HW2

打分细则

打分细则：

1. 基础：[60 分] 修改 color_grading.frag 中的代码，正确实现 ColorGrading 功能并成功编译运行。（参照代码框架说明 6 和 7）

2. 基础：[20 分] 在达成基础的前提下成功导入个性化的 LUT 图替换原 LUT 图。 （参照代码框架说明 9）

3. 提高：[20 分] 添加一个新的 Pass 实现某个自己感兴趣的后处理效果。(参 照代码框架说明 9）

基础题1

1. 首先引擎中有一个 PRenderPassBase 基类，它定义了每一个渲染功能所需的 基本步骤；

```
50  class PRenderPassBase
51  {
52  public:
53      struct FrameBufferAttachment
54      {
55          VkImage        image;
56          VkDeviceMemory mem;
57          VkImageView    view;
58          VkFormat       format;
59      };
60
61      struct Framebuffer
62      {
63          int           width;
64          int           height;
65          VkFramebuffer framebuffer;
66          VkRenderPass  render_pass;
67
68          std::vector<FrameBufferAttachment> attachments;
69      };
70
71      struct Descriptor
72      {
```

而对于一个具体的渲染功能，则另定义一个子类继承于此基类。

与 实现 ColorGrading 相关的所有 C++代码写在一个 ColorGradingPass 子类中， 并大部分位于 engine\source\runtime\function\render\passes\color_grading_pass.cpp 文件中。同学们对这部分代码稍作阅读即可， 暂不要求掌握。

2. 程序初始化时，会调用 ColorGradingPass::initialize 进行初始化，主要是定义实现此功能需要给 Vulkan 用到的各类设置、参数和资源等。

3. 窗口大小改变时，会调用 ColorGradingPass::updateAfterFramebufferRecreate 进行同步，主要是将新创建的 framebuffer 同步到当前 ColorGradingPass 的 DescriptorSet。
4. 渲染循环中，当执行到 ColorGrading 的渲染时，会调用 ColorGradingPass::draw 进行绘制。目前它是渲染循环的最后一步，在它之前 的绘制步骤已经将这一帧场景的物理渲染、光照阴影、天空盒绘制以及 tonemapping 执行完毕。
5. 调用 PColorGradingPass::draw 进行绘制时，会绑定好先前初始化好的 ColorGrading Pipeline 以指导 GPU 进行渲染运算。Pipeline 中就包括设定好 的 fragment shader。本次作业我们需要补充的 color_grading.frag 就是 fragment shader 的源代码。
6. 在 color_grading.frag 中，我们将对单个像素进行处理。已提供的数据和资源有：像素的原颜色 in_color，以及 2D 贴图采样器 color_grading_lut_texture_sampler。其中，in_color 已经在上一个 Tone Mapping Pass 中转换到了 SRGB 空间。同学们需要用它们算出像素在经过 ColorGrading 后的新颜色 out_color。可能需要用到的函数有：
    a. 获得 2D 贴图大小：highp ivec2 lut_tex_size = textureSize (uniform sampler2D sampler, 0);
    b. 获取 in_color 的值 ： highp vec4 color=subpassLoad(in_color).rgba;
    c. 根据位置采样 2D 贴图：highp vec4 texture (uniform sampler2D sampler, highp vec2 pos);
7. 我们目前使用的是 Linear 采样器。你可能需要在 color_grading.frag 中模拟采样器的行为。
8. 我们已为你预先提供了 7 张 LUT 图。你应当修改 engine/asset/global/rendering.global.json 中的以下资源的路径，来使用你 自己 LUT 图。

