

Games104_homework2_report

Online version: <https://nbqmlgi3yg.feishu.cn/docs/doccnkalDESWbtgcZuONs3BU6c>

The content in the online version may be more updated.

Author: huandzh@gmail.com ([huandzh](#) ([DHuan](#)) ([github.com](#)))

Date: 2022-05-10

a> ColorGrading Code

Shader code:

OpenGL Shading Language

```
1  #version 310 es
2
3  #extension GL_GOOGLE_include_directive : enable
4
5  #include "constants.h"
6
7  layout(input_attachment_index = 0, set = 0, binding = 0) uniform highp subpassInput in_color;
8
9  layout(set = 0, binding = 1) uniform sampler2D color_grading_lut_texture_sampler;
10
11 layout(location = 0) out highp vec4 out_color;
12
13 highp vec2 lut_uv(highp float red, highp float green,
14                  highp float blue_slice, highp float slice_size) {
15     // u = (u_Red + u_Blue) / size, v = u_green
16     // `min` fix special case for color.b = 1.0F
17     return vec2((red + min(blue_slice, slice_size - 1.0F)) / slice_size, green);
18 }
19
20 highp float get_slice_size() {
21     highp ivec2 lut_tex_size = textureSize(color_grading_lut_texture_sampler, 0);
22     return float(lut_tex_size.y);
23     // return 16.0F; // can be hard coded as 16 for color_grading_lut_01.png
24 }
25
26 highp vec4 get_lut_color(highp vec4 color, highp float slice_size) {
```

```

27 // scale blue by slice_size, and get integral and fractional part
28 // example: with a sampler of 16 slices
29 // 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_size), 0.0);
