

----- SPIR-V & New shader system -----

》》》这次 Cherno 做了很多提交，所以我的笔记可能篇幅较长，但我会仔细记录。
请认真浏览。

》》》 basic architecture layout of this episode (本集基本构架)
(前庭仅供个人参考，并无侵犯版权的想法，若违反版权条款，并非本人意图)

个人在学习过程中觉得值得查阅的几个文档：

游戏开发者大会文档 (关于 SPIR-V 与 渲染接口 OpenGL/Vulkan、GLSL/HLSL 之间的关系，SPIR-V 的工具及其执行流程)	https://www.neilhenning.dev/wp-content/uploads/2015/03/AnIntroductionToSPIR-V.pdf
俄勒冈州立大学演示文档 (SPIR-V 与 GLSL 之间的关系， SPIR-V 的实际使用方法：Win10)	https://web.engr.oregonstate.edu/~mjb/cs557/Handouts/VulkanGLSL1pp.pdf
Vulkan 官方 Github Readme 文档 (GLSL 与 SPIR-V 之间的映射关系，以及可以在线使用的编辑器，非常好用)	https://github.com/KhronosGroup/Vulkan-Guide/blob/main/chapters/mapping_data_to_shaders.adoc
	在线文档示例 (https://godbolt.org/z/oMys8a78T)
大原Khronos开发者大会 (SPIR-V 语言的规范，及其意义)	https://www.lunarg.com/wp-content/uploads/2023/05/SPIRV-Osaka-MAY2023.pdf

前 33 分钟，基本上讲述以下几点：

<p>1.着色器将会支持 OpenGL 和 Vulkan，故着色器中做了更改（涉及到 OpenGL 和 Vulkan 在着色器语法上的不同：比如 Uniform 的使用）</p> <p>2.为了避免性能浪费，并高效的使用数据/统一变量，将采用 UniformBuffer 这种高级 GLSL。</p> <p>(参考文档1-来自 LearnOpenGL 教程：https://learnopengl-cn.github.io/04%20Advanced%20OpenGL/08%20Advanced%20GLSL/)</p> <p>(参考文档2-来自 Vulkan 教程：https://vulkan-tutorial.com/Uniform_buffers/Descriptor_layout_and_buffer#page-Uniform-buffer)</p> <p>建议阅读全文，这样理解更加深刻。</p>	<p>• Uniform buffer</p> <h3>使用Uniform缓冲</h3> <p>我们已经讨论了如何在着色器中定义Uniform块，并设定它们的内存布局了，但我们还没有讨论如何使用它们。</p> <p>首先，我们需要调用 <code>glGenBuffers</code>，创建一个Uniform缓冲对象。一旦我们有了一个缓冲对象，我们需要将它绑定到 <code>GL_UNIFORM_BUFFER</code> 目标，并调用 <code>glBufferData</code>，分配足够的内存。</p> <pre>unsigned int uboExampleBlock; glGenBuffers(1, &uboExampleBlock); glBindBuffer(GL_UNIFORM_BUFFER, uboExampleBlock); glBufferData(GL_UNIFORM_BUFFER, 152, NULL, GL_STATIC_DRAW); // 分配152字节的内存 glBindBuffer(GL_UNIFORM_BUFFER, 0);</pre> <p>现在，每当我们需要对缓冲更新或插入数据，我们就会绑定到 <code>uboExampleBlock</code>，并使用 <code>glBufferSubData</code> 来更新它的内存。我们只需要更新这个Uniform缓冲一次，所有使用这个缓冲的着色器就都使用的是更新后的数据了。但是，如何才能让OpenGL知道哪个Uniform缓冲对应的是哪个Uniform块呢？</p> <p>在OpenGL上下文中，定义了一些绑定点(Binding Point)，我们可以将一个Uniform缓冲中链接至它。在创建Uniform缓冲之后，我们将它绑定到其中一个绑定点上，并将着色器中的Uniform块绑定到相同的绑定点，把它们连接到一起。下面的这个图示例了这个：</p> <p>• Uniform buffer.</p> <h3>Uniform buffer 均匀缓冲</h3> <p>In the next chapter we'll specify the buffer that contains the UBO data for the shader, but we need to create this buffer first. We're going to copy new data to the uniform buffer every frame, so it doesn't really make any sense to have a staging buffer. It would just add extra overhead in this case and likely degrade performance instead of improving it.</p> <p>在下一章中，我们将指定包含着色器 UBO 数据的缓冲。但我们需要首先创建该缓冲。我们将每帧复制数据到该缓冲。因此，我们不需要 staging 缓冲，因为它在这种情况下只会增加额外的开销，并可能会降低性能而不是提高性能。</p> <p>We should have multiple buffers, because multiple frames may be in flight at the same time and we don't want to update the buffer in preparation of the next frame while a previous one is still reading from it! Thus, we need to have as many uniform buffers as we have frames in flight, and write to a uniform buffer that is not currently being read by the GPU</p> <p>我们应该有多个缓冲，因为多个帧可能同时在飞行。我们不想在上一帧仍在读取时更新缓冲（以准备下一帧）。因此，我们需要拥有与飞行中的帧一样多的统一缓冲，并写入 GPU 当前未读取的统一缓冲。</p> <p>To that end, add new class members for <code>uniformBuffers</code>, and <code>uniformBuffersMemory</code>:</p> <p>为此，为 <code>uniformBuffers</code> 和 <code>uniformBuffersMemory</code> 添加新的类成员：</p> <pre>VkBuffer IndexBuffer; VkDeviceMemory IndexBufferMemory; std::vector<VkBuffer> uniformBuffers; std::vector<VkDeviceMemory> uniformBuffersMemory; std::vector<uint> uniformBuffersMapped;</pre> <p>Similarly, create a new function <code>createUniformBuffers</code> that is called after <code>createIndexBuffer</code> and allocates the buffers:</p> <p>类似地，创建一个新函数 <code>createUniformBuffers</code>，该函数在 <code>createIndexBuffer</code> 之后调用并分配缓冲：</p> <pre>void InitVulkan() { ... createVertexBuffer(); createIndexBuffer(); createUniformBuffers(); }</pre>
<p>3.OpenGL 和 Vulkan 在着色器语言上的使用规范，还有不同之处。</p> <p>参考文献：OpenGL教程 (https://learnopengl-cn.github.io/02%20Lighting/03%20Materials/)</p> <p>参考文献：俄勒冈州立大学演示文件《GLSL For Vulkan》 (https://eecs.oregonstate.edu/~mjb/cs557/Handouts/VulkanGLSL1pp.pdf)</p> <p>附录： 参考文献：Github 中文 Readme (https://github.com/zenny-chen/GLSL-for-Vulkan)</p>	<p>• GLSG 中的结构体示例：</p> <pre>#version 330 core struct Material { vec3 ambient; vec3 diffuse; vec3 specular; float shininess; }; uniform Material material;</pre> <p>在片段着色器中，我们创建一个结构体(struct)来存储物体的材质属性。我们也可以把它们存储为独立的uniform值。但是作为一个结构体来存储生命会有多建一些。我们首先定义结构体的布局(layout)，然后简单地以刚创建的结构体作为类型声明一个uniform变量。</p> <p>• 如果查看 Vulkan API 在编写着色器时使用 GLSL 的语法规则，可以查看 Github 仓库 (中文：https://github.com/zenny-chen/GLSL-for-Vulkan) 或查看 Vulkan 的官方入门指南 (http://vulkan-tutorial.com/Introduction)</p>