lut图采样原理：
示例给出的lut图是这样的：

有16*16列 有16行

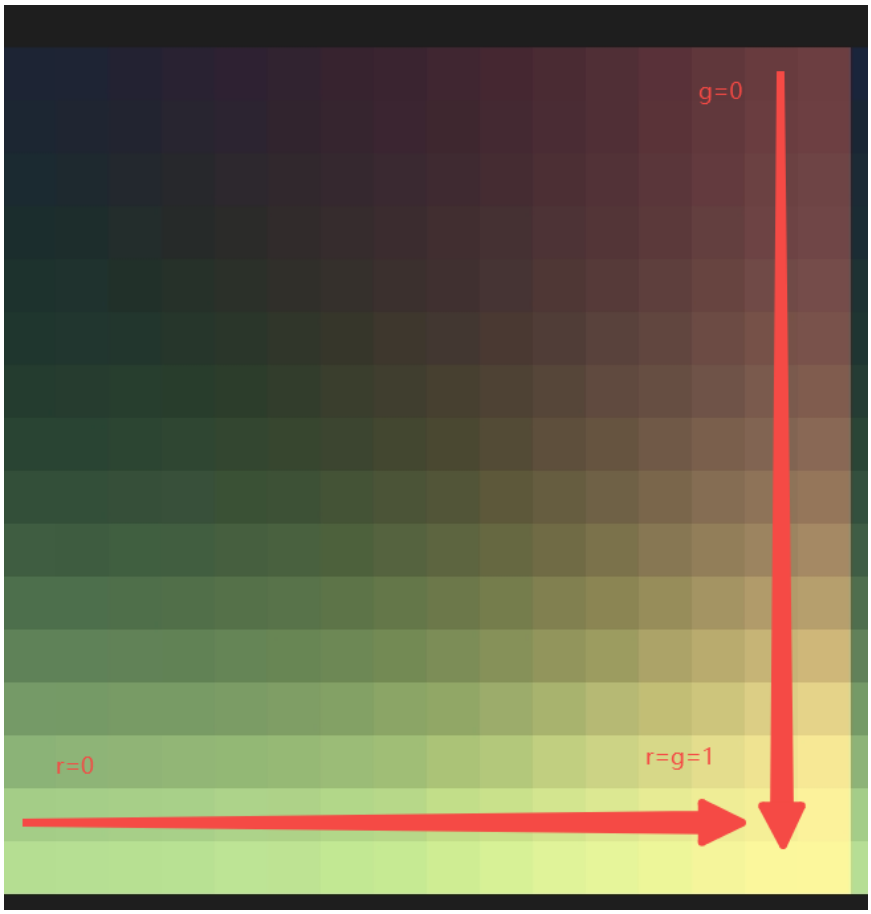
若将0-1的r值映射到0-15行上

则r应当乘以15

0的r值对应第0行

0.1的r值对应第16*0.1=1.6 floor(1.6)=1行

1的r值对应第



不知道为什么，我做这个作业就是会有很尖锐的纹路，即使是直接复制别人的代码到我的电脑上运行也会有纹路。但是别人的截图明明没有纹路。

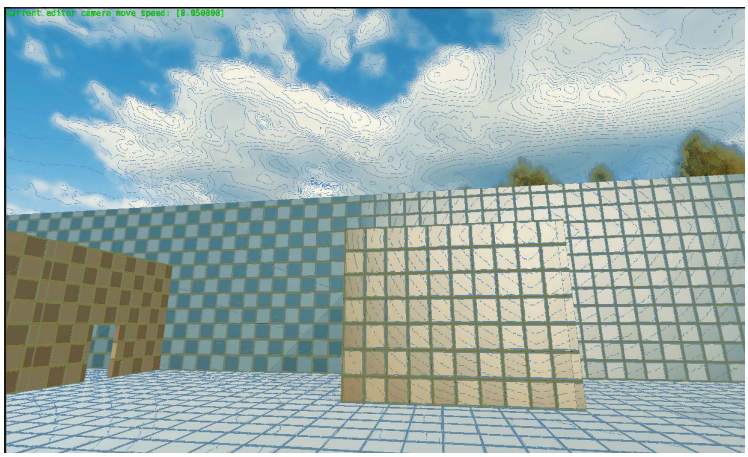
GAMES104/PA02 at master · kyriewxcode/GAMES104