36 highp vec4 color_right = textureLod(color_grading_lut_texture_sampler,
37                                   lut_uv(color.r, color.g, slice + 1.0F, slice_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 \(shadertoy.com\)](https://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:

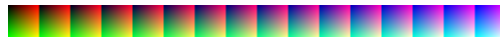
- Line 32: Blue offset should start from the left-bottom corner of the slice, thus blue should be mapped to `[0, slice_size-1]`.
- Line 17: When the last slice should be used as the left slice, the right slice should be clamped as the last slice too. `if` check is avoided by using `min`.

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:



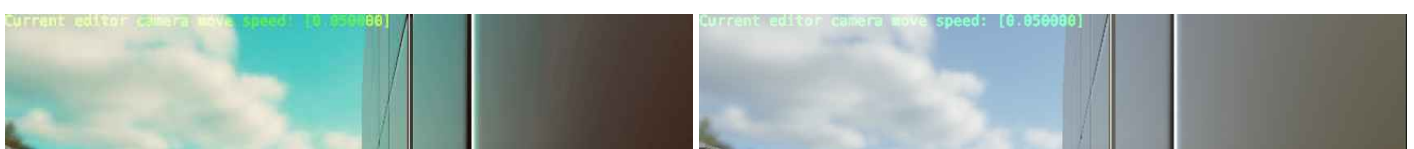
Method

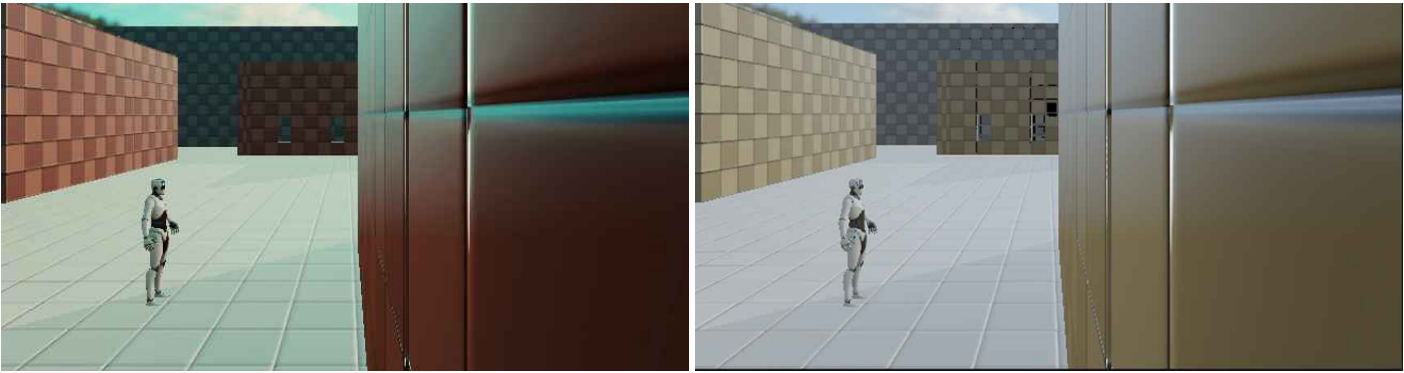