参考文献: Vulkan 教程官网 (<https://vulkan-tutorial.com/Introduction>)

或前往 Vulkan! 教程门户网站 (<https://vulkan-tutorial.com/Introduction/>)

•不同之处:

How Vulkan GLSL Differs from OpenGL GLSL

4

Detecting that a GLSL Shader is being used with Vulkan/SPIR-V:

- In the compiler, there is an automatic `#define VULKAN 100`


Vulkan Vertex and Instance Indices:

```
gl_VertexIndex
gl_InstanceIndex
```

- Both are 0-based

gl_FragColor:

- In OpenGL, `gl_FragColor` broadcasts to all color attachments
- In Vulkan, it just broadcasts to color attachment location 0
- Best idea: don't use it at all – explicitly declare out variables to have specific location numbers



oeb - December 17, 2020

How Vulkan GLSL Differs from OpenGL GLSL

5

Shader combinations of separate texture data and samplers:

```
uniform sampler s;
uniform texture2D t;
vec4 rgba = texture( sampler2D( t, s ), vST );
```

Descriptor Sets:

```
layout( set=0, binding=0 ) ... ;
```

Push Constants:

```
layout( push_constant ) ... ;
```

Specialization Constants:


```
layout( constant_id = 3 ) const int N = 5;
```

- Only for scalars, but a vector's components can be constructed from specialization constants

Specialization Constants for Compute Shaders:

```
layout( local_size_x_id = 8, local_size_y_id = 16 );
```

- This sets `gl_WorkGroupSize.x` and `gl_WorkGroupSize.y`
`gl_WorkGroupSize.z` is set as a constant



oeb - December 17, 2020

4.SPIR-V的使用思路，使用逻辑。

参考文献: SPIR-V 官网 (https://www.khronos.org/api/index_2017/spir)

参考文献: Vulkan 教程 (https://vulkan-tutorial.com/Drawing_a_triangle/Graphics_pipeline_basics/Shader_modules)

参考文献: Vulkan 指南 (https://docs.vulkan.org/guide/latest/what_is_spirv.html)

参考文献: 俄勒冈州立大学演示文件 (<https://web.engr.oregonstate.edu/~mjb/cs557/Handouts/VulkanGLSL.1pp.pdf>)

参考文献: 2016 年 3 月 - 游戏开发者大会 (<https://www.neilheming.dev/wp-content/uploads/2015/03/AnIntroductionToSPIR-V.pdf>)

附件: 关于 SPIR-V 也可以参考 SPIR-V 的 github 仓库: (<https://github.com/KhronosGroup/SPIRV-Guide>)

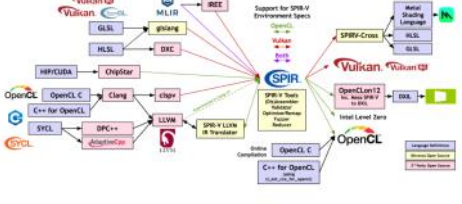
• SPIR-V 的生态系统:

SPIR-V Language Ecosystem

SPIR-V 语言生态系统

The SPIR-V ecosystem includes a rich variety of language front-ends (compilers), development tools and run-times (consumers).

该生态系统包括丰富的语言前端(编译器)、开发工具和运行时(消费者)。



• SPIR-V 的概念:

Unlike earlier APIs, shader code in Vulkan has to be specified in a bytecode format as opposed to human-readable syntax like **GLSL** and **HLSL**. This bytecode format is called **SPIR-V** and is designed to be used with both Vulkan and OpenCL (both Khronos APIs). It is a format that can be used to write graphics and compute shaders, but we will focus on shaders used in Vulkan's graphics pipelines in this tutorial.

与早期的 API 不同, Vulkan 中的着色器代码必须以字节码格式指定,而不是像 **GLSL** 和 **HLSL** 这样的人类可读语法。这种字节码格式称为 **SPIR-V**, 现在与 Vulkan 和 OpenCL (均为 Khronos API) 一起使用。它是一种可用于编写图形和计算着色器的格式。但在本教程中我们将重点关注 Vulkan 图形管道中使用的着色器。

Vulkan Guide / Logistics Overview / What is SPIR-V

What is SPIR-V 什么是 SPIR-V

NOTE

Please read the [SPIRV Guide](#) for more in detail information about SPIR-V

请阅读[SPIRV 指南](#),了解有关 SPIR-V 的更详细的信息

SPIR-V is a binary intermediate representation for graphical-shader stages and compute kernels. With Vulkan, an application can still write their shaders in a high-level shading language such as GLSL or HLSL, but a SPIR-V binary is needed when using `vkCreateShaderModule`. Khronos has a very nice [white paper](#) about SPIR-V and its advantages, and a high-level description of the representation. There are also two great Khronos presentations from Vulkan DevDay 2016 [here](#) and [here](#) (video of both).

SPIR-V是面向着色器阶段和计算内核的二进制中间表示。使用 Vulkan, 应用程序仍然可以使用高级着色语言(例如 GLSL 或 HLSL)编写着色器,但使用 `vkCreateShaderModule` 时需要 SPIR-V 二进制文件。Khronos 有一些关于 SPIR-V 及其优点的非常好的[白皮书](#),以及对其表示的清晰描述。[这里](#)和[这里](#)还有 2016 年 Vulkan DevDay 的两种精彩的 Khronos 演示 [\(两者的视频\)](#)。

• SPIR-V 管线:

	<p>从 OpenGL 4.5 开始，OpenGL 也支持通过 SPIR-V 加载编译好的着色器二进制文件。流程与 Vulkan 类似，只不过 OpenGL 在内部做了更多的高层封装。</p> <p>加载过程：</p> <p>示例：</p> <pre>GLuint program = glCreateProgram(); // 加载 SPIR-V 二进制文件 GLuint shader = glCreateShader(GL_VERTEX_SHADER); glShaderBinary(1, &shader, GL_SHADER_BINARY_FORMAT_SPIR_V, spirvData, spirvDataSize); glSpecializeShader(shader, "main", 0, nullptr, nullptr); // 绑定和链接程序 glAttachShader(program, shader); glLinkProgram(program);</pre> <p>-----</p> <p>3.2 在 Vulkan 中使用 SPIR-V</p> <p>加载过程：</p> <p>创建一个 VkShaderModule 对象，该对象包含 SPIR-V 二进制代码。</p> <p>使用 SPIR-V 二进制代码来创建 Vulkan 着色器管线（例如，创建顶点着色器和片段着色器的管线）。</p> <p>示例：Vulkan 使用 SPIR-V</p> <pre>// 加载 SPIR-V 文件（假设你已经将 shader.spv 文件加载为二进制数据） VkShaderModuleCreateInfo createInfo = {}; createInfo.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO; createInfo.codeSize = shaderData.size(); createInfo.pCode = reinterpret_cast<const uint32_t*>(shaderData.data()); // 创建着色器模块 VkShaderModule shaderModule; VkResult result = vkCreateShaderModule(device, &createInfo, nullptr, &shaderModule); // 使用这个 shaderModule 来创建图形管线</pre>
4. 执行着色器程序	<p>在 OpenGL 中，SPIR-V 着色器程序被链接到程序对象中，并通过调用 <code>glUseProgram</code> 来激活该程序，之后通过绘制调用来执行。</p> <p>在 Vulkan 中，着色器被绑定到渲染管线或计算管线中，随后可以通过绘制命令（例如 <code>vkCmdDraw</code>）或计算命令（例如 <code>vkCmdDispatch</code>）来执行。</p>

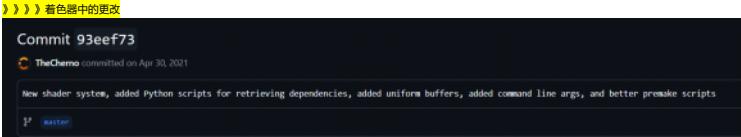
》》》上述涉及语言的纵向对比图

GLSL	<pre>#version 330 core in vec3 fragColor; // 从顶点着色器传递过来的颜色 out vec4 FragColor; // 输出颜色到屏幕 void main() { FragColor = vec4(fragColor, 1.0); // 输出最终颜色 }</pre>
<p>SPIR-V</p> <p>SPIR-V 本身的核心是一个二进制格式，然而为了便于开发和调试，SPIR-V 也可以以类似汇编语言的文本形式表达，这种形式通常称为 SPIR-V Assembly。</p> <p>它是 SPIR-V 的一种可读性较好的文本表示方式，开发者可以通过这种形式来编写、调试和优化 SPIR-V 代码，然后再将其转换为二进制格式以供图形 API 使用。</p> <p>实际上，SPIR-V Assembly 代码最终还是会通过工具（如 spirv-as）转化为二进制格式，供 Vulkan 或 OpenGL 使用。</p>	<p>SPIR-V</p> <pre>#302 2307 0000 0100 0100 0000 0e00 0000 0000 0000 1100 0200 0100 0000 0000 0000 0100 0000 474c 534c 2e73 7464 2e34 3530 0000 0000 0e00 0300 0000 0000 0100 0000 0f00 0000 0400 0000 0400 0000 0d01 050e 0000 0000 0900 0000 1000 0300 0400 0000 0700 0000 0300 0300 0200 0000 8c00 0000 0500 0400 0400 0000 0d01 050e 0000 0000 0500 0000 0900 0000 070c 5f46 7261 0743 0f0c 0f72 0000 0000 1300 0200 0200 0000 2100 0300 0300 0000 0200 0000 1600 0300 0600 0000 2000 0000 1700 0400 0700 0000 0600 0000 0400 0000 2000 0400 0800 0000 0300 0000 0700 0000 3000 0400 0800 0000 0900 0000 0300 0000 2000 0400 0800 0000 0a00 0000 cdc c3e 2000 0400 0600 0000 0b00 0000 cdc c43f 2000 0400 0600 0000 0c00 0000 0000 803f 2c00 0700 0700 0000 0d00 0000 0a00 0000 0a00 0000 0000 0000 0e00 0000 3600 0500 0200 0000 0400 0000 0000 0000 0300 0000 f800 0200 0500 0000 1e00 0100 0900 0000 0000 0000 f000 0100 3800 0100</pre> <p>SPIR-V Assembly</p> <pre>; SPIR-V ; Version: 1.0 ; Generator: Khronos GLSLang Reference Front End; 1 ; Bound: 14 ; Schema: 0 OpCapability Shader %1 = OpExtInstImport "GLSL.std.450" OpMemoryModel Logical GLSL450 OpEntryPoint Fragment %4 "main" %9 OpExecutionMode %4 OriginUpperLeft OpSource GLSL 450 OpName %4 "main" OpName %9 "out_colour" OpDecorate %9 Location 0 %2 = OpTypeVoid %3 = OpTypeFunction %2 %6 = OpTypeFloat 32 %7 = OpTypeVector %6 4 %8 = OpTypePointer Output %7 %9 = OpVariable %8 Output %10 = OpConstant %6 0.4 %11 = OpConstant %6 0.8 %12 = OpConstant %6 1 %13 = OpConstantComposite %7 %10 %10 %11 %12 %4 = OpFunction %2 None %3 %5 = OpLabel OpStore %9 %13 OpReturn OpFunctionEnd</pre>
OpenGL	<pre>GLuint shaderProgram = glCreateProgram(); glAttachShader(shaderProgram, vertexShader); glAttachShader(shaderProgram, fragmentShader); glLinkProgram(shaderProgram); glUseProgram(shaderProgram); // 主要渲染循环 while (!glfwWindowShouldClose(window)) { glClear(GL_COLOR_BUFFER_BIT); glUseProgram(shaderProgram);</pre>

Vulkan

```
VkInstance instance;
VkApplicationInfo appInfo = {};
appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
appInfo.pApplicationName = "Vulkan 示例";
appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
appInfo.pEngineName = "No Engine";
appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
appInfo.apiVersion = VK_API_VERSION_1_0;

VkInstanceCreateInfo createInfo = {};
createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
createInfo.pApplicationInfo = &appInfo;
```



以下是详细解释:

1 premake脚本更改
(and better premake scripts)

2 + -- Hazel Dependencies
3 +
4 + VULKAN_SDK = os.getenv("VULKAN_SDK")

```
15 + IncludeDir["shaders"] = "${wks.location}/Hazel/vendor/shaders/include"
16 + IncludeDir["SPIRV_Cross"] = "${wks.location}/Hazel/vendor/SPIRV-Cross"
17 + IncludeDir["VulkanSDK"] = "${VULKAN_SDK}/include"
18 +
19 + LibraryDir = {}
20 +
21 + LibraryDir["VulkanSDK"] = "${VULKAN_SDK}/lib"
22 + LibraryDir["VulkanSDK_Debug"] = "${wks.location}/Hazel/vendor/VulkanSDK/lib"
23 +
24 + Library = {}
25 + Library["Vulkan"] = "${LibraryDir.VulkanSDK}/vulkan-1.lib"
26 + Library["VulkanGLES"] = "${LibraryDir.VulkanSDK}/VkLayer_utils.lib"
27 +
28 + Library["ShaderC_Debug"] = "${LibraryDir.VulkanSDK_Debug}/shader_shared.lib"
29 + Library["SPIRV_Cross_Debug"] = "${LibraryDir.VulkanSDK_Debug}/spirv-cross-core.lib"
30 + Library["SPIRV_Cross_GLES_Debug"] = "${LibraryDir.VulkanSDK_Debug}/spirv-cross-gles.lib"
31 + Library["SPIRV_Tools_Debug"] = "${LibraryDir.VulkanSDK_Debug}/SPIRV_Tools.lib"
32 +
33 + Library["ShaderC_Release"] = "${LibraryDir.VulkanSDK}/shader_shared.lib"
34 + Library["SPIRV_Cross_Release"] = "${LibraryDir.VulkanSDK}/spirv-cross-core.lib"
35 + Library["SPIRV_Cross_GLES_Release"] = "${LibraryDir.VulkanSDK}/spirv-cross-gles.lib"
```

