Games104_homework2_report

Online version: https://nbqmlgi3yg.feishu.cn/docs/doccnnkaIDESWbtgcZuONs3BU6c

The content in the online version may be more updated.

a> ColorGrading Code

Shader code:

OpenGL Shading Language

```
#version 310 es

#extension GL_GOOGLE_include_directive : enable

#include "constants.h"

layout(input_attachment_index = 0, set = 0, binding = 0) uniform highp subpassInput in_color;

layout(set = 0, binding = 1) uniform sampler2D color_grading_lut_texture_sampler;

layout(location = 0) out highp vec4 out_color;

highp vec2 lut_uv(highp float red, highp float green,
                  highp float blue_slice, highp float slice_size) {
    // u = (u_Red + u_Blue) / size, v = u_green
    // `min` fix special case for color.b = 1.0F
    return vec2((red + min(blue_slice, slice_size - 1.0F)) / slice_size, green);
}

highp float get_slice_size() {
    highp ivec2 lut_tex_size = textureSize(color_grading_lut_texture_sampler, 0);
    return float(lut_tex_size.y);
    // return 16.0F; // can be hard cored as 16 for color_grading_lut_01.png
}

highp vec4 get_lut_color(highp vec4 color, highp float slice_size) {
    // scale blue by slice_size, and get integral and fractional part
    // example: with a sampler of 16 slices
    // 5.6(scaled blue) => slice 5, slice_weight 0.6
```

```
30      highp float slice;
31      // max blue = 15.0, then rgb (0.6, g, 1.0) will be sampled from (15.6/16.0,
    g)
32      highp float slice_weight = modf(color.b * (slice_size - 1.0F), slice);
33
34      highp vec4 color_left = textureLod(color_grading_lut_texture_sampler,
35                                         lut_uv(color.r, color.g, slice, slice_siz
    e), 0.0);
36      highp vec4 color_right = textureLod(color_grading_lut_texture_sampler,
37                                          lut_uv(color.r, color.g, slice + 1.0F, s
    lice_size), 0.0);
38
39      return vec4(
40          // interpolate 1D using weight as fractional part of scaled blue
41          mix(color_left, color_right, slice_weight).rgb,
42          // using alpha from original color
43          color.a);
44  }
45
46  void main() {
47      highp vec4 color = subpassLoad(in_color).rgba;
48      highp float slice_size = get_slice_size();
49
50      out_color = get_lut_color(color, slice_size);
51  }
```

Method

The missing part of color grading pass is merely the shader, since other cpp codes are already present.

Adapted from a 2D implementation(Color grading LUT (shadertoy.com)), the shader code has been implemented as following:

Steps:

1. Find two slices and the weight by blue value (left slice <= blue <= right slice) using `modf`
2. Using blue offset + red offset as `u`, green offset as `v`
3. Get two colors from the two slices using `textureLod` with predefined sampler and `uv`
4. Mix the two colors using `mix` with the weight, and keep the alpha value

Code structure:

- Function `lut_uv` : get uv by red,green,blue_slice, and slice_size
- Function `get_lut_color` : get color by input color and slice_size
- Function `get_slice_size` : get slice_size of texture

Extra notes:

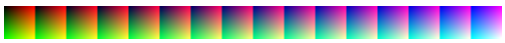
- Line 32: Blue offset should start from the left-bottom corner of the slice, thus blue should be mapped to `[0, slice_size-1]`.
- Line 17: When the last slice should be used as the left slice, the right slice should be clamped as the last slice too. `if` check is avoided by using `min`.

b> Custom LUT

Custom LUT image has been created with the method from unreal document (Using Lookup Tables (LUTs) for Color Grading | Unreal Engine Documentation)

Neutral LUT Image

Download from the mentioned doc:



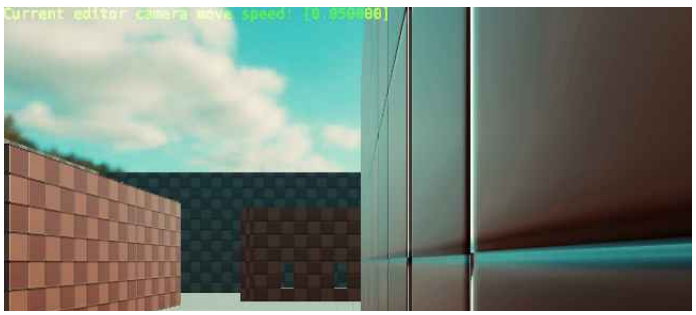
Method

1. Get a screen snapshot of display in `Pilot Engine` without color grading
2. Tune the image to desired using adjustment layers in `Photoshop`
3. Copy the layers above a neutral LUT image as background
4. Export the result to a custom LUT image (png format without transparency)
5. Add it to engine assets and change global rending config

Expected image effect

A color grading gives the feeling of early autumn:

- Left: expected
- Right: without color grading

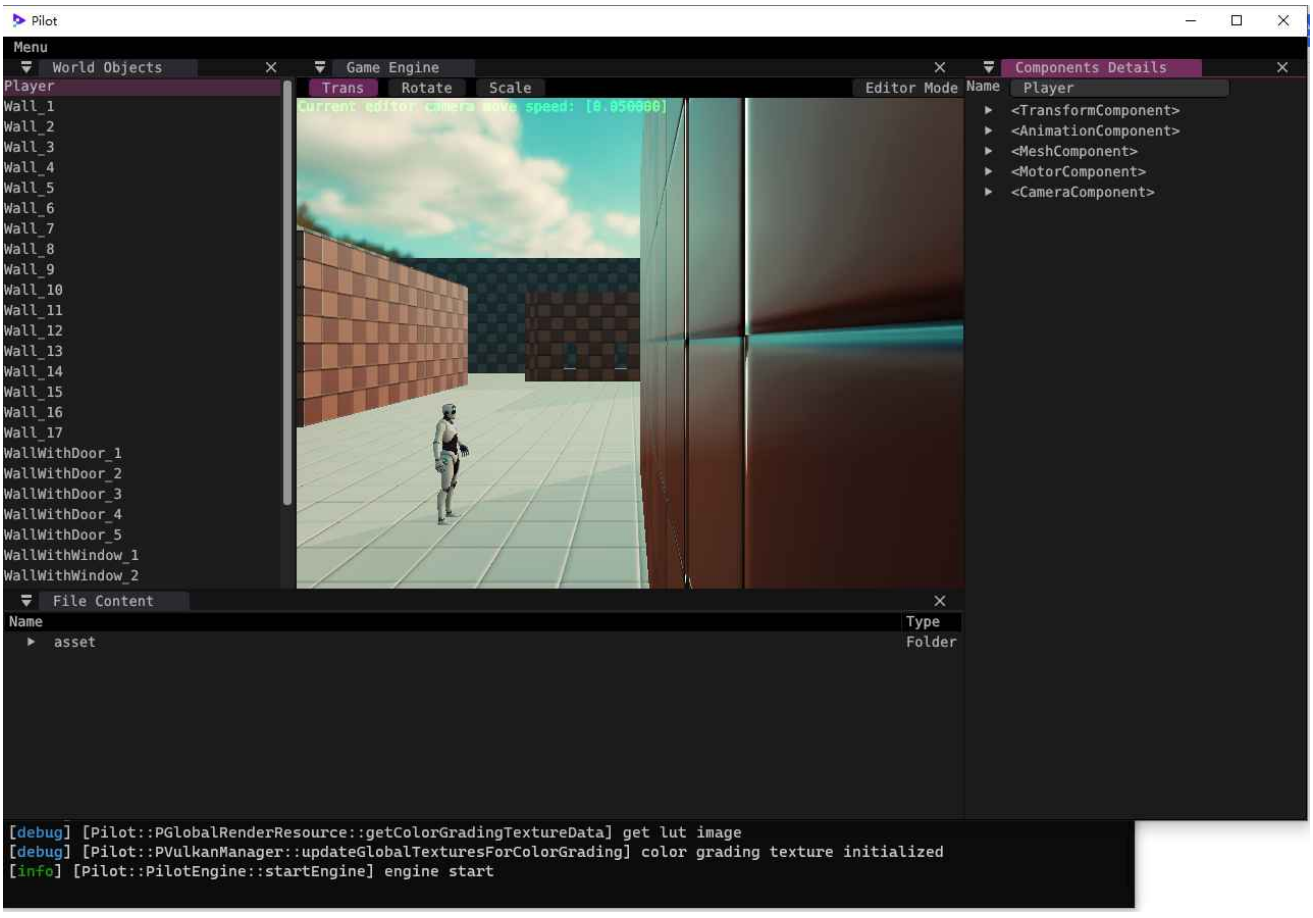



Custom LUT Image

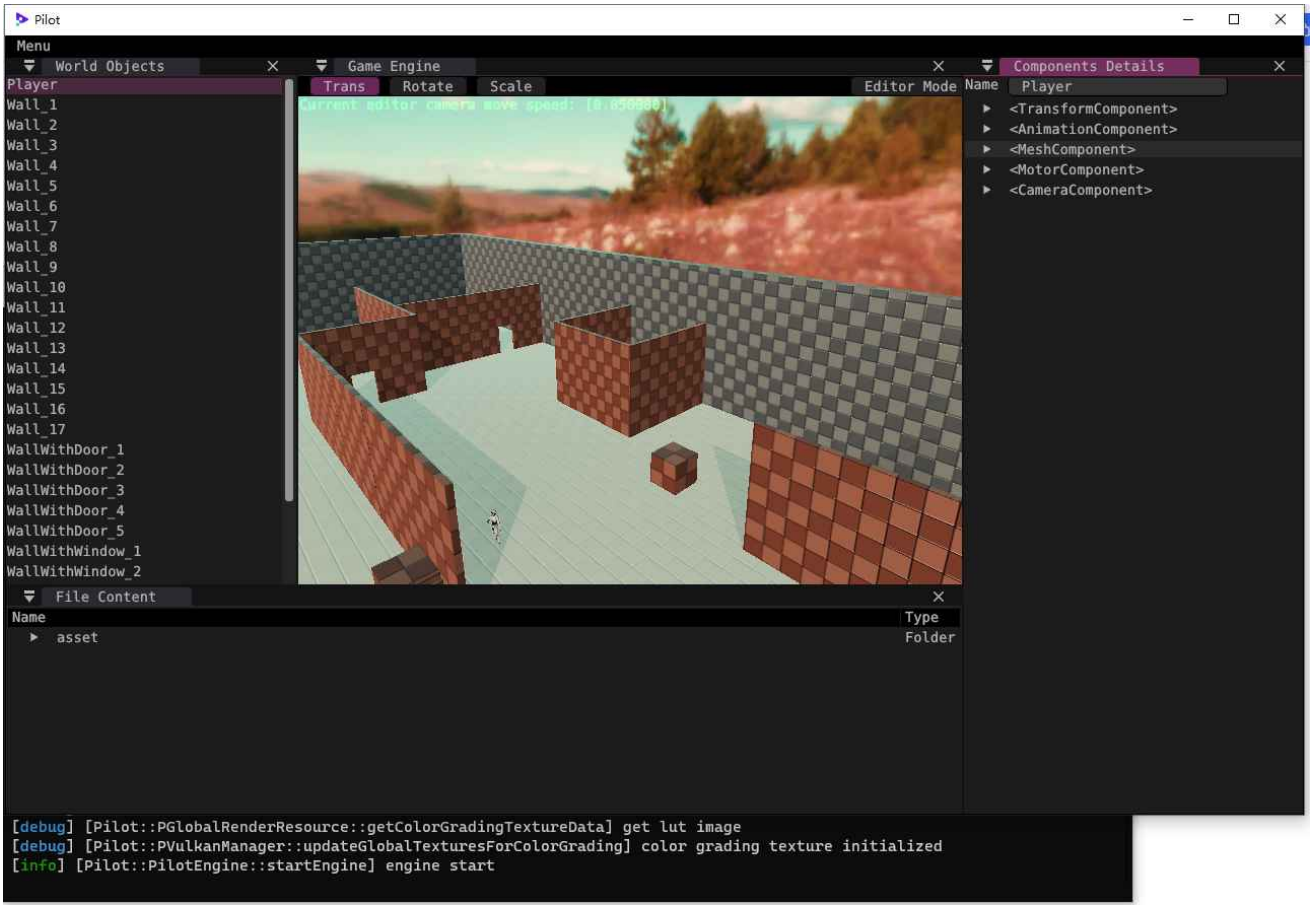
`color_grading_lut_autumn.png`

Live in `Pilot Engine`

Default display



Aerial display



c> FXAA render pass

A custom render pass for `FXAA` anti-aliasing has been implemented to meet the requirements for part c of the homework.