1. Get a screen snapshot of display in `Pilot Engine` without color grading
2. Tune the image to desired using adjustment layers in `Photoshop`
3. Copy the layers above a neutral LUT image as background
4. Export the result to a custom LUT image (png format without transparency)
5. Add it to engine assets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

Expected image effect

A color grading gives the feeling of early autumn:

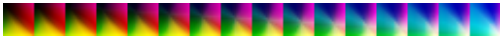
- Left: expected
- Right: without color grading





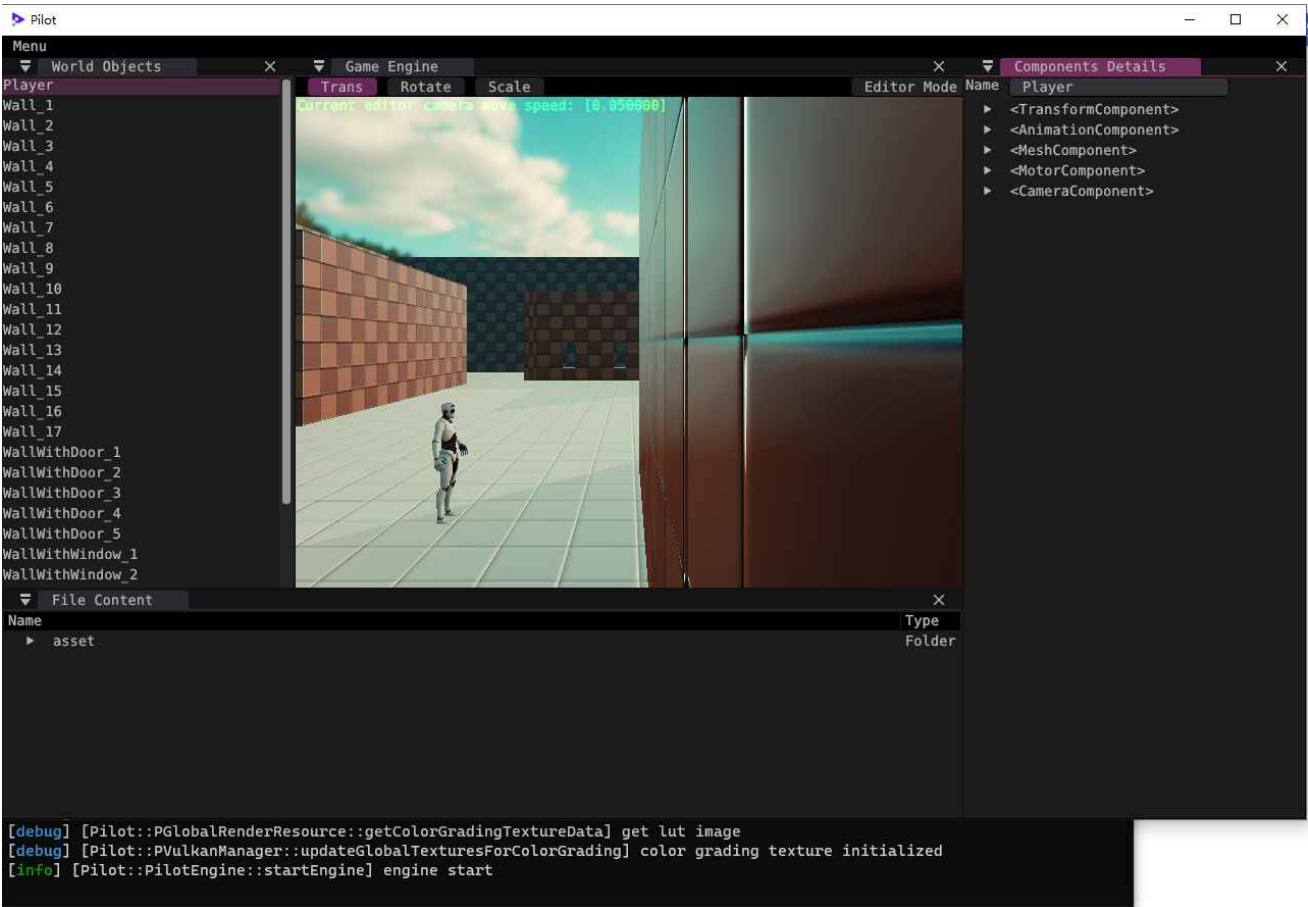
Custom LUT Image

color_grading_lut_autumn.png

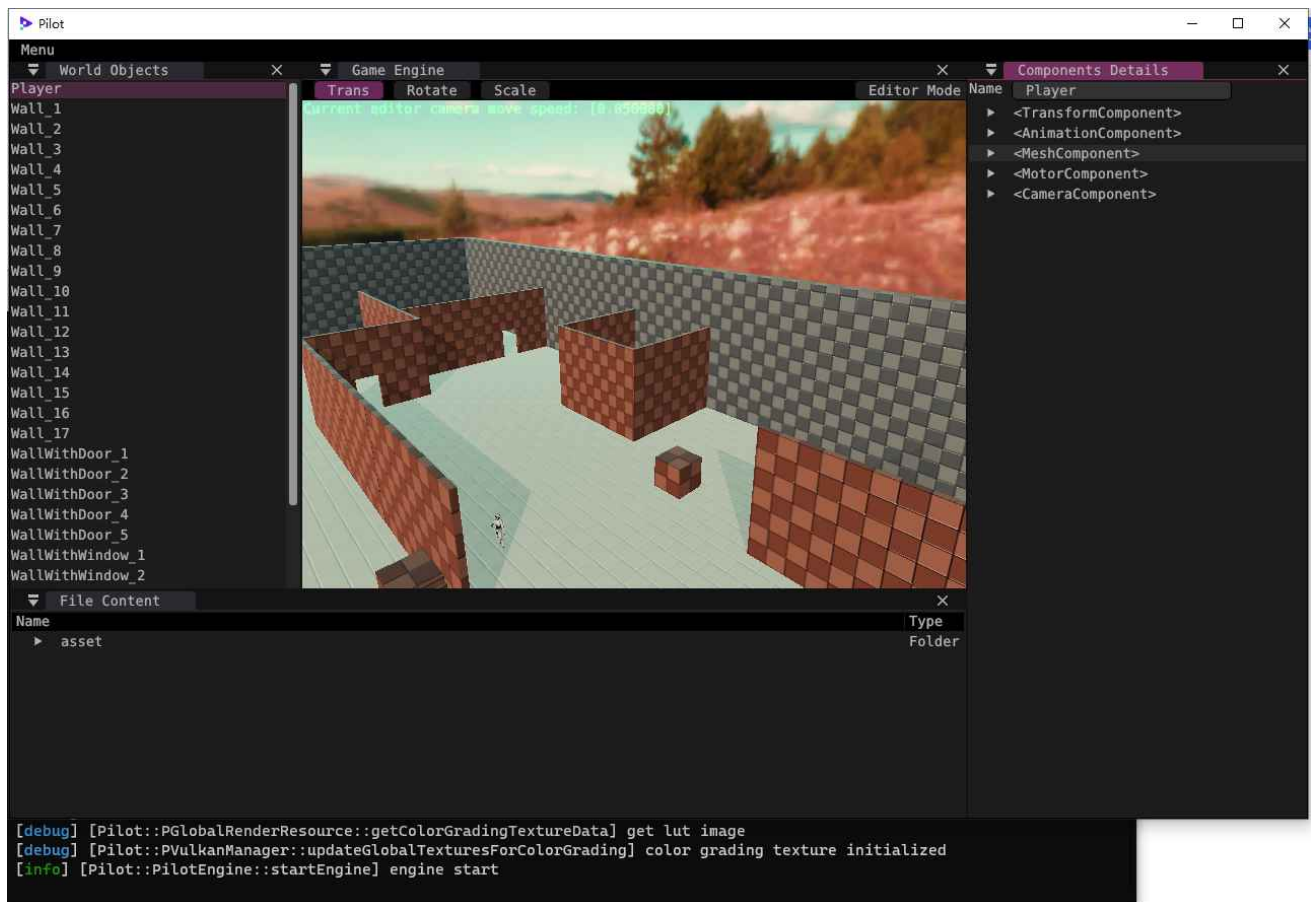


Live in Pilot Engine

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.

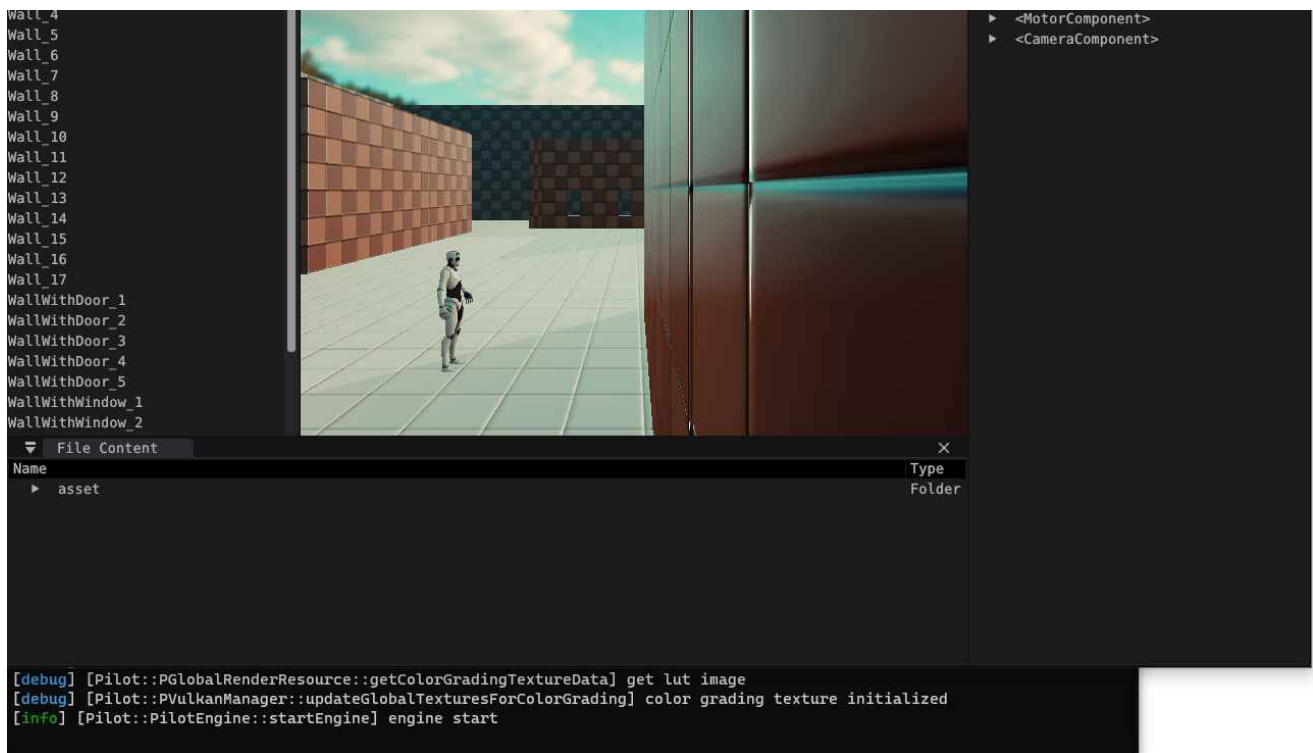
