

----- scene viewport -----

》》》做了两件事：设置视口和设置相机比例

》》》为什么要设置 m_ViewportSize 为 glm::vec2 而不是 ImVec2？

因为后面需要进行 != 运算，而 ImVec2 没有这个运算符的定义，只有 glm::vec2 有这个运算符的定义。

```
template<typename T, qualifier Q>
GLM_FUNC_QUALIFIER GLM_CONSTEXPR bool operator!=(vec<2, T, Q> const& v1, vec<2, T, Q> const& v2)
{
    return !(v1 == v2);
}
```

所以需要 ImVec2 接收 GetContentRegionAvail 返回的 ImVec2 类型的 panelSize，然后将两者进行比较。


```
ImVec2 panelSize = ImGui::GetContentRegionAvail();
if (m_ViewportSize != *(glm::vec2*)&panelSize)
{
}
```

》》》发现一个问题

```
ImVec2 panelSize = ImGui::GetContentRegionAvail(); // 获取面板大小
if (m_ViewportSize != *(glm::vec2*)&panelSize)
{
    m_ViewportSize = { panelSize.x, panelSize.y }; // 及时更新视口大小
    m_Framebuffer->Resize(m_ViewportSize.x, m_ViewportSize.y);
    m_CameraController.Resize(m_ViewportSize.x, m_ViewportSize.y);
}
ImGuiTextureID textureID = (void*)m_Framebuffer->GetColorAttachmentRendererID();
ImGui::Image(textureID, ImVec2{ m_ViewportSize.x, m_ViewportSize.y }, ImVec2{ 0, 1 }, ImVec2{ 1, 0 });
```



其中，无论对 m_Framebuffer 是否调用 Resize，其渲染结果和响应好像都是一样的，并没有什么影响（实际上这应该对图像的分辨率有一定影响，但为何我没有发现什么明确特征？）。而且不调用 Framebuffer->Resize 的话，调整窗口大小的时候图像并不会出现闪烁的现象。（所以说闪烁正是因为帧缓冲对纹理附件的刷新而导致的）


》》》另一个问题


 @KennyTutorials 4年前

Im found a little bug when we double click on title bar / border, then engine is crashed and say that framebuffer isn't complete. Maybe this bug only on my engine? I think we just destroy the viewport window, but at same time trying drawing framebuffer on this window.

当我们双击标题栏/边框时，我发现了一个小错误，然后引擎崩溃并说帧缓冲区不完整。也许这个错误只出现在我的引擎上？我认为我们只是销毁视口窗口，但同时尝试在此窗口上绘制帧缓冲区。



  回复


 3 条回复

 @FlukierJupiter 4年前

use the ImGuiTabBarFlags_NoTooltip flag when creating the ImGui window to prevent the window collapsing



创建 ImGui 窗口时使用 ImGuiTabBarFlags_NoTooltip 标志以防止窗口折叠


  回复

 @FlukierJupiter 4年前

you could just return from the invalidate function if the width or height is zero, before creating the frame buffer

如果宽度或高度为零，则在创建帧缓冲区之前，您可以从无效函数返回

 2  回复

 @KennyTutorials 4年前

@FlukierJupiter Thanks for help! It works)

@FlukierJupiter 感谢您的帮助！有用)

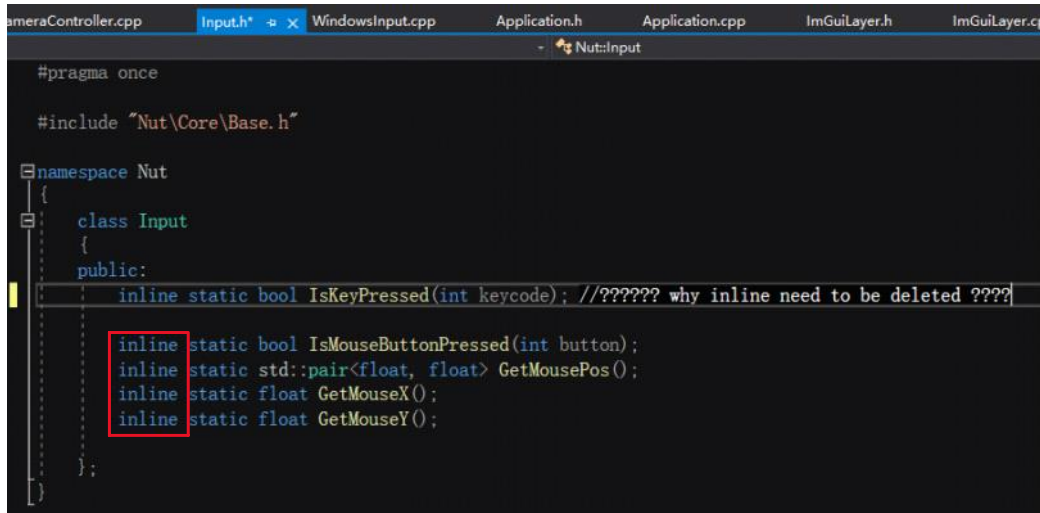
》》》值得一提的是，相机的纵横比更新函数参数需要为 float 类型的，而不是 uint 类型，否则会导致窗口尺寸过小时无渲染结果。

```
void OrthographicCameraController::Resize(float width, float height)
{
    m_AspectRatio = width / height;
    UpdateViewport();
}
```