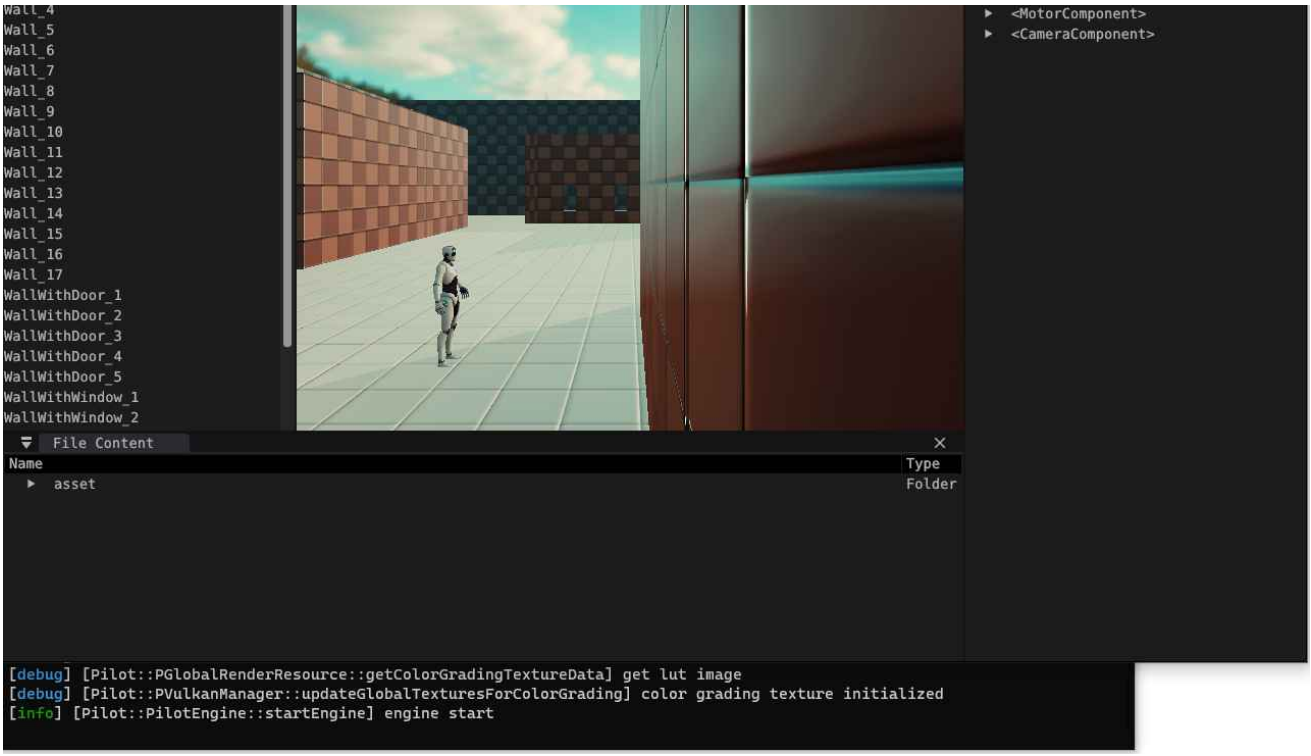
`FXAA` is an anti-aliasing technique and suitable to be implemented in post process. In this part, the implementation in `Pilot` is emphasized.

And the algorithm in shader has been adapted from established code from reference.

- (Tested) Fast Approximate Anti-Aliasing (FXAA) Demo (GLSL) | Geeks3D
- https://www.shadertoy.com/view/ls3GWS (as source for this implementation)