premake5.lua

```
1 1  @ 3.4 - 4.5  @
2 1  include "${vendor/premake/premake_customization/solution_items.lua"
3 2  + include "Dependencies.lua"
4 3  workspace "Hazel"
5 4  architecture "x86_64"
6 5  @ 23,17 +24,6  @ workspace "Hazel"
23 24
24 25  outputdir = "${cfg.buildcfg}-%{cfg.system}-%{cfg.architecture}"
25 26
26 27  -- Include directories relative to root folder (solution directory)
27 28  IncludeDir = {}
28 29  IncludeDir["GLFW"] = "${wks.location}/Hazel/vendor/GLFW/include"
29 30  IncludeDir["Glad"] = "${wks.location}/Hazel/vendor/Glad/include"
30 31  IncludeDir["ImGui"] = "${wks.location}/Hazel/vendor/ImGui"
31 32  IncludeDir["glm"] = "${wks.location}/Hazel/vendor/glm"
32 33  IncludeDir["vth_image"] = "${wks.location}/Hazel/vendor/vth_image"
33 34  IncludeDir["entt"] = "${wks.location}/Hazel/vendor/entt/include"
34 35  IncludeDir["yaml_cpp"] = "${wks.location}/Hazel/vendor/yaml-cpp/include"
35 36  IncludeDir["glfw3"] = "${wks.location}/Hazel/vendor/glfw3"
36
37 27  group "Dependencies"
38 28  include "vendor/premake"
39 29  include "Hazel/vendor/GLFW"
40 30
```

Hazel/premake5.lua

```
1 1  @ 2,7 +2,7  @ project "Hazel"
2 2  kind "ConsoleApp"
3 3  language "C++"
4 4  cppdialect "C++17"
5 5  staticruntime "on"
6 6  staticruntime "off"
7 7  targetdir ("${wks.location}/bin" .. outputdir .. "/" .. prj.name)
8 8  objdir ("${wks.location}/bin-int" .. outputdir .. "/" .. prj.name)
9 9
```

Hazel/premake5.lua

```
1 1  @ 2,7 +2,7  @ project "Hazel"
2 2  kind "StaticLib"
3 3  language "C++"
4 4  cppdialect "C++17"
5 5  staticruntime "on"
6 6  staticruntime "off"
7 7  targetdir ("${wks.location}/bin" .. outputdir .. "/" .. prj.name)
8 8  objdir ("${wks.location}/bin-int" .. outputdir .. "/" .. prj.name)
9 9
10 10  @ 40,7 +40,8  @ project "Hazel"
40 40  IncludeDir["vth_image"],
41 41  IncludeDir["entt"],
42 42  IncludeDir["yaml_cpp"],
43 43  IncludeDir["glfw3"],
44 44  IncludeDir["VulkanSDK"],
45 45  IncludeDir["VulkanSDK_Debug"],
46 46  IncludeDir["VulkanSDK_Release"],
47 47  IncludeDir["VulkanSDK_GLES"],
48 48  IncludeDir["VulkanSDK_GLES_Debug"],
49 49  IncludeDir["VulkanSDK_GLES_Release"],
50 50  IncludeDir["VulkanSDK_GLES_GLES_Debug"],
51 51  IncludeDir["VulkanSDK_GLES_GLES_Release"],
52 52  IncludeDir["VulkanSDK_GLES_GLES_GLES_Debug"],
53 53  IncludeDir["VulkanSDK_GLES_GLES_GLES_Release"]
54 54
```

分区 GameEngine 的第 5 页

```
73 +         "H[library.ShaderC_Debug]",
74 +         "H[library.SPIRV_Cross_Debug]",
75 +         "H[library.SPIRV_Cross_GLSL_Debug]"
76 +     }
77 +
78 +     filter "configurations:Release"
79 +     defines "NDEBUG"
80 +     runtime "Release"
81 +     optimize "on"
82 +
83 +     links
84 +     [
85 +         "H[library.ShaderC_Release]",
86 +         "H[library.SPIRV_Cross_Release]",
87 +         "H[library.SPIRV_Cross_GLSL_Release]"
88 +     ]
89 +
90 +     filter "configurations:Dist"
91 +     defines "NDEBUG"
92 +     runtime "Release"
93 +     optimize "on"
94 +
95 +     links
96 +     [
97 +         "H[library.ShaderC_Release]",
98 +         "H[library.SPIRV_Cross_Release]",
99 +         "H[library.SPIRV_Cross_GLSL_Release]"
100 +     ]
```

2.py脚本
(Python scripts for retrieving dependencies)

```
1 + import subprocess
2 + import pkg_resources
3 +
4 + def install(package):
5 +     print(f"Installing {package} module...")
6 +     subprocess.check_call([python, "-m", "pip", "install", package])
7 +
8 + def validate_package(package):
9 +     required = { package }
10 +     installed = {pkg.key for pkg in pkg_resources.working_set}
11 +     missing = required - installed
12 +
13 +     if missing:
14 +         install(package)
15 +
16 + def validate_packages():
17 +     validate_package('requests')
18 +     validate_package('fake-useragent')
```

1. 确保在执行过程中 requests 和 fake-useragent 这两个模块已经安装。如果没有安装，它会自动使用 pip 安装它们。

```
1 + import os
2 + import subprocess
3 + import CheckPython
4 +
5 + # Make sure everything we need is installed
6 + CheckPython.ValidatePackages()
7 +
8 + import Vulkan
9 +
10 + # Change from Scripts directory to root
11 + os.chdir('../')
12 +
13 + if (not Vulkan.CheckVulkanSDK()):
14 +     print("Vulkan SDK not installed.")
15 +
16 + if (not Vulkan.CheckVulkanSDKDebugLibs()):
17 +     print("Vulkan SDK debug libs not found.")
18 +
19 + print("Running premake...")
20 + subprocess.call(["cmd", "premake5.exe", "-c2019"])
```