And the algorithm in shader has been adapted from established code from reference.

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

Subpass Rendering Result:

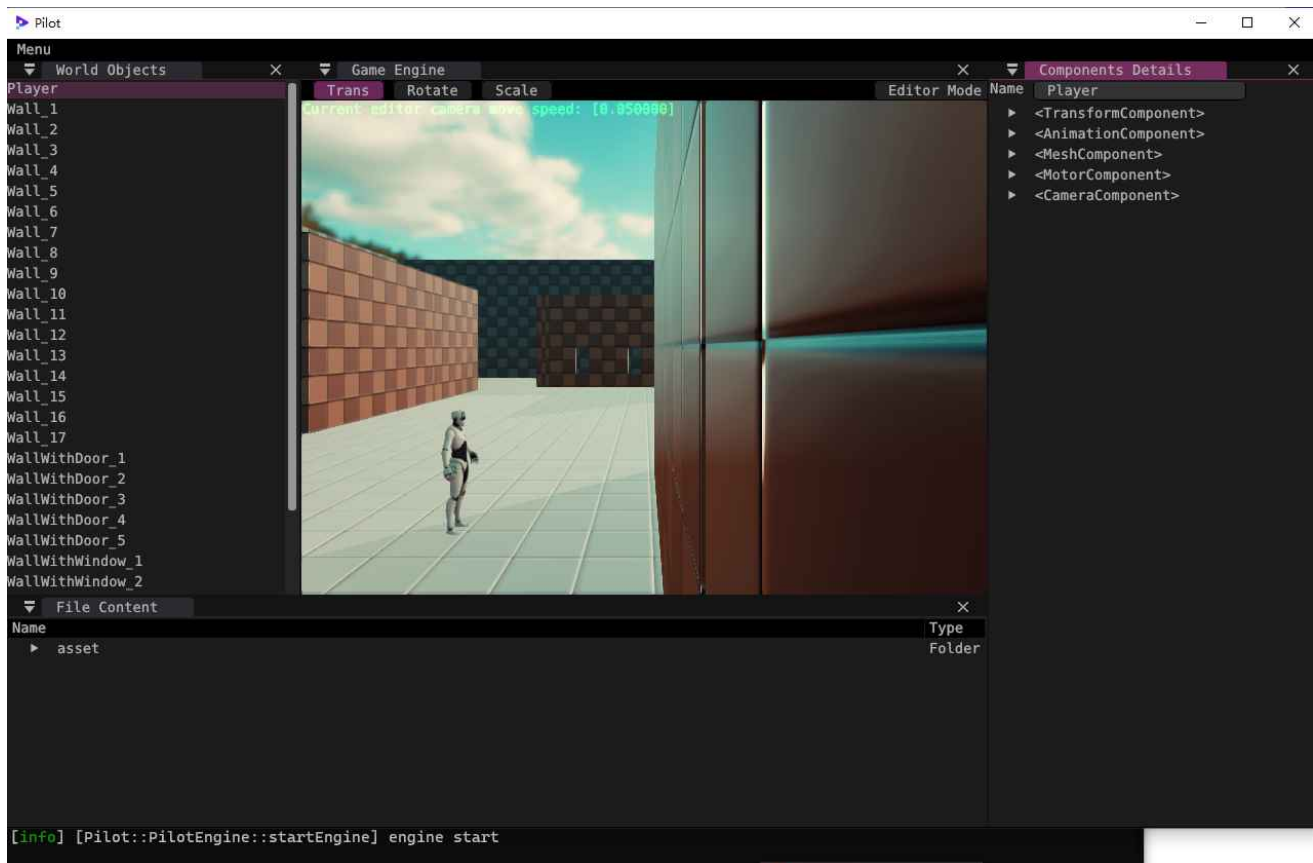
FXAA Off:





FXAA On:

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

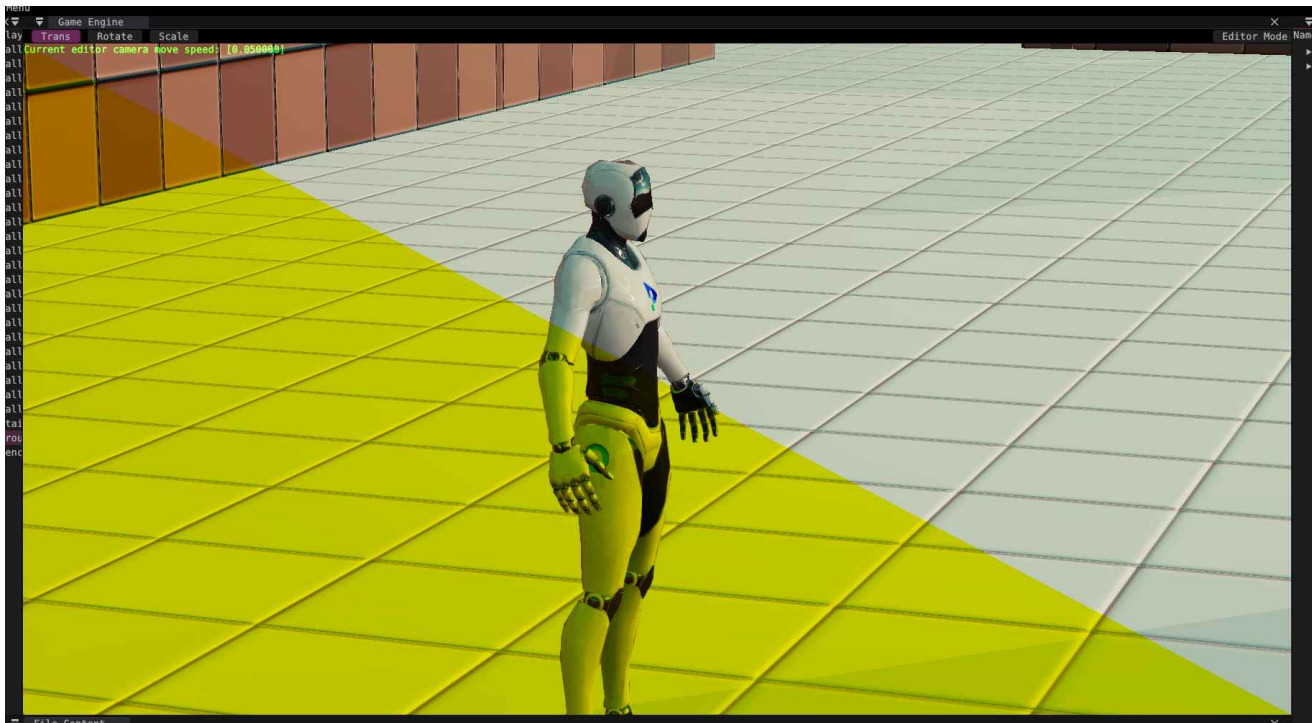


Live in Pilot Engine(Debug mode)

(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` algorithm in frag shader
10. Remove any `Vulkan` validation errors

Coding notes

In the long way to finishing these steps, I have encountered many 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++

[illegible]

[illegible]

engine/source/runtime/function/render/source/vulkan_manager/passes/anti_aliasing.cpp
(indeference routine omitted)

C++

