 1195
1182 1196
1183 1197
1184 1198
1185 1199
1186 1200
1187 1201
1188 1202
1189 1203
1190 1204
1191 1205
1192 1206
1193 1207
1194 1208
1195 1209
1196 1210
1197 1211
1198 1212
1199 1213
1200 1214
1201 1215
1202 1216
1203 1217
1204 1218
1205 1219
1206 1220
1207 1221
1208 1222
1209 1223
1210 1224
1211 1225
1212 1226
1213 1227
1214 1228
1215 1229
1216 1230
1217 1231
1218 1232
1219 1233
1220 1234
1221 1235
1222 1236
1223 1237
1224 1238
1225 1239
1226 1240
1227 1241
1228 1242
1229 1243
1230 1244
1231 1245
1232 1246
1233 1247
1234 1248
1235 1249
1236 1250
1237 1251
1238 1252
1239 1253
1240 1254
1241 1255
1242 1256
1243 1257
1244 1258
1245 1259
1246 1260
1247 1261
1248 1262
1249 1263
1250 1264
1251 1265
1252 1266
1253 1267
1254 1268
1255 1269
1256 1270
1257 1271
1258 1272
1259 1273
1260 1274
1261 1275
1262 1276
1263 1277
1264 1278
1265 1279
1266 1280
1267 1281
1268 1282
1269 1283
1270 1284
1271 1285
1272 1286
1273 1287
1274 1288
1275 1289
1276 1290
1277 1291
1278 1292
1279 1293
1280 1294
1281 1295
1282 1296
1283 1297
1284 1298
1285 1299
1286 1300
1287 1301
1288 1302
1289 1303
1290 1304
1291 1305
1292 1306
1293 1307
1294 1308
1295 1309
1296 1310
1297 1311
1298 1312
1299 1313
1300 1314
1301 1315
1302 1316
1303 1317
1304 1318
1305 1319
1306 1320
1307 1321
1308 1322
1309 1323
1310 1324
1311 1325
1312 1326
1313 1327
1314 1328
1315 1329
1316 1330
1317 1331
1318 1332
1319 1333
1320 1334
1321 1335
1322 1336
1323 1337
1324 1338
1325 1339
1326 1340
1327 1341
1328 1342
1329 1343
1330 1344
1331 1345
1332 1346
1333 1347
1334 1348
1335 1349
1336 1350
1337 1351
1338 1352
1339 1353
1340 1354
1341 1355
1342 1356
1343 1357
1344 1358
1345 1359
1346 1360
1347 1361
1348 1362
1349 1363
1350 1364
1351 1365
1352 1366
1353 1367
1354 1368
1355 1369
1356 1370
1357 1371
1358 1372
1359 1373
1360 1374
1361 1375
1362 1376
1363 1377
1364 1378
1365 1379
1366 1380
1367 1381
1368 1382
1369 1383
1370 1384
1371 1385
1372 1386
1373 1387
1374 1388
1375 1389
1376 1390
1377 1391
1378 1392
1379 1393
1380 1394
1381 1395
1382 1396
1383 1397
1384 1398
1385 1399
1386 1400
1387 1401
1388 1402
1389 1403
1390 1404
1391 1405
1392 1406
1393 1407
1394 1408
1395 1409
1396 1410
1397 1411
1398 1412
1399 1413
1400 1414
1401 1415
1402 1416
1403 1417
1404 1418
1405 1419
1406 1420
1407 1421
1408 1422
1409 1423
1410 1424
1411 1425
1412 1426
1413 1427
1414 1428
1415 1429
1416 1430
1417 1431
1418 1432
1419 1433
1420 143
```

```
const float* snap = nullptr, const float* pivot = nullptr);
```

参数解析:

view:	相机的视图矩阵（cameraView），表示相机的位置 and 方向。
projection:	相机的投影矩阵（cameraProjection），用于确定场景中物体的透视效果。
operation:	变换操作类型（平移、旋转或缩放），通过 (ImGui::OPERATION)m_GizmoType 指定。
mode:	变换模式，通常是 ImGui::LOCAL（局部变换）或 ImGui::GLOBAL（全局变换）。
matrix:	物体的变换矩阵（transform），表示物体在场景中的位置、旋转和缩放。
deltaMatrix:	可选参数，表示变化的矩阵。
snap:	可选参数，指定对变换进行捕捉的值（例如，步长）。如果为 nullptr，则不进行捕捉。
pivot:	可选参数，指定旋转的中心点。

》》》ImGui::OPERATION 的定义