- 1. 确保所需的 Python 包已经安装。
- 2. 检查 Vulkan SDK 是否安装，并确保 Vulkan SDK 的调试库存在。
- 3. 改变当前工作目录到项目根目录。
- 4. 使用 premake 工具生成 Visual Studio 2019 项目的构建文件。

```
1 + import requests
2 + import sys
3 + import time
4 +
5 + from fake_useragent import UserAgent
6 +
7 + def DownloadFile(url, filepath):
8 +     with open(filepath, 'wb') as f:
9 +         ua = UserAgent()
10 +         headers = {'User-Agent': ua.stream}
11 +         response = requests.get(url, headers=headers, stream=True)
12 +         total = response.headers.get('content-length')
13 +
14 +         if total is None:
15 +             f.write(response.content)
16 +         else:
17 +             downloaded = 0
18 +             total = int(total)
19 +             startTime = time.time()
20 +             for data in response.iter_content(chunk_size=max(int(total/1000), 1024*1024)):
21 +                 downloaded += len(data)
22 +                 f.write(data)
23 +                 done = int(10*downloaded/total)
24 +                 percentage = (downloaded / total) * 100
25 +                 elapsedTime = time.time() - startTime
26 +                 avgPerSecond = (downloaded / 1024) / elapsedTime
27 +                 avgSpeedString = '{:2f} MB/s'.format(avgPerSecond)
28 +                 if (avgPerSecond > 1024):
29 +                     avgPerSecond = avgPerSecond / 1024
30 +                 avgSpeedString = '{:2f} MB/s'.format(avgPerSecond)
```

DownloadFile(url, filepath) 函数的作用是从指定 URL 下载文件，并显示实时的下载进度（包括下载进度条和速度）。
YesOrNo() 函数用于与用户进行交互，获取用户的确认输入，返回布尔值表示“是”或“否”。

```

scripts/Vulkan.py
""" @ -1.0 +1.00 """
1 + import os
2 + import subprocess
3 + import sys
4 + from pathlib import Path
5 +
6 + import Wills
7 +
8 + from io import BytesIO
9 + from urllib.request import urlopen
10 + from zipfile import ZipFile
11 +
12 + VULKAN_SDK = os.environ.get("VULKAN_SDK")
13 + VULKAN_SDK_INSTALLER_URL = "https://sdk.lunarg.com/sdk/download/1.2.139.0/windows/vulkan_sdk.exe"
14 + HAZEL_VULKAN_VERSION = "1.2.139.0"
15 + VULKAN_SDK_EXE_PATH = "hazel/vendor/VulkanSDK/VulkanSDK.exe"
16 +
17 + def InstallVulkanSDK():
18 +     print(f"Downloading {} to {}".format(VULKAN_SDK_INSTALLER_URL, VULKAN_SDK_EXE_PATH))
19 +     Wills.DownloadFile(VULKAN_SDK_INSTALLER_URL, VULKAN_SDK_EXE_PATH)
20 +     print("Done!")
21 +     print("Running Vulkan SDK installer...")
22 +     os.startfile(os.path.abspath(VULKAN_SDK_EXE_PATH))
23 +     print("Do run this script after installation")
24 +
25 + def InstallVulkanPrompt():

```

用于检查和安装 Vulkan SDK

InstallVulkanSDK(): 下载并运行 Vulkan SDK 安装程序。

InstallVulkanPrompt(): 提示用户是否安装 Vulkan SDK。

CheckVulkanSDK(): 检查 Vulkan SDK 是否安装并且版本是否正确。

CheckVulkanSDKDebugLibs(): 检查 Vulkan SDK 的调试库是否存在，如果缺失则下载并解压。

3 Application 中的 ApplicationCommandLineArgs
(added command line args)

```

Hazel/src/Hazel/Core/Application.cpp
1
2 @ -13,7 +13,8 @ namespace Hazel {
13
14
15     Application* Application::s_Instance = nullptr;
16
17     Application::Application(const std::string& name)
18     {
19         m_CommandLineArgs(args);
20
21     }
22
23     HZ_PROFILE_FUNCTION();
24
25

```

```

Hazel/src/Hazel/Core/Application.h
18 + struct ApplicationCommandLineArgs
19 + {
20 +     int Count = 0;
21 +     char** Args = nullptr;
22 +
23 +     const char* operator[](int index) const
24 +     {
25 +         HZ_ASSERT(index < Count);
26 +         return Args[index];
27 +     }
28 + };
29 +
30 + class Application
31 + {
32 + public:
33 +     Application(const std::string& name = "Hazel App");
34 +     Application(const std::string& name = "Hazel App", ApplicationCommandLineArgs args = {});
35 +     virtual ~Application();
36 +
37 +     void GetEvent(Event& e);
38 +
39 + @ -13,11 +13,14 @ namespace Hazel {
40 +     Application* GetInstance() { return s_Instance; }
41 +
42 +     ApplicationCommandLineArgs GetCommandLineArgs() const { return m_CommandLineArgs; }
43 +
44 + private:
45 +     void Run();
46 +     bool OnWindowClose(Window& window);
47 +     bool OnWindowResize(Window& window);
48 +
49 + private:
50 +     ApplicationCommandLineArgs m_CommandLineArgs;
51 +     ScopeWindow m_Window;
52 +     WindowLayer m_WindowLayer;
53 +     bool m_Running = true;

```

```

Hazel/src/Hazel/Core/EntryPoint.h
1
2 @ -1,10 +1,17 @
1 + #pragma once
2 + #include "Hazel/Core/Base.h"
3 + #include "Hazel/Core/Application.h"
4 +
5 + #ifdef HZ_PLATFORM_WINDOWS
6 +     extern Hazel::Application* Hazel::CreateApplication();
7 +     extern Hazel::Application* Hazel::CreateApplication(ApplicationCommandLineArgs args);
8 +
9 + int main(int argc, char** argv)
10 + {
11 +     Hazel::Log::Init();
12 +
13 +     HZ_PROFILE_BEGIN_SESSION("Startup", "HazelProfile-Startup.json");
14 +     auto app = Hazel::CreateApplication();
15 +     auto app = Hazel::CreateApplication({argc, argv});
16 +     HZ_PROFILE_END_SESSION();
17 +
18 +     HZ_PROFILE_BEGIN_SESSION("Runtime", "HazelProfile-Runtime.json");

```

```

Hazel/src/Editor/EditorLayer.cpp
1
2 @ -13,6 +13,14 @ namespace Hazel {
13
14
15     m_ActiveScene = CreateRef<Scene>();
16
17     auto commandLineArgs = Application::Get().GetCommandLineArgs();
18     if (commandLineArgs.Count > 1)
19     {
20         auto sceneFilePath = commandLineArgs[1];
21         SceneSerializer serializer(m_ActiveScene);
22         serializer.Deserialize(sceneFilePath);
23     }
24
25     m_EditorCamera = EditorCamera(50.0f, 1.770f, 0.1f, 1000.0f);
26
27
28 #if 0