-----ImGui Layer Events-----

》》》发现一个问题:

Hazel中有一次维护是删除 inline 关键字的,我大致看了眼,觉得没有必要,就没有提交到 Nut,只是添加到待办里面了,这导致一个问题。



```
#pragma once

#include "Nut\Core\Base.h"

namespace Nut
{
    class Input
    {
    public:
        inline static bool IsKeyPressed(int keycode); //?????? why inline need to be deleted ???
        inline static bool IsMouseButtonPressed(int button);
        inline static std::pair<float, float> GetMousePos();
        inline static float GetMouseX();
        inline static float GetMouseY();
    };
}
```

操作:

在简化了Input.h之后,只剩下了5个函数的声明,而且这些函数在简化前都是内联函数,在.h文件中就已经定义过了。

建议:

所以在删除掉了定义之后,还应该删除inline关键字,我们要确保使用 inline 关键字的时候就对函数在头文件中定义,否则不添加inline关键字,避免出现错误。

如果仅仅删除了定义,但是没有删除inline关键字,就会出现 LNK2019 的报错,比如:

"public: static bool __cdecl Nut::Input::IsKeyPressed(int)" (?IsKeyPressed@Input@Nut@@@SA_NH@Z),
函数 "public: void __cdecl Nut::OrthoGraphicCameraController::OnUpdate(class Nut::Timestep)" (?OnUpdate@OrthoGraphicCameraController@Nut@@@QEAXVTimestep@2@@Z) 中引用了该符号。

问题:

OrthoCameraController本应使用函数,可是为什么会查找不到,或者说对这个函数链接失败呢?

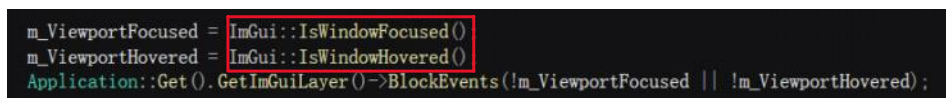
原因:

这正是因为我头文件中只声明了函数为 inline,然后没在头文件中定义这个函数,而是在 CPP 文件中定义它。

此时编译器会在编译时找不到这个函数的定义,因为头文件已经告诉编译器这是一个 inline 函数,并期望在头文件中找到它的实现。

这样会导致链接错误或重复定义错误,这全都由于 CPP 文件中的定义与头文件中的 inline 声明不匹配。

》》》提醒:



```
m_ViewportFocused = ImGui::IsWindowFocused();
m_ViewportHovered = ImGui::IsWindowHovered();
Application::Get().GetImGuiLayer()->BlockEvents(!m_ViewportFocused || !m_ViewportHovered);
```

记得不要写成 ImGui::IsWindowFocused; ;)

-----Where to go + Code review (前瞻与代码审核)-----

》》》Cherno 所做的

Cherno 在这一集前8分钟修复了一个小Bug,然后就算是开始审核代码了,基本上讲了自己对游戏引擎的理解与期望,还有接下来的进程。

》》》我将提交一些维护代码,因为这一集也没什么要做的。

》》》调整ImGui窗口大小时闪烁的原因是:我们在绘制ImGui窗口时同步更新了FrameBuffer和Camera



```
ImVec2 panelSize = ImGui::GetContentRegionAvail();
if (m_ViewportSize != (glm::vec2*)&panelSize)
{
    m_ViewportSize = { panelSize.x, panelSize.y };
    m_Framebuffer->Resize(panelSize.x, panelSize.y);
    m_CameraController.Resize(panelSize.x, panelSize.y);
}
```