```
// call it when you want a gizmo
// Needs view and projection matrices.
// matrix parameter is the source matrix (where will be gizmo be drawn) and might be transformed by the function. Return deltaMatrix is optional
// translation is applied in world space
enum OPERATION
{
    TRANSLATE_X = (1u << 0),
    TRANSLATE_Y = (1u << 1),
    TRANSLATE_Z = (1u << 2),
    ROTATE_X    = (1u << 3),
    ROTATE_Y    = (1u << 4),
    ROTATE_Z    = (1u << 5),
    ROTATE_SCREEN = (1u << 6),
    SCALE_X     = (1u << 7),
    SCALE_Y     = (1u << 8),
    SCALE_Z     = (1u << 9),
    BOUNDS     = (1u << 10),
    SCALE_XU    = (1u << 11),
    SCALE_YU    = (1u << 12),
    SCALE_ZU    = (1u << 13),

    TRANSLATE = TRANSLATE_X | TRANSLATE_Y | TRANSLATE_Z,
    ROTATE    = ROTATE_X | ROTATE_Y | ROTATE_Z | ROTATE_SCREEN,
    SCALE     = SCALE_X | SCALE_Y | SCALE_Z,
    SCALEU    = SCALE_XU | SCALE_YU | SCALE_ZU, // universal
    UNIVERSAL = TRANSLATE | ROTATE | SCALEU
};
```

》》》代码设计：窗口响应

```
216 - Application::Get().GetImGuiLayer()->BlockEvents(!m_ViewportFocused || !m_ViewportHovered);
220 + Application::Get().GetImGuiLayer()->BlockEvents(!m_ViewportFocused && !m_ViewportHovered);
```

更改前：只要不满足 聚焦于窗口上/悬停在窗口上的任——一个条件，便会阻塞当前窗口中的事件（不再捕获鼠标或键盘的活动）

更改后：只有 窗口没有被聚焦且鼠标没有悬停在窗口上的时候，才会阻塞事件。

这让我们在结构面板中选中实体之后，只需要将鼠标悬停在 viewport 窗口上，便可以通过键盘调整/响应 Gizmo，这在实际使用中很舒服（本人亲测：>

》》》为什么这里需要使用 Const 标识?

```
Entity Scene::GetPrimaryCamera()
{
    auto view = m_Registry.view<CameraComponent>().
    for (auto entity : view)
    {
        const auto& cameraComponent = m_Registry.get<CameraComponent>(entity);
        if (cameraComponent.Primary == true)
            return Entity{ entity, this };
    }
    return {};
}
```

可能是因为没有需要对 cameraComponent 进行更改吧。

》》》这个函数的意义?

```
#pragma once

#include <glm/glm.hpp>

namespace Nut { namespace Math {
    bool DecomposeTransform(const glm::mat4& transform, glm::vec3& outTranslation, glm::vec3& outRotation, glm::vec3& outScale);
}}

```

》》》这段代码的意义？

```
if (ImGui::IsUsing())
{
    glm::vec3 translation, rotation, scale;
    Math::DecomposeTransform(transform, translation, rotation, scale);
    glm::vec3 deltaRotation = rotation - tc.Rotation;
    tc.Translation = translation;
    tc.Rotation += deltaRotation;
    tc.Scale = scale;
}

```

----- Multiple Render Targets and Framebuffer refactor -----

》》》gl_VertexID 在 GLSL 中关于顶点ID的一些细节：

gl_VertexID

gl_Position和gl_PointSize都是**输出变量**，因为它们的值是作为顶点着色器的输出被读取的。我们可以对它们进行写入，来改变结果。顶点着色器还为我们提供了一个有趣的**输入变量**，我们只能对它进行读取，它叫做gl_VertexID。

整型变量gl_VertexID储存了正在绘制顶点的当前ID。当（使用glDrawElements）进行索引渲染的时候，这个变量会存储正在绘制顶点的当前索引。当（使用glDrawArrays）不使用索引进行绘制的时候，这个变量会储存从渲染调用开始的已处理顶点数量。

虽然现在它没有什么具体的用途，但知道我们能够访问这个信息总是好的。

----- Maintenance -----

》》》后续会整合 Jul/22 2022 之后的维护