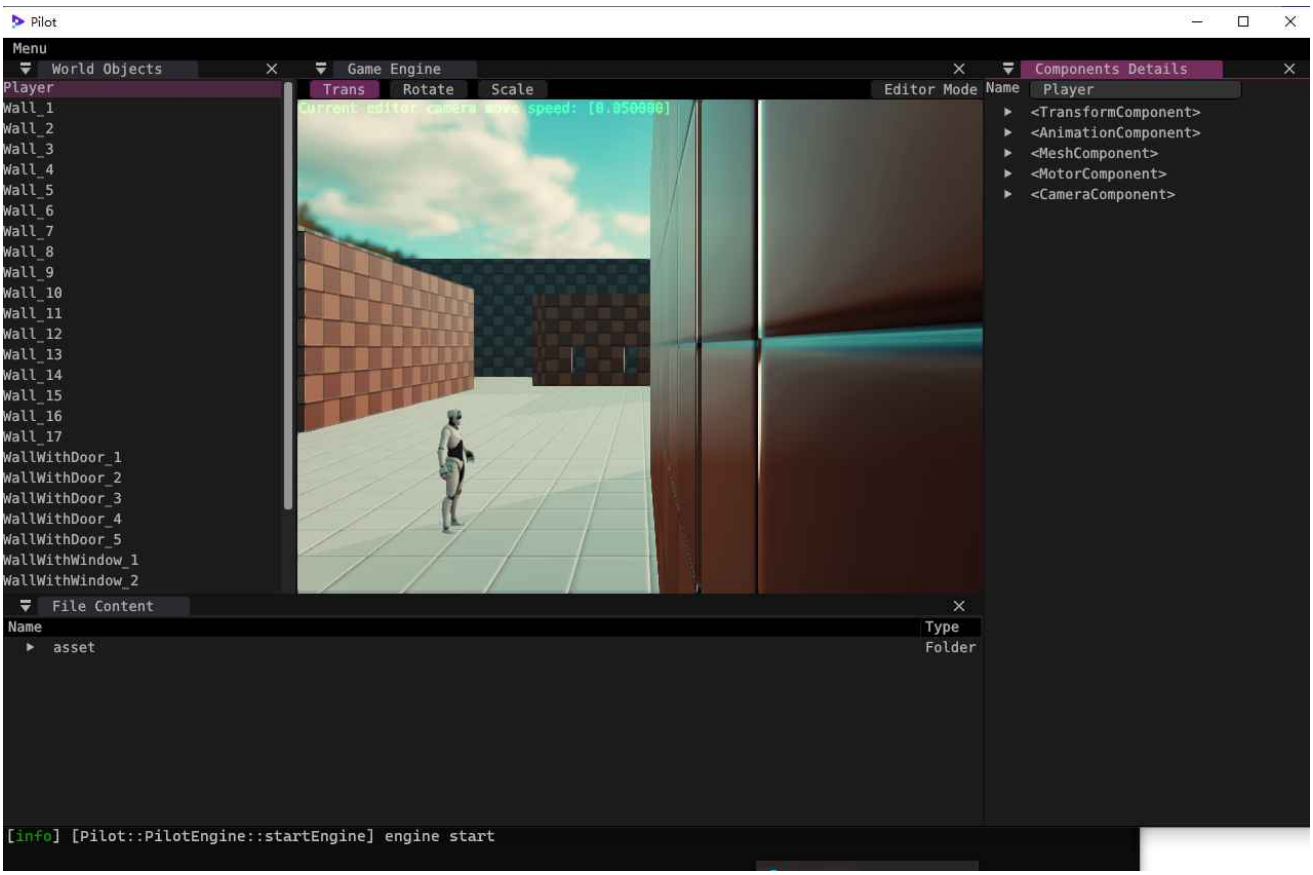
Subpass Rending Result:

FXAA Off:



FXAA On:

- Smoother: see bottom edge of the wall, outline of arms
- Blurer(unwanted effect): see hand of character

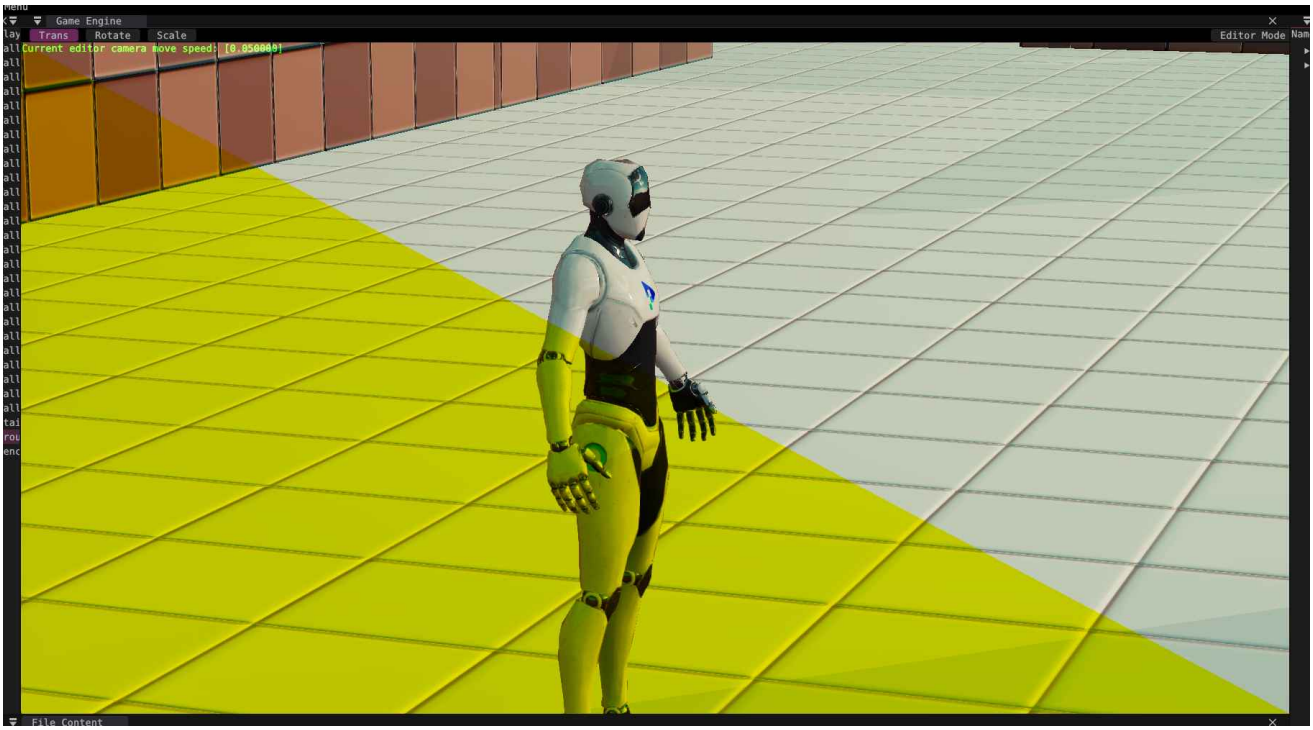


Live in Pilot Engine(Debug mode)