```

4 Uniform Buffer 的定义以及使用, 包括着色器更新 (added uniform buffers)

```

1 // Hazel/src/Hazel/Renderer/Renderer2D.cpp
2
3 #include "Hazel/Renderer/Renderer2D.h"
4
5 #include "Hazel/Renderer/Shader.h"
6 #include "Hazel/Renderer/VertexArray.h"
7 #include "Hazel/Renderer/RenderCommand.h"
8
9 #include <glm/gtc/matrix_transform.hpp>
10 #include <glm/gtc/type_ptr.hpp>
11
12 namespace Hazel {
13
14     @@@ @@@ @@@ @@@ @@@ namespace Hazel {
15
16     glm::vec4 QuadVertexPosition[4];
17
18     Renderer2D::Statistics Stats;
19
20     struct CameraData
21     {
22         glm::mat4 ViewProjection;
23         CameraData* View;
24         Ref<Shader> ForwardPass; CameraData* ForwardPass;
25     };
26
27 };

```


s 着色器系统更新:
(New shader system)

Timer 的定义

着色器更新

```
#include <stdio.h>
#include <iostream>
using namespace std;

int main() {
    int n;
    while (n) {
        int x = n % 10;
        if (x == 0) continue;
        printf("%d ", x);
        n /= 10;
    }
    return 0;
}
```

```

1  #include <nc/Platform/Windows/WindowsPlatformUtils.cpp>
2
3  namespace Haxl {
4
5  //
6
7  //
8
9  //
10
11  //
12
13  //
14
15  //
16
17  //
18
19  //
20
21  //
22
23  //
24
25  //
26
27  //
28
29  //
30
31  //
32
33  //
34
35  //
36
37  //
38
39  //
40
41  //
42
43  //
44
45  //
46
47  //
48
49  //
50
51  //
52
53  //
54
55  //
56
57  //
58
59  //
60
61  //
62
63  //
64
65  //
66
67  //
68
69  //
70
71  //
72
73  //
74
75  //
76
77  //
78
79  //
80
81  //
82
83  //
84
85  //
86
87  //
88
89  //
90
91  //
92
93  //
94
95  //
96
97  //
98
99  //
100
101  //
102
103  //
104
105  //
106
107  //
108
109  //
110
111  //
112
113  //
114
115  //
116
117  //
118
119  //
120
121  //
122
123  //
124
125  //
126
127  //
128
129  //
130
131  //
132
133  //
134
135  //
136
137  //
138
139  //
140
141  //
142
143  //
144
145  //
146
147  //
148
149  //
150
151  //
152
153  //
154
155  //
156
157  //
158
159  //
160
161  //
162
163  //
164
165  //
166
167  //
168
169  //
170
171  //
172
173  //
174
175  //
176
177  //
178
179  //
180
181  //
182
183  //
184
185  //
186
187  //
188
189  //
190
191  //
192
193  //
194
195  //
196
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
227  //
228
229  //
230
231  //
232
233  //
234
235  //
236
237  //
238
239  //
240
241  //
242
243  //
244
245  //
246
247  //
248
249  //
250
251  //
252
253  //
254
255  //
256
257  //
258
259  //
260
261  //
262
263  //
264
265  //
266
267  //
268
269  //
270
271  //
272
273  //
274
275  //
276
277  //
278
279  //
280
281  //
282
283  //
284
285  //
286
287  //
288
289  //
290
291  //
292
293  //
294
295  //
296
297  //
298
299  //
300
301  //
302
303  //
304
305  //
306
307  //
308
309  //
310
311  //
312
313  //
314
315  //
316
317  //
318
319  //
320
321  //
322
323  //
324
325  //
326
327  //
328
329  //
330
331  //
332
333  //
334
335  //
336
337  //
338
339  //
340
341  //
342
343  //
344
345  //
346
347  //
348
349  //
350
351  //
352
353  //
354
355  //
356
357  //
358
359  //
360
361  //
362
363  //
364
365  //
366
367  //
368
369  //
370
371  //
372
373  //
374
375  //
376
377  //
378
379  //
380
381  //
382
383  //
384
385  //
386
387  //
388
389  //
390
391  //
392
393  //
394
395  //
396
397  //
398
399  //
400
401  //
402
403  //
404
405  //
406
407  //
408
409  //
410
411  //
412
413  //
414
415  //
416
417  //
418
419  //
420
421  //
422
423  //
424
425  //
426
427  //
428
429  //
430
431  //
432
433  //
434
435  //
436
437  //
438
439  //
440
441  //
442
443  //
444
445  //
446
447  //
448
449  //
450
451  //
452
453  //
454
455  //
456
457  //
458
459  //
460
461  //
462
463  //
464
465  //
466
467  //
468
469  //
470
471  //
472
473  //
474
475  //
476
477  //
478
479  //
480
481  //
482
483  //
484
485  //
486
487  //
488
489  //
490
491  //
492
493  //
494
495  //
496
497  //
498
499  //
500
501  //
502
503  //
504
505  //
506
507  //
508
509  //
510
511  //
512
513  //
514
515  //
516
517  //
518
519  //
520
521  //
522
523  //
524
525  //
526
527  //
528
529  //
530
531  //
532
533  //
534
535  //
536
537  //
538
539  //
540
541  //
542
543  //
544
545  //
546
547  //
548
549  //
550
551  //
552
553  //
554
555  //
556
557  //
558
559  //
560
561  //
562
563  //
564
565  //
566
567  //
568
569  //
570
571  //
572
573  //
574
575  //
576
577  //
578
579  //
580
581  //
582
583  //
584
585  //
586
587  //
588
589  //
590
591  //
592
593  //
594
595  //
596
597  //
598
599  //
600
601  //
602
603  //
604
605  //
606
607  //
608
609  //
610
611  //
612
613  //
614
615  //
616
617  //
618
619  //
620
621  //
622
623  //
624
625  //
626
627  //
628
629  //
630
631  //
632
633  //
634
635  //
636
637  //
638
639  //
640
641  //
642
643  //
644
645  //
646
647  //
648
649  //
650
651  //
652
653  //
654
655  //
656
657  //
658
659  //
660
661  //
662
663  //
664
665  //
666
667  //
668
669  //
670
671  //
672
673  //
674
675  //
676
677  //
678
679  //
680
681  //
682
683  //
684
685  //
686
687  //
688
689  //
690
691  //
692
693  //
694
695  //
696
697  //
698
699  //
700
701  //
702
703  //
704
705  //
706
707  //
708
709  //
710
711  //
712
713  //
714
715  //
716
717  //
718
719  //
720
721  //
722
723  //
724
725  //
726
727  //
728
729  //
730
731  //
732
733  //
734
735  //
736
737  //
738
739  //
740
741  //
742
743  //
744
745  //
746
747  //
748
749  //
750
751  //
752
753  //
754
755  //
756
757  //
758
759  //
760
761  //
762
763  //
764
765  //
766
767  //
768
769  //
770
771  //
772
773  //
774
775  //
776
777  //
778
779  //
780
781  //
782
783  //
784
785  //
786
787  //
788
789  //
790
791  //
792
793  //
794
795  //
796
797  //
798
799  //
800
801  //
802
803  //
804
805  //
806
807  //
808
809  //
810
811  //
812
813  //
814
815  //
816
817  //
818
819  //
820
821  //
822
823  //
824
825  //
826
827  //
828
829  //
830
831  //
832
833  //
834
835  //
836
8
```



```
18 def installVulkanSDK():
19     print("Downloading {} to {}".format(VULKAN_SDK_DOWNLOAD_URL, VULKAN_SDK_EXE_PATH))
20     utils.downloadFile(VULKAN_SDK_DOWNLOAD_URL, VULKAN_SDK_EXE_PATH)
21     print("Done!")
22     print("Running Vulkan SDK installer...")
```

