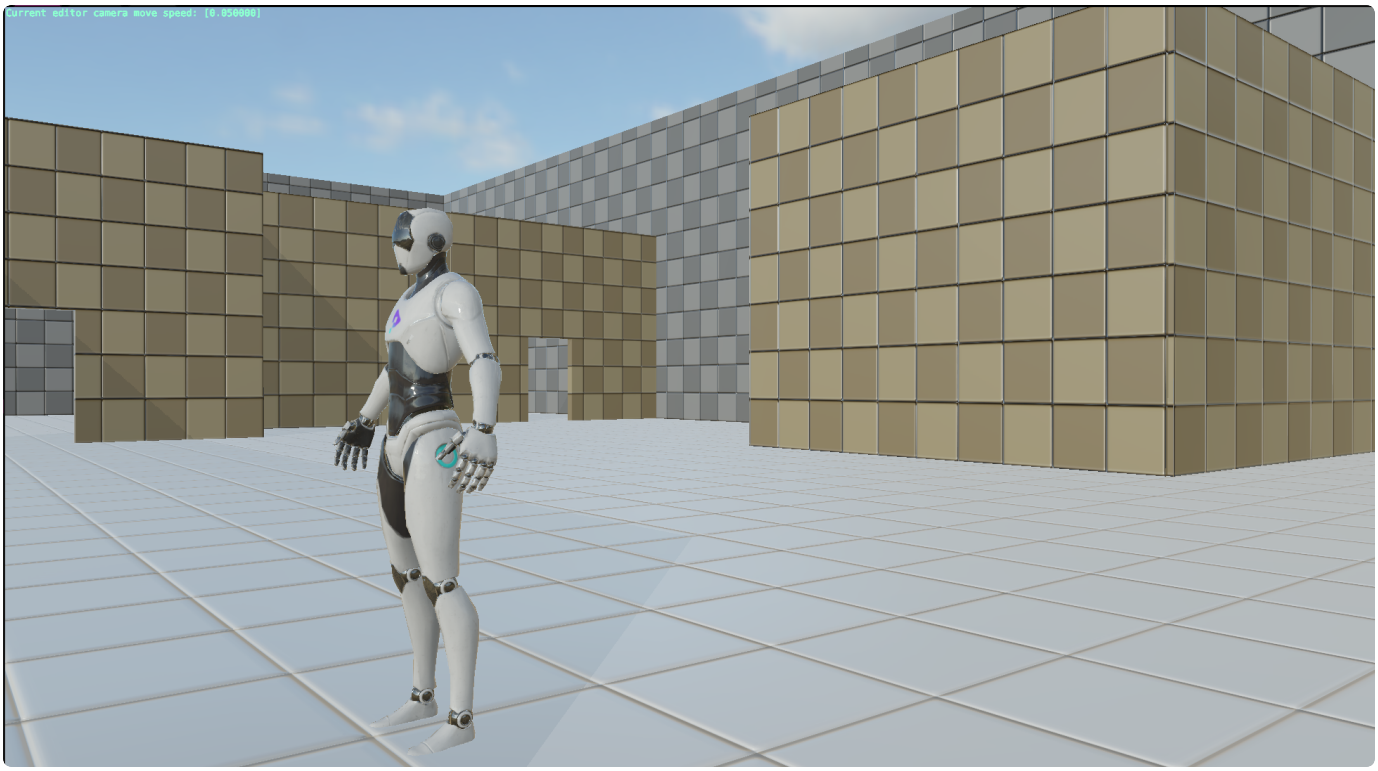
Games104_homework2_report

原图：



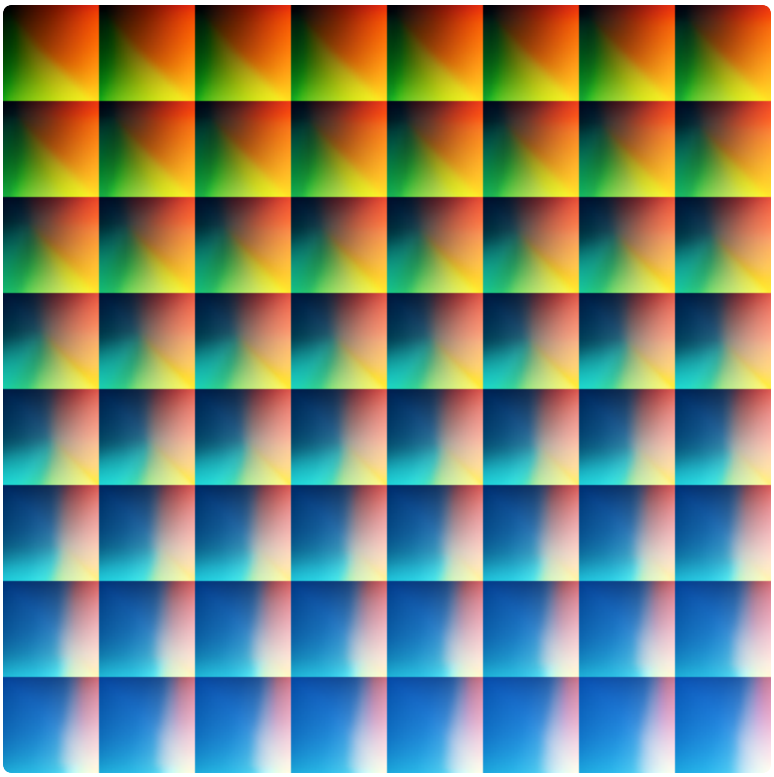
1. 实现ColorGrading功能

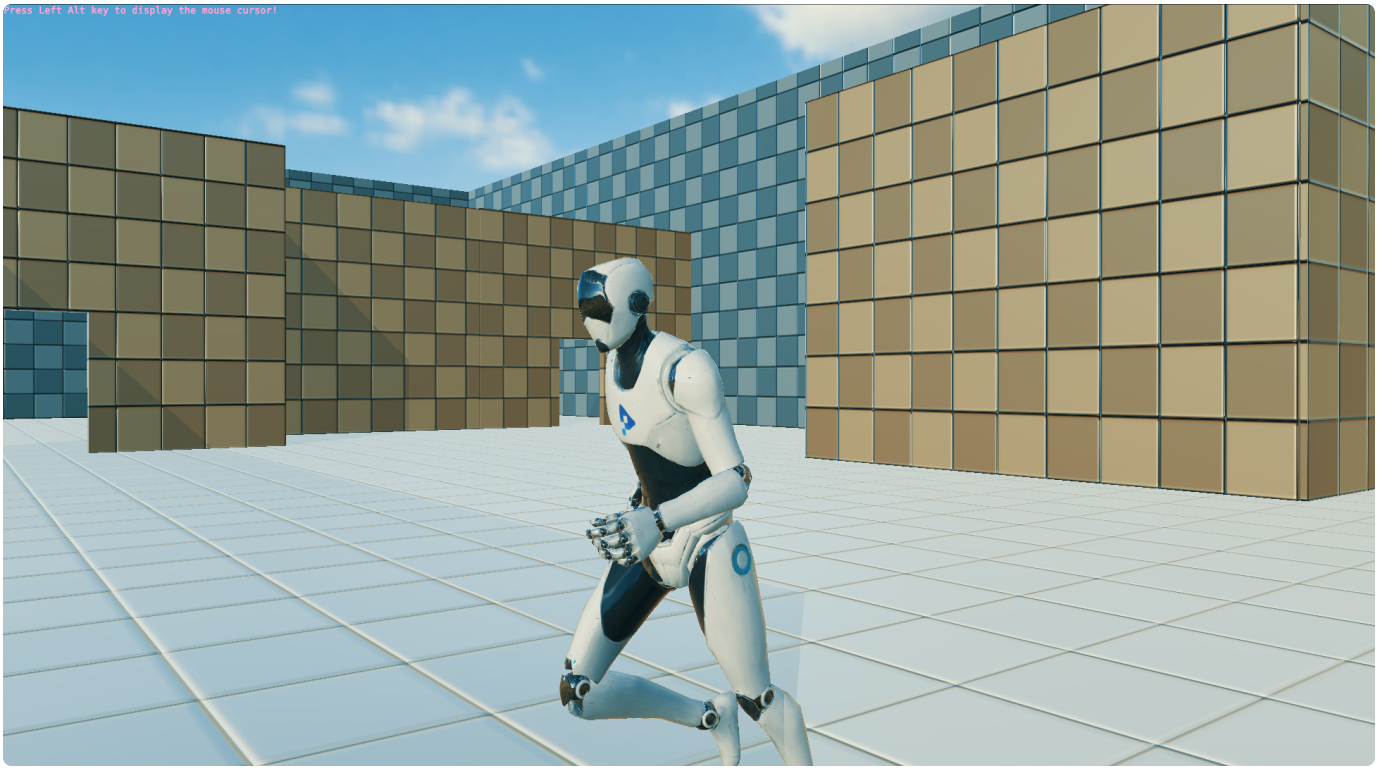
```
#version 310 es

#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;

#define CELLS_PER_ROW 8.0
#define CELLS_PER_COLUM 8.0

void main()
{
    highp vec4 base_color = subpassLoad(in_color).rgba;
    highp ivec2 lut_tex_size = textureSize(color_grading_lut_texture_sampler, 0);

    highp float half_pixel_size = 0.5 / float(lut_tex_size.x * lut_tex_size.y);
    highp float cell_xSize =  1.0 / CELLS_PER_ROW;
    highp float cell_ySize =  1.0 / CELLS_PER_COLUM;

    highp float blue_cell = base_color.b * (CELLS_PER_ROW * CELLS_PER_COLUM - 1.0);
    highp float xOffset = half_pixel_size + base_color.r * (cell_xSize - (2.0 * half_pixel_size));
    highp float yOffset = half_pixel_size + base_color.g * (cell_ySize - (2.0 * half_pixel_size));

    highp vec2 lower_cell, lower_sample, upper_cell, upper_sample;

    lower_cell.y = floor(blue_cell / CELLS_PER_ROW);
    lower_cell.x = floor(blue_cell) - lower_cell.y * CELLS_PER_COLUM;
    lower_sample.x = lower_cell.x * cell_xSize + xOffset;
    lower_sample.y = lower_cell.y * cell_ySize + yOffset;

    upper_cell.y = floor(ceil(blue_cell) / CELLS_PER_ROW);
    upper_cell.x = ceil(blue_cell) - upper_cell.y * CELLS_PER_COLUM;
    upper_sample.x = upper_cell.x * cell_xSize + xOffset;
    upper_sample.y = upper_cell.y * cell_ySize + yOffset;