homework2的shader的效果问题 · Issue #1 · kyriewxcode/GAMES104

可能性1：我少写了什么代码，但是别人的报告里也没说他们改了其他的代码啊，为什么我就不行

可能性2：是这个引擎渲染管线的bug

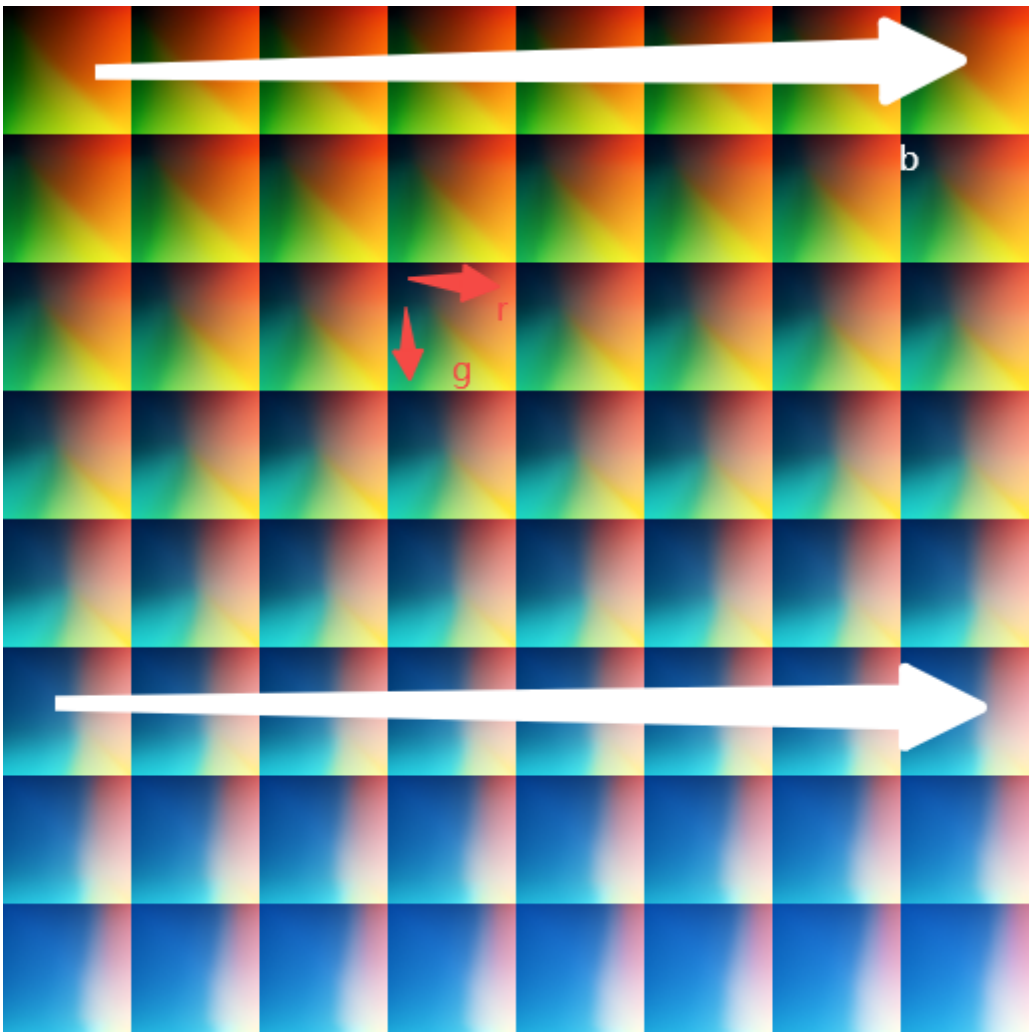
可能性3：我的电脑的什么线性采样有问题之类的

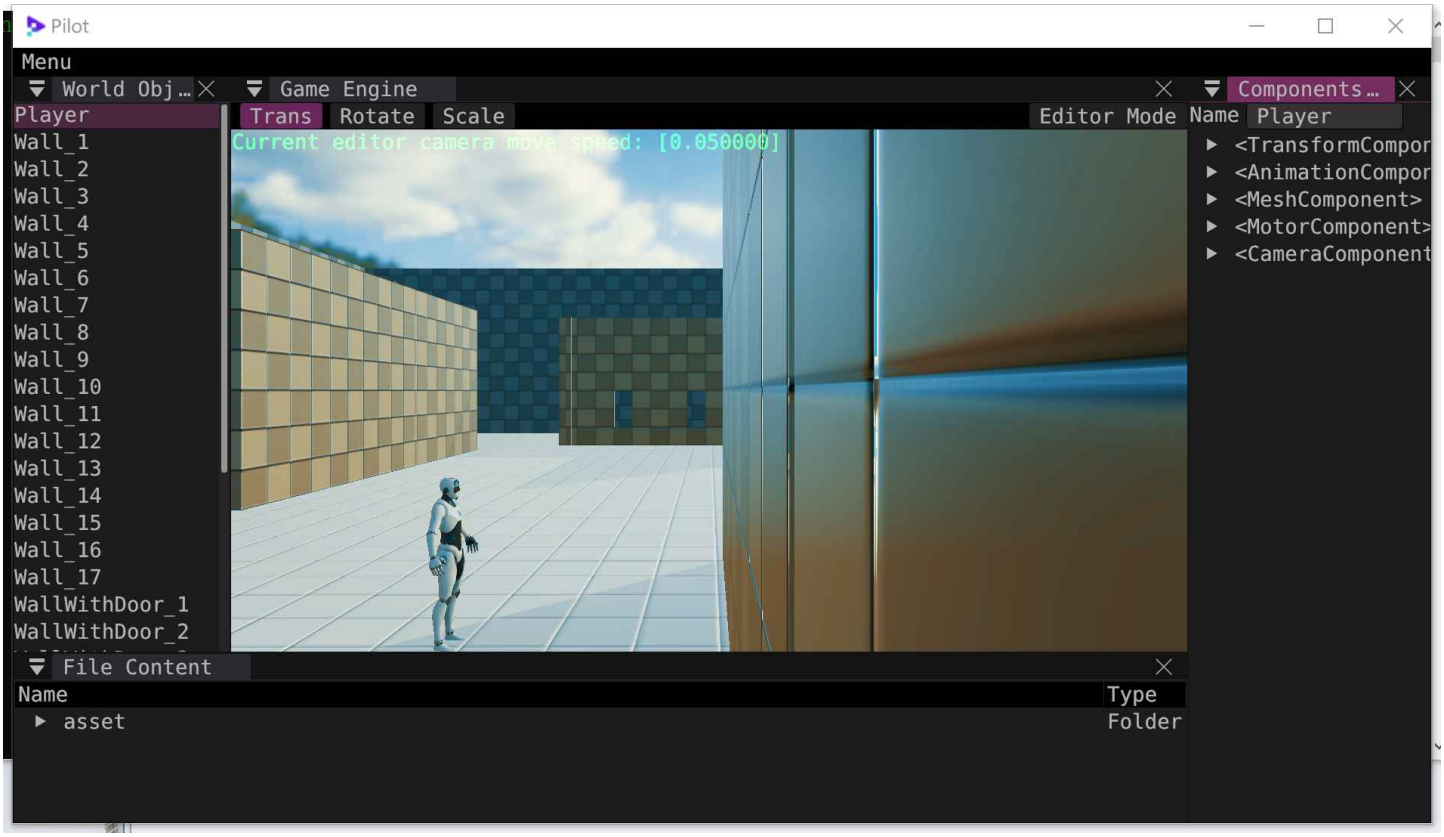


我找到了问题。我是直接在main分支上修改了shader代码。在homework2分支上修改代码就不会有此问题。


Pilot
Menu
World Obj…
Player
Wall_1
Wall_2
Wall_3
Wall_4
Wall_5
Wall_6
Wall_7
Wall_8
Wall_9
Wall_10
Wall_11
Wall_12
Wall_13
Wall_14
Wall_15
Wall_16
Wall_17
WallWithDoor_1
WallWithDoor_2
WallWithDoor_3
WallWithDoor_4
WallWithDoor_5
Game Engine
Trans
Rotate
Scale
Editor Mode
Current editor camera move speed: [0.050000]
Components…
Name
Player
<TransformCompor
<AnimationCompor
<MeshComponent>
<MotorComponent>
<CameraComponent
File Content
Name
Type
asset
Folder