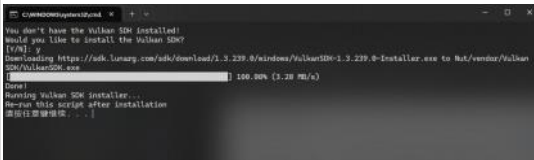
附录：如果你想进入官网查看适合你系统的 SDK，以下是网址 ->
(<https://vulkan.lunarg.com/sdk/home>)



当前我只更新了 SDK Installer 的安装地址，但是我还没有更新随后的 debug lib.zip，这是下一个问题会出现的地方，现在先不讨论。

```
48 VulkanSDKDebugLibURL = "https://files.lunarg.com/SDK/1.2.179.0/VulkanSDK-1.2.179.0-DebugLibs.zip"
49 OutputDirectory = "bat/vendor/VulkanSDK"
50 TempZipFile = f"{OutputDirectory}/VulkanSDK.zip"
51
52
53 def CheckVulkanSDKDebugLibs():
54     shadercdlib = Path(f"{OutputDirectory}/lib/shadercd.lib")
55     if (not shadercdlib.exists()):
56         print("No Vulkan SDK debug libs found. (Checked {shadercdlib})")
57         print("Downloading", VulkanSDKDebugLibURL)
58         with urlopen(VulkanSDKDebugLibURL) as zipresp:
59             with ZipFile(BytesIO(zipresp.read())) as zfile:
60                 zfile.extractall(OutputDirectory)
61         print("Vulkan SDK debug libs located at {OutputDirectory}")
62     return True
```

我们先重新运行一遍，使用更新之后的 SDK install。

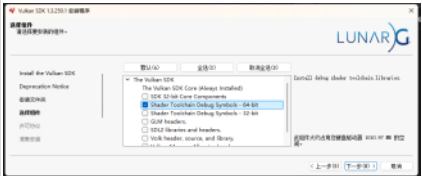


于是运行后出现这样的窗口：

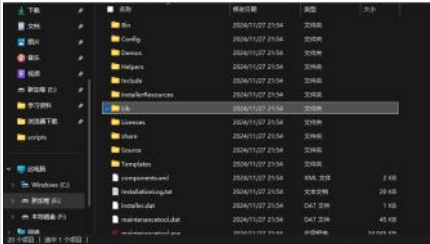


安装 vulkan SDK

我没有选择任何拓展，但在安装过程中，我不是很确定这个拓展和 DebugLibs 有没有什么直接关系。就先标注一下。（毕竟这将会占用我1G空间 bushi）



随后便得到这样的文件结构：



问题三

我们发现 Cherno 另外下载了一个 Debuglib.zip，并对其进行了一些处理。

但是在1.2.198.1版本之后，lunarg 公司不再支持 debuglibs 的单独下载。现在 SDK 中的调试库通常随着 Vulkan 库一起分发，不再单独打包成一个 zip 文件。

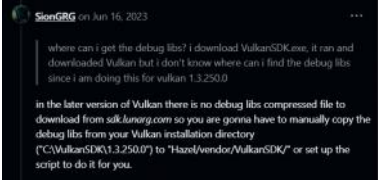
所以现在，这些文件通常直接包含在 Vulkan SDK 的核心目录下，特别是在 lib 目录中

```
48 VulkanSDKDebugLibURL = "https://files.lunarg.com/SDK/1.2.179.0/VulkanSDK-1.2.179.0-DebugLibs.zip"
49 OutputDirectory = "bat/vendor/VulkanSDK"
50 TempZipFile = f"{OutputDirectory}/VulkanSDK.zip"
51
52
53 def CheckVulkanSDKDebugLibs():
54     shadercdlib = Path(f"{OutputDirectory}/lib/shadercd.lib")
55     if (not shadercdlib.exists()):
56         print("No Vulkan SDK debug libs found. (Checked {shadercdlib})")
57         print("Downloading", VulkanSDKDebugLibURL)
58         with urlopen(VulkanSDKDebugLibURL) as zipresp:
59             with ZipFile(BytesIO(zipresp.read())) as zfile:
60                 zfile.extractall(OutputDirectory)
61         print("Vulkan SDK debug libs located at {OutputDirectory}")
62     return True
```

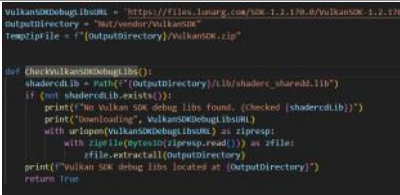
我们也可以从评论中窥见这一更改。（@SlonGRG）



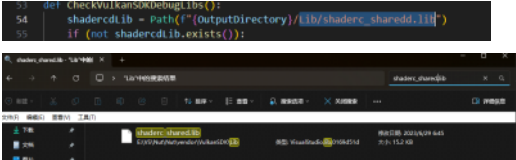
我们也可以从评论中窥见这一更改。 (@SionGRG)



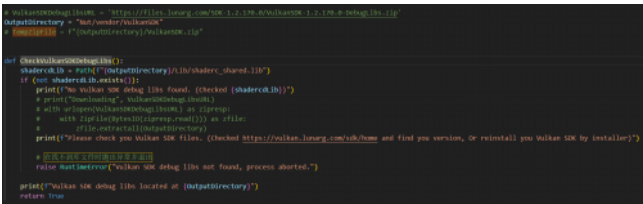
现在我们需要更改这个函数
(CheckVulkanSDKDebugLibs) 的逻辑



首先, 我对这个 shaderc_shared.lib 的路径有点疑惑: 因为我的确查找到了 shader_shared.lib 这个库, 而不是shaderc_shared.lib,

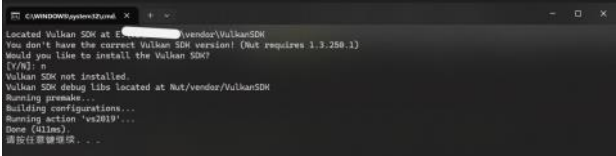


现在我开始更改, 不过我发现原先的逻辑是: 如果没有找到调试库, 就在线去下载。
但现在这些文件将会在安装 Vulkan SDK 时, 同步安装在文件夹中, 所以如果没有找到的话, 一定是安装是出了什么问题。



我便做了以下更改: (仅仅是口头提醒一下 :-)