```

```
    highp vec3 color = mix(
        texture(color_grading_lut_texture_sampler, lower_sample).rgb,
        texture(color_grading_lut_texture_sampler, upper_sample).rgb,
        fract(blue_cell)
        );

    out_color = mix(base_color, vec4(color, 1.0), 1.0);
}
```

- 使用引擎资源库中的lut02图，在color_grading.frag里进行编写
- 固定blue通道值，通过blue的ceil和floor值，定位lut图中的两个cell
- 计算这两个cell中，red和green的offset，用blue定位用的小数部分作为权重
- 插值这两个cell，得到最终的red和green

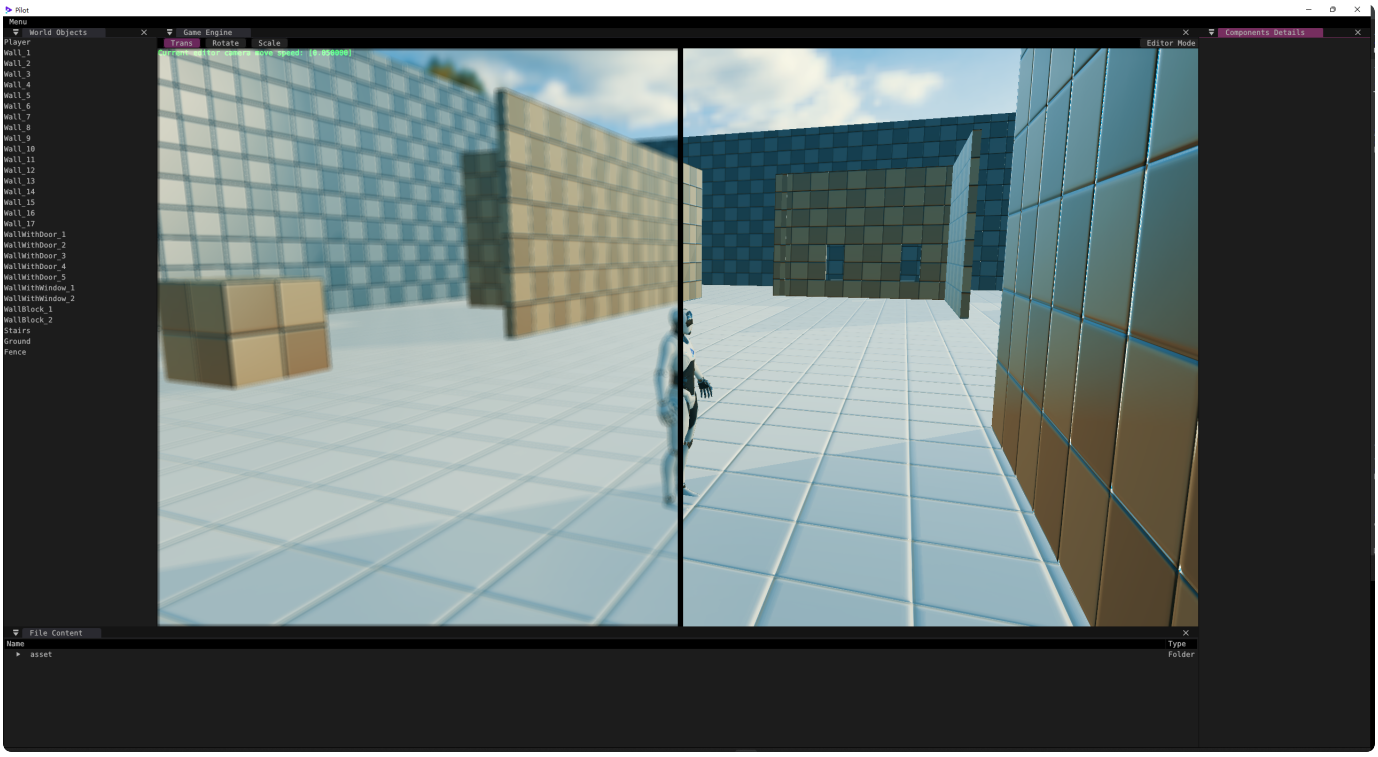
2. 自定义LUT图





- 尝试了和引擎资源不同的二维lut图
- 和第一部分类似的计算，需要修改 `#define CELLS_PER_ROW` 和 `#define CELLS_PER_COLUM` 就能得到正确结果

3. 自定义后处理效果



Vert

```glsl
#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 [-1,1] to [0,1]
    out_uv = gl_Position.xy * 0.5 + 0.5;
}

```

Fragment

```
#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 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;
};

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;
}

highp vec2 Circle(highp float Start, highp float Points, highp float Point)
{
    highp float Rad = (3.141592 * 2.0 * (1.0 / Points)) * (Point + Start);
    return vec2(sin(Rad), cos(Rad));
}

highp vec4 Blur(sampler2D tex, highp vec2 in_uv)
```

```
{
    highp vec2 uv = get_screen_uv(in_uv);
    highp vec2 PixelOffset = 1.0 / resolution_data.screen_resolution.zw;

    highp float Start = 2.0 / 14.0;
    highp vec2 Scale = 0.66 * 4.0 * 2.0 * PixelOffset.xy;

    highp vec3 N0 = texture(tex, uv + Circle(Start, 14.0, 0.0) * Scale).r
gb;
    highp vec3 N1 = texture(tex, uv + Circle(Start, 14.0, 1.0) * Scale).r
gb;
    highp vec3 N2 = texture(tex, uv + Circle(Start, 14.0, 2.0) * Scale).r
gb;
    highp vec3 N3 = texture(tex, uv + Circle(Start, 14.0, 3.0) * Scale).r