(to turn off debug mode: comment out all debug defs in shader)

Left: FXAA On without blue channel

Right: FXAA off



Left: FXAA On Right: FXAA off

Method

The code of the new render pass for `FXAA` has been copied and modified from tone mapping pass and color grading pass, given the fact that configuring `Vulkan` is no trivial work. Then, how to meet shader requirements for `FXAA` is focused.

Implementation requirements for `FXAA`:

The `FXAA` algorithm samples from the frame buffer. Besides regular render pass settings, the implementation needs inputs as following:

- A sampler of the rendering result from last subpass
- Resolution and offset of the game display screen in the sampler for the right `uv` in texture coordinates
- Correct `uv` of the current color

Steps:

1. Set build target to debug (noticed very late, thus hours wasted)
2. Setup the new render pass just as tone mapping and rename classes and variables, and name the pass `PAntiAliasingPass`
3. Modify main camera pass to include new subpass
4. Add sampler settings to the new subpass
5. In vert shader, add correct uv output
6. In frag shader, add debug code settings supporting 2 split screens for comparing the result and the origin
7. Achieve sampling at framebuffer with hard cored screen resolution
8. Add `ResolutionData`, and use `push constants` to pass it to shader
9. Adapt `FXAA` alogrithm in frag shader
10. Remove any `Vulkan` validation errors

Coding notes

In the long way to finishing these steps, I have encountered may pitfalls. Here are the notes for them.

- Even odd in `PMainCameraPass::setupRenderPass`: Wrong sequences will leave the rendering visually skipped

- Screen offset: Image from buffer has the extent of the whole window. Sampling should be adjusted with screen offsets. And these offsets are not available to shaders without passing proper data buffer or constants in advance.
- Sampling of input attachment directly: It is not allowed. The project will not be built unless a previous successful build is already available.
- Sampling layout creation: Wrong flags generate invalid sampling or validation errors

Appendix - Codes Extracts

Cpp Codes:

engine/source/runtime/function/render/include/render/vulkan_manager/vulkan_passes.h

C++

```
    // Add resolution data
    struct ResolutionData {
        glm::vec4 screen_resolution;
        glm::vec4 editor_screen_resolution;
    };

    // Add a new pass class
    class PAntiAliasingPass : public PRenderPassBase
    {
    public:
        void initialize(VkRenderPass render_pass, VkImageView input_attachment);
        // update resolution data using push constant
        void updateResolutionData();
        void draw();

        void updateAfterFramebufferRecreate(VkImageView input_attachment);

    private:
        void setupDescriptorSetLayout();
        void setupPipelines();
        void setupDescriptorSet();
    };

    // >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>omit

    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_anti_aliasing, // add a new enum
        _main_camera_subpass_ui,
        _main_camera_subpass_combine_ui,
        _main_camera_subpass_count
    };

    class PMainCameraPass : public PRenderPassBase
    {
    // >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>omit
            // add a new pass arg to draw
        void draw(PAntiAliasingPass& anti_aliasing_pass,
                PColorGradingPass& color_grading_pass,
                  PToneMappingPass&  tone_mapping_pass,
                  PUIPass&           ui_pass,
                  PCombineUIPass&    combine_ui_pass,
                  uint32_t           current_swapchain_image_index,
                  void*              ui_state);
     // >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>omit
      }
```

engine/source/runtime/function/render/source/vulkan_manager/passes/anti_aliasing.cpp

(indeference routine omitted)