```

1      void PAntiAliasingPass::setupDescriptorSetLayout()
2      {
3          _descriptor_infos.resize(1);
4
5          VkDescriptorSetLayoutBinding post_process_global_layout_bindings[2] =
6              {};
7
8          VkDescriptorSetLayoutBinding& post_process_global_layout_input_attachment_binding =
9              post_process_global_layout_bindings[0];
10         post_process_global_layout_input_attachment_binding.binding = 0;
11         post_process_global_layout_input_attachment_binding.descriptorType = VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT;
12         post_process_global_layout_input_attachment_binding.descriptorCount = 1;
13         post_process_global_layout_input_attachment_binding.stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT;
14
15         VkDescriptorSetLayoutBinding& post_process_global_layout_image_sampler_binding =
16             post_process_global_layout_bindings[1];

```

```

16         post_process_global_layout_image_sampler_binding.binding = 1;
17         post_process_global_layout_image_sampler_binding.descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
18         post_process_global_layout_image_sampler_binding.descriptorCount = 1;
19         post_process_global_layout_image_sampler_binding.stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT;
20
21         VkDescriptorSetLayoutCreateInfo post_process_global_layout_create_info;
22         post_process_global_layout_create_info.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
23         post_process_global_layout_create_info.pNext = NULL;
24         post_process_global_layout_create_info.flags = 0;
25         post_process_global_layout_create_info.bindingCount =
26             sizeof(post_process_global_layout_bindings) / sizeof(post_process_global_layout_bindings[0]);
27         post_process_global_layout_create_info.pBindings = post_process_global_layout_bindings;
28
29         if (VK_SUCCESS != vkCreateDescriptorSetLayout(m_p_vulkan_context->device,
30                                                     &post_process_global_layout_create_info,
31                                                     NULL,
32                                                     &descriptor_infos[0].layout))
33         {
34             throw std::runtime_error("create post process global layout");
35         }
36     }
37
38     void PAntiAliasingPass::setupPipelines()
39     {
40         _render_pipelines.resize(1);
41
42         VkDescriptorSetLayout descriptorset_layouts[1] = {_descriptor_infos[0].layout};
43         VkPipelineLayoutCreateInfo pipeline_layout_create_info {};
44         pipeline_layout_create_info.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
45         pipeline_layout_create_info.setLayoutCount = 1;
46         pipeline_layout_create_info.pSetLayouts = descriptorset_layouts;
47
48         // per https://vkguide.dev/docs/chapter-3/push\_constants/
49         // using push constants
50         // setup push constants
51         VkPushConstantRange push_constant;
52         // this push constant range starts at the beginning
53         push_constant.offset = 0;
54         // this push constant range takes up the size of the struct

```

[illegible]

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