我们应该先更新，后绘制。

问题出现原理以及解决方法：

在 `OnImGuiRender` 函数中处理窗口大小变化时，你会在每一帧的渲染过程中检查窗口尺寸并同时处理窗口尺寸。因为在窗口调整时你重新创建了帧缓冲（`Framebuffer`），那么在调整过程中的某些渲染操作可能会使用未完全准备好的新帧缓冲，这就会导致显示的内容不稳定，从而产生闪烁。
将窗口大小的调整逻辑提前放在 `Onupdate` 函数中，可以确保在每一帧的渲染之前已经完成了所有的帧缓冲调整。这意味着当 `ImGuiRender` 执行时，帧缓冲已经是正确的状态，减少了因帧缓冲调整导致的闪烁现象。

未准备好的帧缓冲概念：

1. 帧缓冲（`Framebuffer`）重建过程：

当窗口大小变化时，通常需要重新创建或调整帧缓冲的尺寸，以适应新的窗口尺寸。这个过程包括删除旧的帧缓冲对象并创建新的对象，同时可能需要重新分配或调整与之关联的纹理和深度缓冲区。

2. 未准备好的帧缓冲：

在帧缓冲重新创建或调整的过程中，新的帧缓冲可能尚未完全配置和初始化。例如，新的纹理可能尚未正确分配或绑定，或者深度缓冲区的设置尚未完成。在这个过渡期间，帧缓冲可能处于一个不稳定的状态，无法正确显示内容。

》》》另外，还要注意一个问题：

```
void EditorLayer::OnUpdate(Timestep ts)
{
    NUT_PROFILE_FUNCTION(); // 一个作用域只能声明一个 Timer 变量

    // Logic Update
    ImVec2 panelSize = ImGui::GetContentRegionAvail();
    if (m_ViewportSize != *((glm::vec2*)&panelSize) && m_ViewportSize.x > 0.0f && m_ViewportSize.y > 0.0f) { // 及时更新视口大小
        m_ViewportSize = { panelSize.x, panelSize.y };
        m_Framebuffer->Resize((uint32_t)panelSize.x, (uint32_t)panelSize.y);
        m_CameraController.Resize(panelSize.x, panelSize.y);
    }

    if ((m_ViewportSize.x != m_Framebuffer->GetSpecification().Width || m_ViewportSize.y != m_Framebuffer->GetSpecification().Height)
        && m_ViewportSize.x > 0.0f && m_ViewportSize.y > 0.0f) {
        m_Framebuffer->Resize((uint32_t)m_ViewportSize.x, (uint32_t)m_ViewportSize.y);
        m_CameraController.Resize(m_ViewportSize.x, m_ViewportSize.y);
    }

    // Screen Update
    if (m_ViewportFocused)
        m_CameraController.OnUpdate(ts);
}
```

第一种逻辑更新方式是不可取的，因为 `ImGui::GetContentRegionAvail()` 获取的是当前 `ImGui` 窗口的面板大小，需要在 `ImGui` 窗口绘制范围内进行使用，否则获取的 `Window` 值为 `nullptr`，即没有找到可获取的 `ImGui` 窗口。

第二种方式可取，因为每一次 `m_Viewport` 在 `ImGui` 窗口事件触发时更新后，每当下一次绘制开始执行 `OnUpdate` 函数，`m_ViewportSize` 已经是新窗口尺寸，而 `specification` 中存储的是旧窗口尺寸。这时触发 `Resize` 函数，随后帧缓冲 `m_Framebuffer` 更新，相应的帧缓冲 `m_Framebuffer` 中存储的 `specification` 也会更新为新窗口尺寸。

下一次窗口大小改变时，也是类似的操作。

逻辑：

需要注意的是，这里的 `m_Viewport` 值是在 `Onupdate` 函数执行后更新的，也就是说，图像的更新逻辑为：当前绘制时先判断逻辑，然后执行绘制。检测窗口尺寸变化的代码确实在绘制函数中，不过没有直接绘制，而是将新窗口尺寸保留在全局变量 `m_ViewportSize` 中，在下次绘制开启前更新窗体逻辑，然后在绘制函数中更新实际窗口尺寸。（简而言之，就是：当前帧检测到变化，但不更新，在下一帧开始时，发送变化值并执行更新）