C++

```
    void PAntiAliasingPass::setupDescriptorSetLayout()
    {
        _descriptor_infos.resize(1);

        VkDescriptorSetLayoutBinding post_process_global_layout_bindings[2] =
 {};

        VkDescriptorSetLayoutBinding& post_process_global_layout_input_attachmen
t_binding =
            post_process_global_layout_bindings[0];
        post_process_global_layout_input_attachment_binding.binding         = 0;
        post_process_global_layout_input_attachment_binding.descriptorType  = VK
_DESCRIPTOR_TYPE_INPUT_ATTACHMENT;
        post_process_global_layout_input_attachment_binding.descriptorCount = 1;
        post_process_global_layout_input_attachment_binding.stageFlags      = VK
_SHADER_STAGE_FRAGMENT_BIT;

        VkDescriptorSetLayoutBinding& post_process_global_layout_image_sampler_b
inding =
            post_process_global_layout_bindings[1];
```

```
        post_process_global_layout_image_sampler_binding.binding         = 1;
        post_process_global_layout_image_sampler_binding.descriptorType  = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
        post_process_global_layout_image_sampler_binding.descriptorCount = 1;
        post_process_global_layout_image_sampler_binding.stageFlags      = VK_SHADER_STAGE_FRAGMENT_BIT;

        VkDescriptorSetLayoutCreateInfo post_process_global_layout_create_info;
        post_process_global_layout_create_info.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
        post_process_global_layout_create_info.pNext = NULL;
        post_process_global_layout_create_info.flags = 0;
        post_process_global_layout_create_info.bindingCount =
            sizeof(post_process_global_layout_bindings) / sizeof(post_process_global_layout_bindings[0]);
        post_process_global_layout_create_info.pBindings = post_process_global_layout_bindings;

        if (VK_SUCCESS != vkCreateDescriptorSetLayout(m_p_vulkan_context->_device,
                                                      &post_process_global_layout_create_info,
                                                      NULL,
                                                      &_descriptor_infos[0].layout))
        {
            throw std::runtime_error("create post process global layout");
        }
    }

        void PAntiAliasingPass::setupPipelines()
    {
        _render_pipelines.resize(1);

        VkDescriptorSetLayout      descriptorset_layouts[1] = {_descriptor_infos[0].layout};
        VkPipelineLayoutCreateInfo pipeline_layout_create_info {};
        pipeline_layout_create_info.sType          = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
        pipeline_layout_create_info.setLayoutCount = 1;
        pipeline_layout_create_info.pSetLayouts    = descriptorset_layouts;

        // per https://vkguide.dev/docs/chapter-3/push_constants/
        // using push constants
        // setup push constants
        VkPushConstantRange push_constant;
        // this push constant range starts at the beginning
        push_constant.offset = 0;
        // this push constant range takes up the size of the struct
```

```
        push_constant.size = sizeof(ResolutionData);
        // beware the BIT
        push_constant.stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT;
        // add the push constant to pipeline
        pipeline_layout_create_info.pPushConstantRanges  = &push_constant;
        pipeline_layout_create_info.pushConstantRangeCount = 1;
     // >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>omit
      }

    void PAntiAliasingPass::updateAfterFramebufferRecreate(VkImageView input_attachment)
    {

        VkDescriptorImageInfo post_process_per_frame_input_attachment_info = {};
        post_process_per_frame_input_attachment_info.sampler =
            PVulkanUtil::getOrCreateNearestSampler(m_p_vulkan_context->_physical_device, m_p_vulkan_context->_device);
        post_process_per_frame_input_attachment_info.imageView   = input_attachment;
        post_process_per_frame_input_attachment_info.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;

        // use input attachment as sampler image
        VkDescriptorImageInfo post_process_global_layout_image_sampler_image_info = {};
        post_process_global_layout_image_sampler_image_info.sampler =
            PVulkanUtil::getOrCreateLinearSampler(m_p_vulkan_context->_physical_device, m_p_vulkan_context->_device);
        post_process_global_layout_image_sampler_image_info.imageView   = input_attachment;
        post_process_global_layout_image_sampler_image_info.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;

        VkWriteDescriptorSet post_process_descriptor_writes_info[2];

        VkWriteDescriptorSet& post_process_descriptor_input_attachment_write_info =
            post_process_descriptor_writes_info[0];
        post_process_descriptor_input_attachment_write_info.sType           = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
        post_process_descriptor_input_attachment_write_info.pNext           = NULL;
        post_process_descriptor_input_attachment_write_info.dstSet          = _descriptor_infos[0].descriptor_set;
        post_process_descriptor_input_attachment_write_info.dstBinding      = 0;
        post_process_descriptor_input_attachment_write_info.dstArrayElement = 0;
        post_process_descriptor_input_attachment_write_info.descriptorType  = VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT;
```

```
90          post_process_descriptor_input_attachment_write_info.descriptorCount = 1;
91          post_process_descriptor_input_attachment_write_info.pImageInfo = &post_p
    rocess_per_frame_input_attachment_info;
92
93          VkWriteDescriptorSet& post_process_descriptor_LUT_write_info = post_proc
    ess_descriptor_writes_info[1];
94          post_process_descriptor_LUT_write_info.sType                = VK_STRUCT
    URE_TYPE_WRITE_DESCRIPTOR_SET;
95          post_process_descriptor_LUT_write_info.pNext                = NULL;
96          post_process_descriptor_LUT_write_info.dstSet               = _descript
    or_infos[0].descriptor_set;
97          // beware wrong dstBinding
98          post_process_descriptor_LUT_write_info.dstBinding      = 1;
99          post_process_descriptor_LUT_write_info.dstArrayElement = 0;
100         post_process_descriptor_LUT_write_info.descriptorType  = VK_DESCRIPTOR_T
    YPE_COMBINED_IMAGE_SAMPLER;
101         post_process_descriptor_LUT_write_info.descriptorCount = 1;
102         post_process_descriptor_LUT_write_info.pImageInfo      = &post_process_g
    lobal_layout_image_sampler_image_info;
103
104         vkUpdateDescriptorSets(m_p_vulkan_context->_device,
105                                sizeof(post_process_descriptor_writes_info) /
106                                    sizeof(post_process_descriptor_writes_info[0
    ]),
107                                post_process_descriptor_writes_info,
108                                0,
109                                NULL);
110     }
111
112     void PAntiAliasingPass::updateResolutionData()
113     {
114         // create ResolutionData
115         ResolutionData resolution_data = {
116             glm::vec4(
117                 0.0F, 0.0F, m_p_vulkan_context->_swapchain_extent.width, m_p_vul
    kan_context->_swapchain_extent.height),
118             glm::vec4(m_command_info._viewport.x,
119                       m_command_info._viewport.y,
120                       m_command_info._viewport.width,
121                       m_command_info._viewport.height)};
122
123         vkCmdPushConstants(m_command_info._current_command_buffer,
124                            _render_pipelines[0].layout,
125                            VK_SHADER_STAGE_FRAGMENT_BIT,
126                            0,
127                            sizeof(ResolutionData),
128                            &resolution_data);
129     }
```

```
130
```

engine/source/runtime/function/render/source/vulkan_manager/passes/main_camera.cpp