基础题2


b
r
g



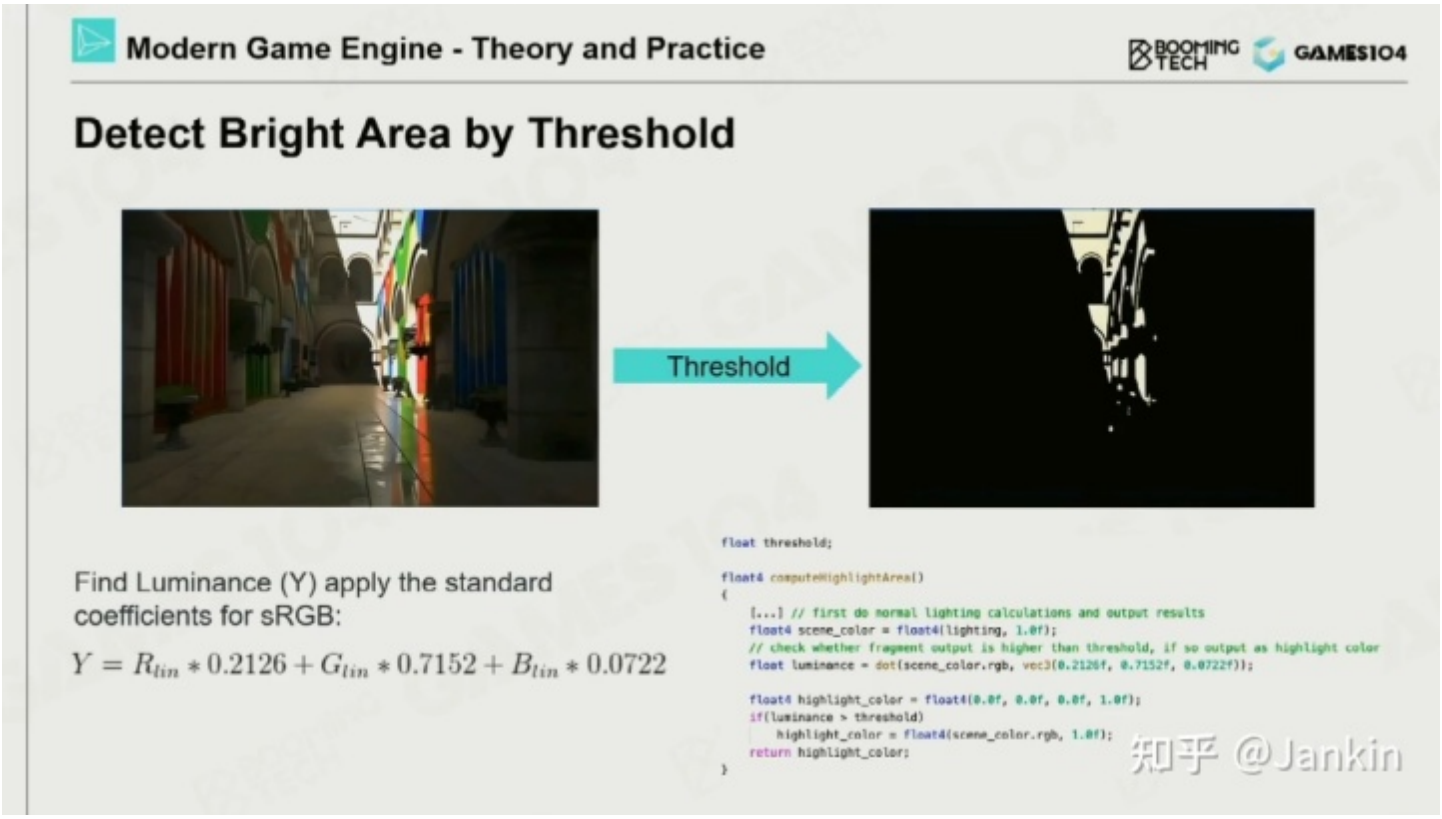
附加题（未完成）

在达成基础的前提下成功导入个性化的 LUT 图替换原 LUT 图。

（提高项，可选）添加一个新的 Pass 实现某个自己感兴趣的后处理效果。

实现bloom

bloom原理



首先把高亮的部分给取出来，然后我们可以算这个亮度超过某个阈值的时候，就能够被选取出来；

取得是颜色，不仅仅是强度；

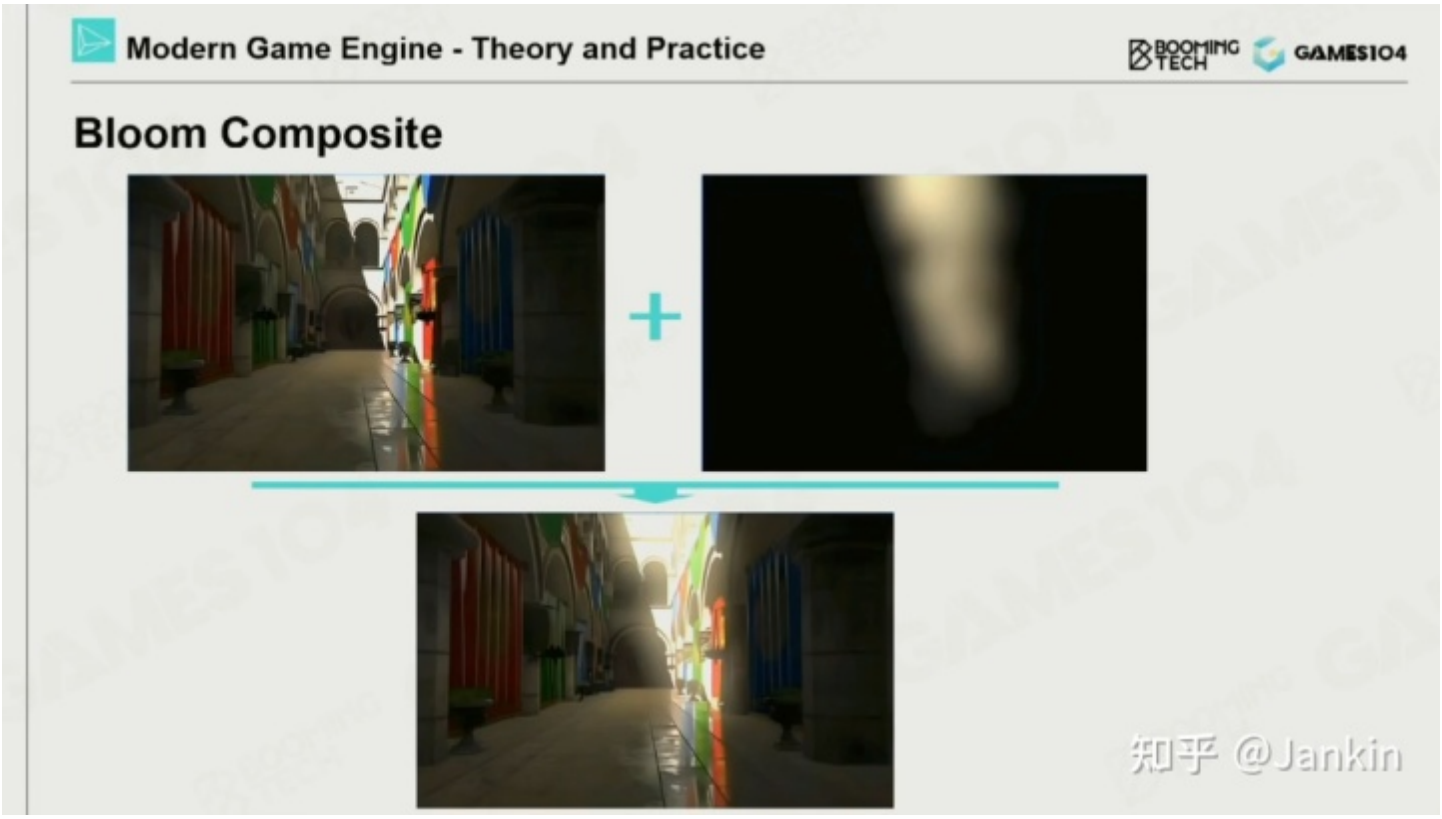
取出这个东西之后，就给模糊一下，用高斯，遇事不决就那Hack；

其实这个高斯 还挺简单的，先横着取一遍，然后在纵着取一遍；

但是这个范围是不够大的；

就是对图像不断进行降采样，然后在最低的一节高斯一下，最后再给他们通过一些权重给加在一起；

我们能得到光被晕的很开的效果；



叠加上去之后就能看到一个非常好的效果；

附加题代码结构

我们的渲染循环是 PVulkanManager::renderFrame。其中 会调用 PMainCameraPass::draw 进行延迟渲染。

你可以模仿 color_grading.cpp 和 tone_mapping.cpp 添加一个自己感兴趣的 Pass。

color_grading.cpp的代码结构

```
void PColorGradingPass::initialize(VkRenderPass render_pass, VkImageView inp
{
    _framebuffer.render_pass = render_pass;
    setupDescriptorSetLayout();// 设置一些asset的大小