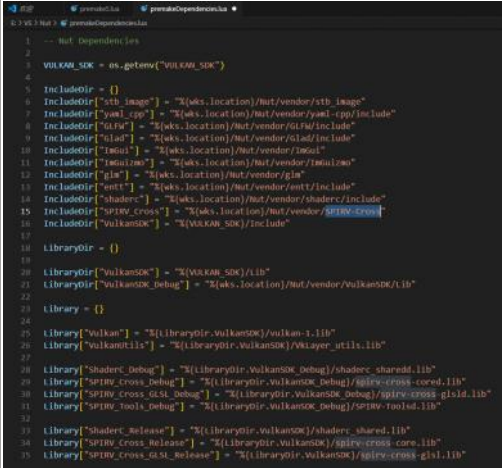
随后我重新运行 Setup.bat, 并拒绝再次安装 installer, 便得到这样的结果:



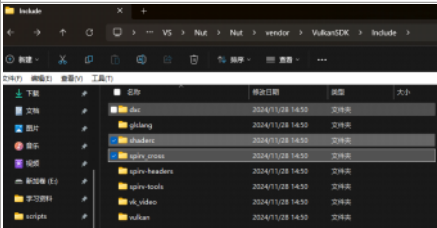
我想应该是对的。

》》》接下来我将更改 Premake 文件

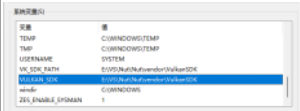
第一步, 我们在项目的根目录下重新编写一个 premake 文件, 这个文件主要用来索引 vendor 中的外部库 (API)



但我发现有些问题, 比如 shaderc 和 spirv_cross 的路径已经发生改变, 参考 1.3.250.1 版本: 这两个文件来位于 VulkanSDK/Include 下



系统变量示例:

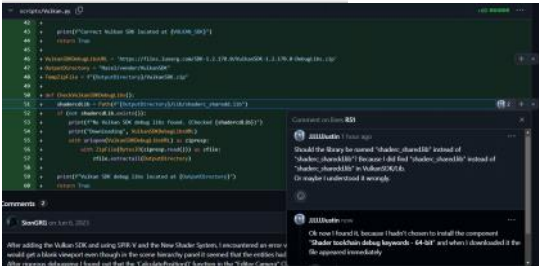


而且由于我没有下载某些组件，这使很多文件并不存在。（我将其标注出来）

[illegible]

于是我决定下载拓展，这一步通过运行 `maintenancetool.exe` 文件实现：

The screenshot shows the 'Components' tab of the Vulkan SDK installer. The 'Vulkan SDK' is expanded, revealing a list of components with checkboxes. The 'SDK 32-bit Core Components' and 'Shader Toolchain Debug Symbols - 64-bit' are checked, while the others are unchecked. The 'Install' button is visible on the right side of the window.

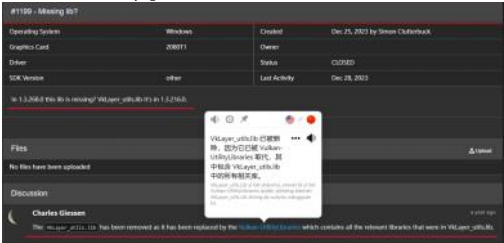


这个组件将会解决这部分问题：

[illegible]

虽然下载了一个组件可以解决但部分问题，但是尽管在之后我下载了其余的所有组件，VulkanSDK/Include/Lib 这个路径都不存在，且 VKLayer_utils.lib 这个文件也不存在。

于是我发现一个问题: VKLayer_utils.lib 似乎在1.3.216.0 版本中被移除了。



Content browser panel

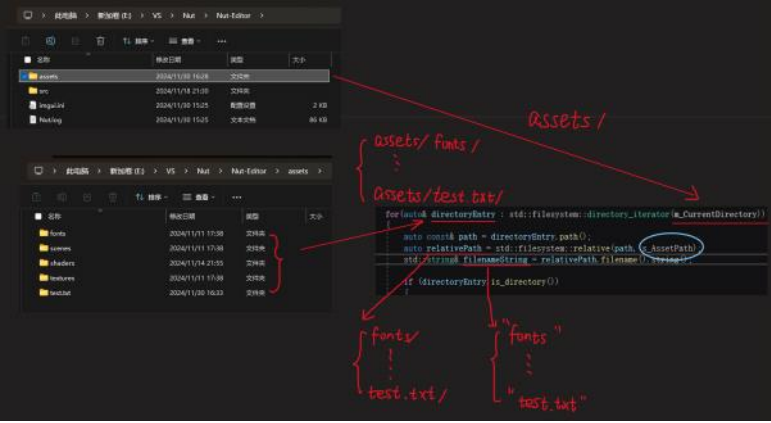
))) 上一集提交过多, 我先将内容浏览器做完, 并提交。

```
    } } } std::filesystem::relative();  
    const auto& path = directoryEntry.path();  
    auto relativePath = std::filesystem::relative(path, s_AssetPath);
```

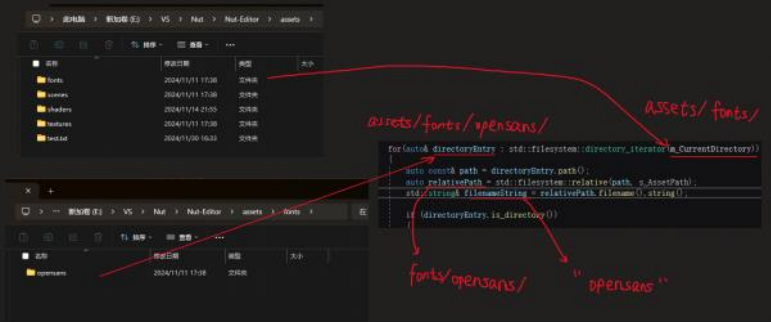
如果 `s_AssetPath` 是 `C:\Projects\MyGame\Assets`, `path` 是 `C:\Projects\MyGame\Assets\Models\Character.obj`. 那么 `std::filesystem::relative(path, s_AssetPath)` 会返回 `Models\Character.obj`, 这是 `path` 相对于 `s_AssetPath` 的相对路径.

》》操作图示：

第一次循环：



第二次循环:



))) "/" 运算符重载

Eg.
"m_CurrentDirectory /= path.filename();"

/= 运算符的重载

概念:
在 C++17 的 std::filesystem::path 中, /= 运算符是被重载的, 用于拼接路径. 其功能是将路径对象 path 中的部分与左侧的路径进行合并.

使用要求:
m_CurrentDirectory 是一个表示当前目录的路径, 通常是一个 std::filesystem::path 类型的对象. path.filename() 返回的是 path 对象中的文件名部分, 且其类型也是 std::filesystem::path.

示例说明:
假设

m_CurrentDirectory	C:\Projects\MyGame\Assets.
path	C:\Projects\MyGame\Assets\Models\Character.obj.
path.filename()	Character.obj.

那么, m_CurrentDirectory /= path.filename(); 的结果会是 m_CurrentDirectory 等于 C:\Projects\MyGame\Assets\Character.obj