C++

```
1     void PMainCameraPass::setupRenderPass()
2     {
3     // >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>omit
4             // anti aliasing pass
5         VkAttachmentReference anti_aliasing_pass_input_attachment_reference {};
6         anti_aliasing_pass_input_attachment_reference.attachment =
7             &backup_odd_color_attachment_description - attachments;
8         anti_aliasing_pass_input_attachment_reference.layout = VK_IMAGE_LAYOUT_S
  HADER_READ_ONLY_OPTIMAL;
9
10        VkAttachmentReference anti_aliasing_pass_color_attachment_reference {};
11        anti_aliasing_pass_color_attachment_reference.attachment =
12            &backup_even_color_attachment_description - attachments;
13        anti_aliasing_pass_color_attachment_reference.layout = VK_IMAGE_LAYOUT_C
  OLOR_ATTACHMENT_OPTIMAL;
14
15        VkSubpassDescription& anti_aliasing_pass   = subpasses[_main_camera_subp
  ass_anti_aliasing];
16        anti_aliasing_pass.pipelineBindPoint       = VK_PIPELINE_BIND_POINT_GRAP
  HICS;
17        anti_aliasing_pass.inputAttachmentCount    = 1;
18        anti_aliasing_pass.pInputAttachments       = &anti_aliasing_pass_input_a
  ttachment_reference;
19        anti_aliasing_pass.colorAttachmentCount    = 1;
20        anti_aliasing_pass.pColorAttachments       = &anti_aliasing_pass_color_a
  ttachment_reference;
21        anti_aliasing_pass.pDepthStencilAttachment = NULL;
22        anti_aliasing_pass.preserveAttachmentCount = 0;
23        anti_aliasing_pass.pPreserveAttachments    = NULL;
24
25        // beware change the odd and even attachments accordingly
26        VkAttachmentReference ui_pass_color_attachment_reference {};
27        ui_pass_color_attachment_reference.attachment = &backup_odd_color_attach
  ment_description - attachments;
28        ui_pass_color_attachment_reference.layout     = VK_IMAGE_LAYOUT_COLOR_AT
  TACHMENT_OPTIMAL;
29
30        uint32_t ui_pass_preserve_attachment = &backup_even_color_attachment_des
  cription - attachments;
31     // >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>omit
32             VkSubpassDependency& anti_aliasing_pass_depend_on_color_grading_pas
  s = dependencies[5];
```

```
        anti_aliasing_pass_depend_on_color_grading_pass.srcSubpass      = _main_camera_subpass_color_grading;
        anti_aliasing_pass_depend_on_color_grading_pass.dstSubpass      = _main_camera_subpass_anti_aliasing;
        anti_aliasing_pass_depend_on_color_grading_pass.srcStageMask =
            VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
        anti_aliasing_pass_depend_on_color_grading_pass.dstStageMask =
            VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
        anti_aliasing_pass_depend_on_color_grading_pass.srcAccessMask =
            VK_ACCESS_SHADER_WRITE_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
        anti_aliasing_pass_depend_on_color_grading_pass.dstAccessMask =
            VK_ACCESS_SHADER_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_READ_BIT;
        anti_aliasing_pass_depend_on_color_grading_pass.dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT;

        // anti aliasing
        VkSubpassDependency& ui_pass_depend_on_anti_aliasing_pass = dependencies[6];
        ui_pass_depend_on_anti_aliasing_pass.srcSubpass      = _main_camera_subpass_anti_aliasing;
        ui_pass_depend_on_anti_aliasing_pass.dstSubpass      = _main_camera_subpass_ui;
        ui_pass_depend_on_anti_aliasing_pass.srcStageMask =
            VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
        ui_pass_depend_on_anti_aliasing_pass.dstStageMask =
            VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
        ui_pass_depend_on_anti_aliasing_pass.srcAccessMask =
            VK_ACCESS_SHADER_WRITE_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
        ui_pass_depend_on_anti_aliasing_pass.dstAccessMask =
            VK_ACCESS_SHADER_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_READ_BIT;
        ui_pass_depend_on_anti_aliasing_pass.dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT;

        VkSubpassDependency& combine_ui_pass_depend_on_ui_pass = dependencies[7];
        combine_ui_pass_depend_on_ui_pass.srcSubpass      = _main_camera_subpass_ui;
        combine_ui_pass_depend_on_ui_pass.dstSubpass      = _main_camera_subpass_combine_ui;
        combine_ui_pass_depend_on_ui_pass.srcStageMask =
            VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
        combine_ui_pass_depend_on_ui_pass.dstStageMask =
            VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
```

```
CHMENT_OUTPUT_BIT;
        combine_ui_pass_depend_on_ui_pass.srcAccessMask =
            VK_ACCESS_SHADER_WRITE_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
        combine_ui_pass_depend_on_ui_pass.dstAccessMask =
            VK_ACCESS_SHADER_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_READ_BIT;
        combine_ui_pass_depend_on_ui_pass.dependencyFlags = VK_DEPENDENCY_BY_REG
ION_BIT;
         // >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>omit
         }
```

engine/source/runtime/function/render/source/vulkan_manager/passes/render_passes.cpp

C++

```cpp
bool Pilot::PVulkanManager::initializeRenderPass()
{
// >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>omit
    // anti-aliasing pass at odd framebuffer
    m_anti_aliasing_pass.initialize(m_main_camera_pass.getRenderPass(),
                                    m_main_camera_pass.getFramebufferImageViews()
[_main_camera_pass_backup_buffer_odd]);
// >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>omit
}
```

Shader Codes:

engine/shader/glsl/anti_aliasing.frag

OpenGL Shading Language

```
#version 310 es

#extension GL_GOOGLE_include_directive : enable

// debugging defines
#define DEBUG
// #define DISABLED
// #define VERTIAL_SPLIT

#include "constants.h"

layout(input_attachment_index = 0, set = 0, binding = 0) uniform highp subpassIn
put in_color;
layout(set=0, binding = 1) uniform highp sampler2D in_texture_sampler;

layout(location = 0) in highp vec2 in_uv;