    setupPipelines();// 设置pipeline的一些选项以及vertex shader和fragment shad
    setupDescriptorSet();// allocate descriptor?
    updateAfterFramebufferRecreate(input_attachment);
}
```

```
void PColorGradingPass::draw()
{
    if (m_render_config._enable_debug_untils_label)
    {
        VkDebugUtilsLabelEXT label_info = {
            VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT, NULL, "Color Grading",
        m_p_vulkan_context->_vkCmdBeginDebugUtilsLabelEXT(m_command_info._cu
    }

    m_p_vulkan_context->_vkCmdBindPipeline(
        m_command_info._current_command_buffer, VK_PIPELINE_BIND_POINT_GRAPH
    m_p_vulkan_context->_vkCmdSetViewport(m_command_info._current_command_bu
    m_p_vulkan_context->_vkCmdSetScissor(m_command_info._current_command_buf
    m_p_vulkan_context->_vkCmdBindDescriptorSets(m_command_info._current_com
                                                 VK_PIPELINE_BIND_POINT_GRAP
                                                 _render_pipelines[0].layout
                                                 0,
                                                 1,
                                                 &_descriptor_infos[0].descr
                                                 0,
                                                 NULL);

    vkCmdDraw(m_command_info._current_command_buffer, 3, 1, 0, 0);

    if (m_render_config._enable_debug_untils_label)
    {
        m_p_vulkan_context->_vkCmdEndDebugUtilsLabelEXT(m_command_info._curr
    }
}
```