```

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];

```



```

33         anti_aliasing_pass_depend_on_color_grading_pass.srcSubpass          = _
main_camera_subpass_color_grading;
34         anti_aliasing_pass_depend_on_color_grading_pass.dstSubpass          = _
main_camera_subpass_anti_aliasing;
35         anti_aliasing_pass_depend_on_color_grading_pass.srcStageMask =
36             VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
37         anti_aliasing_pass_depend_on_color_grading_pass.dstStageMask =
38             VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
39         anti_aliasing_pass_depend_on_color_grading_pass.srcAccessMask =
40             VK_ACCESS_SHADER_WRITE_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
41         anti_aliasing_pass_depend_on_color_grading_pass.dstAccessMask =
42             VK_ACCESS_SHADER_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_READ_BIT;
43         anti_aliasing_pass_depend_on_color_grading_pass.dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT;
44
45         // anti aliasing
46         VkSubpassDependency& ui_pass_depend_on_anti_aliasing_pass = dependencies
[6];
47         ui_pass_depend_on_anti_aliasing_pass.srcSubpass          = _main_camera
_subpass_anti_aliasing;
48         ui_pass_depend_on_anti_aliasing_pass.dstSubpass          = _main_camera
_subpass_ui;
49         ui_pass_depend_on_anti_aliasing_pass.srcStageMask =
50             VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
51         ui_pass_depend_on_anti_aliasing_pass.dstStageMask =
52             VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
53         ui_pass_depend_on_anti_aliasing_pass.srcAccessMask =
54             VK_ACCESS_SHADER_WRITE_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
55         ui_pass_depend_on_anti_aliasing_pass.dstAccessMask =
56             VK_ACCESS_SHADER_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_READ_BIT;
57         ui_pass_depend_on_anti_aliasing_pass.dependencyFlags = VK_DEPENDENCY_BY_
REGION_BIT;
58
59         VkSubpassDependency& combine_ui_pass_depend_on_ui_pass = dependencies[7
];
60         combine_ui_pass_depend_on_ui_pass.srcSubpass          = _main_camera_su
bpass_ui;
61         combine_ui_pass_depend_on_ui_pass.dstSubpass          = _main_camera_su
bpass_combine_ui;
62         combine_ui_pass_depend_on_ui_pass.srcStageMask =
63             VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
64         combine_ui_pass_depend_on_ui_pass.dstStageMask =
65             VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;

```

[illegible]

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

[illegible]

Shader Codes:

engine/shader/glsl/anti_aliasing.frag

OpenGL Shading Language

```

1  #version 310 es
2
3  #extension GL_GOOGLE_include_directive : enable
4
5  // debugging defines
6  #define DEBUG
7  // #define DISABLED
8  // #define VERTICAL_SPLIT
9
10 #include "constants.h"
11
12 layout(input_attachment_index = 0, set = 0, binding = 0) uniform highp subpassInput
    in_color;
13 layout(set=0, binding = 1) uniform highp sampler2D in_texture_sampler;
14
15 layout(location = 0) in highp vec2 in_uv;
16
17 layout(location = 0) out highp vec4 out_color;

```

```

17 struct ResolutionData
18 {
19     highp vec4 screen_resolution;
20     highp vec4 editor_screen_resolution;
21 };
22
23 layout(push_constant) uniform constants
24 {
25     ResolutionData resolution_data;
26 };
27
28 layout(location = 0) out highp vec4 out_color;
29
30 highp vec2 get_screen_uv(highp vec2 uv)
31 {
32     highp vec4 screen_resolution = resolution_data.screen_resolution;
33     highp vec4 editor_screen_resolution = resolution_data.editor_screen_resoluti
on;
34
35     highp vec2 editor_ratio = editor_screen_resolution.zw / screen_resolution.z
w;
36     highp vec2 offset = editor_screen_resolution.xy / screen_resolution.zw;
37
38     return offset.xy + uv.xy * editor_ratio;
39 }
40
41 // Created by Reinder Nijhoff 2016
42 // Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Licen
se.
43 // @reindernijhoff
44 //
45 // https://www.shadertoy.com/view/ls3GWS
46 #define FXAA_SPAN_MAX 8.0
47 #define FXAA_REDUCE_MUL (1.0/FXAA_SPAN_MAX)
48 #define FXAA_REDUCE_MIN (1.0/128.0)
49 #define FXAA_SUBPIX_SHIFT (1.0/4.0)
50
51 highp vec3 FxaaPixelShader( highp vec4 uv, sampler2D tex, highp vec2 rcpFrame) {
52
53     highp vec3 rgbNW = textureLod(tex, uv.zw, 0.0).xyz;
54     highp vec3 rgbNE = textureLod(tex, uv.zw + vec2(1,0)*rcpFrame.xy, 0.0).xyz;
55     highp vec3 rgbSW = textureLod(tex, uv.zw + vec2(0,1)*rcpFrame.xy, 0.0).xyz;
56     highp vec3 rgbSE = textureLod(tex, uv.zw + vec2(1,1)*rcpFrame.xy, 0.0).xyz;
57     highp vec3 rgbM = textureLod(tex, uv.xy, 0.0).xyz;
58
59     highp vec3 luma = vec3(0.299, 0.587, 0.114);
60     highp float lumaNW = dot(rgbNW, luma);
61     highp float lumaNE = dot(rgbNE, luma);
62     highp float lumaSW = dot(rgbSW, luma);

```