```

```
struct ResolutionData
{
    highp vec4 screen_resolution;
    highp vec4 editor_screen_resolution;
};

layout(push_constant) uniform constants
{
    ResolutionData resolution_data;
};

layout(location = 0) out highp vec4 out_color;

highp vec2 get_screen_uv(highp vec2 uv)
{
    highp vec4 screen_resolution = resolution_data.screen_resolution;
    highp vec4 editor_screen_resolution = resolution_data.editor_screen_resolution;

    highp vec2 editor_ratio = editor_screen_resolution.zw / screen_resolution.zw;
    highp vec2 offset = editor_screen_resolution.xy / screen_resolution.zw;

    return offset.xy + uv.xy * editor_ratio;
}

// Created by Reinder Nijhoff 2016
// Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
// @reindernijhoff
//
// https://www.shadertoy.com/view/ls3GWS
#define FXAA_SPAN_MAX 8.0
#define FXAA_REDUCE_MUL   (1.0/FXAA_SPAN_MAX)
#define FXAA_REDUCE_MIN   (1.0/128.0)
#define FXAA_SUBPIX_SHIFT (1.0/4.0)

highp vec3 FxaaPixelShader( highp vec4 uv, sampler2D tex, highp vec2 rcpFrame) {

    highp vec3 rgbNW = textureLod(tex, uv.zw, 0.0).xyz;
    highp vec3 rgbNE = textureLod(tex, uv.zw + vec2(1,0)*rcpFrame.xy, 0.0).xyz;
    highp vec3 rgbSW = textureLod(tex, uv.zw + vec2(0,1)*rcpFrame.xy, 0.0).xyz;
    highp vec3 rgbSE = textureLod(tex, uv.zw + vec2(1,1)*rcpFrame.xy, 0.0).xyz;
    highp vec3 rgbM  = textureLod(tex, uv.xy, 0.0).xyz;

    highp vec3 luma = vec3(0.299, 0.587, 0.114);
    highp float lumaNW = dot(rgbNW, luma);
    highp float lumaNE = dot(rgbNE, luma);
    highp float lumaSW = dot(rgbSW, luma);
```

```
    highp float lumaSE = dot(rgbSE, luma);
    highp float lumaM  = dot(rgbM,  luma);

    highp float lumaMin = min(lumaM, min(min(lumaNW, lumaNE), min(lumaSW, lumaSE)));
    highp float lumaMax = max(lumaM, max(max(lumaNW, lumaNE), max(lumaSW, lumaSE)));

    highp vec2 dir;
    dir.x = -((lumaNW + lumaNE) - (lumaSW + lumaSE));
    dir.y =  ((lumaNW + lumaSW) - (lumaNE + lumaSE));

    highp float dirReduce = max(
        (lumaNW + lumaNE + lumaSW + lumaSE) * (0.25 * FXAA_REDUCE_MUL),
        FXAA_REDUCE_MIN);
    highp float rcpDirMin = 1.0/(min(abs(dir.x), abs(dir.y)) + dirReduce);

    dir = min(vec2( FXAA_SPAN_MAX,  FXAA_SPAN_MAX),
          max(vec2(-FXAA_SPAN_MAX, -FXAA_SPAN_MAX),
          dir * rcpDirMin)) * rcpFrame.xy;

    highp vec3 rgbA = (1.0/2.0) * (
        textureLod(tex, uv.xy + dir * (1.0/3.0 - 0.5), 0.0).xyz +
        textureLod(tex, uv.xy + dir * (2.0/3.0 - 0.5), 0.0).xyz);
    highp vec3 rgbB = rgbA * (1.0/2.0) + (1.0/4.0) * (
        textureLod(tex, uv.xy + dir * (0.0/3.0 - 0.5), 0.0).xyz +
        textureLod(tex, uv.xy + dir * (3.0/3.0 - 0.5), 0.0).xyz);

    highp float lumaB = dot(rgbB, luma);

    if((lumaB < lumaMin) || (lumaB > lumaMax)) return rgbA;

    return rgbB;
}

highp vec4 apply() {
    highp vec2 uv = get_screen_uv(in_uv);

    highp vec2 rcpFrame = 1.0 / resolution_data.screen_resolution.zw;
    highp vec4 uv4 = vec4( uv, uv - (rcpFrame * (0.5 + FXAA_SUBPIX_SHIFT)));

    highp vec3 temp_color = FxaaPixelShader(uv4, in_texture_sampler, rcpFrame);

    // as game display, no need to care color.a
#ifdef DEBUG
    return vec4(temp_color.rgb, 1.0);
#else
    return vec4(temp_color.rgb, 1.0);
```

```
#endif
}


void main() {

    highp vec4 color = subpassLoad(in_color).rgba;
#ifdef DISABLED
    out_color = color;
#else
    // out_color = color;
#ifdef DEBUG
#ifdef VERTIAL_SPLIT
    if (in_uv.x > 0.5) {
#else
    if (in_uv.x > in_uv.y) {
#endif
        out_color = color;
    } else {
#endif
    // shader out_color as following:

    out_color = apply();
    // out_color = vec4(color.r, color.g, 0.1, 0.1);
#ifdef DEBUG
    }
#endif
#endif
}
```

engine/shader/glsl/anti_aliasing.vert

OpenGL Shading Language

```
#version 310 es

#extension GL_GOOGLE_include_directive : enable

#include "constants.h"

layout(location = 0) out vec2 out_uv;

void main() {
    const vec3 fullscreen_triangle_positions[3] = vec3[3](vec3(3.0, 1.0, 0.5), vec3(-1.0, 1.0, 0.5), vec3(-1.0, -3.0, 0.5));

    gl_Position = vec4(fullscreen_triangle_positions[gl_VertexIndex], 1.0);

    // rescale from -1 to 3 to 0 to 1
    out_uv = gl_Position.xy * 0.5 + 0.5;
}
```