其他代码结构

你可能需要调整 PMainCameraPass::setupRenderPass（initialized的时候会被调用） 添加一个新的Vulkan Subpass。

```
    enum
    {
        _main_camera_subpass_basepass = 0,
        _main_camera_subpass_deferred_lighting,
        _main_camera_subpass_forward_lighting,
        _main_camera_subpass_tone_mapping,
        _main_camera_subpass_color_grading,
        _main_camera_subpass_ui,
        _main_camera_subpass_combine_ui,
        _main_camera_subpass_count
    };
```

```
        VkAttachmentReference color_grading_pass_input_attachment_reference {};
        color_grading_pass_input_attachment_reference.attachment =
            &backup_even_color_attachment_description - attachments;
        color_grading_pass_input_attachment_reference.layout = VK_IMAGE_LAYOUT_S

        VkAttachmentReference color_grading_pass_color_attachment_reference {};
        color_grading_pass_color_attachment_reference.attachment =
            &backup_odd_color_attachment_description - attachments;
        color_grading_pass_color_attachment_reference.layout = VK_IMAGE_LAYOUT_C

        VkSubpassDescription& color_grading_pass   = subpasses[_main_camera_subp
        color_grading_pass.pipelineBindPoint       = VK_PIPELINE_BIND_POINT_GRAP
        color_grading_pass.inputAttachmentCount    = 1;
        color_grading_pass.pInputAttachments       = &color_grading_pass_input_a
        color_grading_pass.colorAttachmentCount    = 1;
        color_grading_pass.pColorAttachments       = &color_grading_pass_color_a
        color_grading_pass.pDepthStencilAttachment = NULL;
        color_grading_pass.preserveAttachmentCount = 0;
        color_grading_pass.pPreserveAttachments    = NULL;
```

同时，你可能需要调整 PVulkanManager::initializeDescriptorPool 增加相关 Descriptor 的数量。

```
bool Pilot::PVulkanManager::initializeDescriptorPool()
{
    // Since DescriptorSet should be treated as asset in Vulkan, DescriptorPool
    // should be big enough, and thus we can sub-allocate DescriptorSet from
    // DescriptorPool merely as we sub-allocate Buffer/Image from DeviceMemory.

    VkDescriptorPoolSize pool_sizes[5];
    pool_sizes[0].type            = VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC;
```

```
    pool_sizes[0].descriptorCount = 3 + 2 + 2 + 2 + 1 + 1 + 3 + 3;
    pool_sizes[1].type            = VK_DESCRIPTOR_TYPE_STORAGE_BUFFER;
    pool_sizes[1].descriptorCount = 1 + 1 + 1 * m_max_vertex_blending_mesh_count
    pool_sizes[2].type            = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    pool_sizes[2].descriptorCount = 1 * m_max_material_count;
    pool_sizes[3].type            = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
    pool_sizes[3].descriptorCount = 3 + 5 * m_max_material_count + 1 + 1 ; // In
    pool_sizes[4].type            = VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT;
    pool_sizes[4].descriptorCount = 4 + 1 + 1 + 2;

    VkDescriptorPoolCreateInfo pool_info {};
    pool_info.sType         = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
    pool_info.poolSizeCount = sizeof(pool_sizes) / sizeof(pool_sizes[0]);
    pool_info.pPoolSizes    = pool_sizes;
    pool_info.maxSets =
        1 + 1 + 1 + m_max_material_count + m_max_vertex_blending_mesh_count + 1
    pool_info.flags = 0U;

    if (vkCreateDescriptorPool(m_vulkan_context._device, &pool_info, nullptr, &m
    {
        throw std::runtime_error("create descriptor pool");
    }

    return true;
}

```

我不明白engine\shader\generated\cpp\color_grading_frag.h是怎么generate出来的

然后我发现只要在visual studio的sln里面点generate project就有了

我感觉可能要去学vulkan

如何正确的入门Vulkan? - 知乎

但是vulkan看起来好复杂

作业这边不定义新的渲染管线，直接在原来的管线上改可以吗?

但是有一个问题 就是要模糊的话是要取周围的平均值的

不改渲染管线的话怎么能取到周围的像素值呢

所以还是要改渲染管线?

但是问题在于 怎么修改渲染管线才能让frag能获得周围的像素值呢 这很奇怪