```

62     highp float lumaSW = dot(rgbSW, luma);
63     highp float lumaSE = dot(rgbSE, luma);
64     highp float lumaM  = dot(rgbM,  luma);
65
66     highp float lumaMin = min(lumaM, min(min(lumaNW, lumaNE), min(lumaSW, lumaS
E)));
67     highp float lumaMax = max(lumaM, max(max(lumaNW, lumaNE), max(lumaSW, lumaS
E)));
68
69     highp vec2 dir;
70     dir.x = -((lumaNW + lumaNE) - (lumaSW + lumaSE));
71     dir.y = ((lumaNW + lumaSW) - (lumaNE + lumaSE));
72
73     highp float dirReduce = max(
74         (lumaNW + lumaNE + lumaSW + lumaSE) * (0.25 * FXAA_REDUCE_MUL),
75         FXAA_REDUCE_MIN);
76     highp float rcpDirMin = 1.0/(min(abs(dir.x), abs(dir.y)) + dirReduce);
77
78     dir = min(vec2( FXAA_SPAN_MAX,  FXAA_SPAN_MAX),
79         max(vec2(-FXAA_SPAN_MAX, -FXAA_SPAN_MAX),
80         dir * rcpDirMin)) * rcpFrame.xy;
81
82     highp vec3 rgbA = (1.0/2.0) * (
83         textureLod(tex, uv.xy + dir * (1.0/3.0 - 0.5), 0.0).xyz +
84         textureLod(tex, uv.xy + dir * (2.0/3.0 - 0.5), 0.0).xyz);
85     highp vec3 rgbB = rgbA * (1.0/2.0) + (1.0/4.0) * (
86         textureLod(tex, uv.xy + dir * (0.0/3.0 - 0.5), 0.0).xyz +
87         textureLod(tex, uv.xy + dir * (3.0/3.0 - 0.5), 0.0).xyz);
88
89     highp float lumaB = dot(rgbB, luma);
90
91     if((lumaB < lumaMin) || (lumaB > lumaMax)) return rgbA;
92
93     return rgbB;
94 }
95
96 highp vec4 apply() {
97     highp vec2 uv = get_screen_uv(in_uv);
98
99     highp vec2 rcpFrame = 1.0 / resolution_data.screen_resolution.zw;
100    highp vec4 uv4 = vec4( uv, uv - (rcpFrame * (0.5 + FXAA_SUBPIX_SHIFT)));
101
102    highp vec3 temp_color = FxaaPixelShader(uv4, in_texture_sampler, rcpFrame);
103
104    // as game display, no need to care color.a
105    #ifdef DEBUG
106        return vec4(temp_color.rgb, 1.0);
107    #else
108        return vec4(temp_color.rgb, 1.0);

```

```

109 #endif
110 }
111
112
113 void main() {
114
115     highp vec4 color = subpassLoad(in_color).rgba;
116 #ifdef DISABLED
117     out_color = color;
118 #else
119     // out_color = color;
120 #ifdef DEBUG
121 #ifdef VERTICAL_SPLIT
122     if (in_uv.x > 0.5) {
123 #else
124     if (in_uv.x > in_uv.y) {
125 #endif
126         out_color = color;
127     } else {
128 #endif
129         // shader out_color as following:
130
131         out_color = apply();
132         // out_color = vec4(color.r, color.g, 0.1, 0.1);
133 #ifdef DEBUG
134     }
135 #endif
136 #endif
137 }

```

engine/shader/glsl/anti_aliasing.vert

OpenGL Shading Language

```
1  #version 310 es
2
3  #extension GL_GOOGLE_include_directive : enable
4
5  #include "constants.h"
6
7  layout(location = 0) out vec2 out_uv;
8
9  void main() {
10     const vec3 fullscreen_triangle_positions[3] = vec3[3](vec3(3.0, 1.0, 0.5), v
        ec3(-1.0, 1.0, 0.5), vec3(-1.0, -3.0, 0.5));
11
12     gl_Position = vec4(fullscreen_triangle_positions[gl_VertexIndex], 1.0);
13
14     // rescale from -1 to 3 to 0 to 1
15     out_uv = gl_Position.xy * 0.5 + 0.5;
16 }
```