gb;
    highp vec3 N4 = texture(tex, uv + Circle(Start, 14.0, 4.0) * Scale).r
gb;
    highp vec3 N5 = texture(tex, uv + Circle(Start, 14.0, 5.0) * Scale).r
gb;
    highp vec3 N6 = texture(tex, uv + Circle(Start, 14.0, 6.0) * Scale).r
gb;
    highp vec3 N7 = texture(tex, uv + Circle(Start, 14.0, 7.0) * Scale).r
gb;
    highp vec3 N8 = texture(tex, uv + Circle(Start, 14.0, 8.0) * Scale).r
gb;
    highp vec3 N9 = texture(tex, uv + Circle(Start, 14.0, 9.0) * Scale).r
gb;
    highp vec3 N10 = texture(tex, uv + Circle(Start, 14.0, 10.0) * Scale)
.rgb;
    highp vec3 N11 = texture(tex, uv + Circle(Start, 14.0, 11.0) * Scale)
.rgb;
    highp vec3 N12 = texture(tex, uv + Circle(Start, 14.0, 12.0) * Scale)
.rgb;
    highp vec3 N13 = texture(tex, uv + Circle(Start, 14.0, 13.0) * Scale)
.rgb;
    highp vec3 N14 = texture(tex, uv).rgb;

    highp float W = 1.0 / 15.0;

    highp vec3 color = vec3(0, 0, 0);

    color.rgb =
    (N0 * W) +
    (N1 * W) +
    (N2 * W) +
    (N3 * W) +
    (N4 * W) +
    (N5 * W) +
```

```
    (N6 * W) +
    (N7 * W) +
    (N8 * W) +
    (N9 * W) +
    (N10 * W) +
    (N11 * W) +
    (N12 * W) +
    (N13 * W) +
    (N14 * W);

    return vec4(color.rgb, 1.0);
}

layout(location = 0) out highp vec4 out_color;

#define VERTIAL_SPLIT

void main()
{
    highp vec4 color = subpassLoad(in_color).rgba;
    highp vec4 blurColor = Blur(in_texture_sampler, in_uv);
#ifdef VERTIAL_SPLIT
    if (in_uv.x > 0.5)
    {
        if (in_uv.x <= 0.505)
        {
            out_color = vec4(0, 0, 0, 0);
        }
        else
        {
            out_color = color;
        }
    }
    else
    {
        out_color = blurColor;
    }
#else
    out_color = blurColor;
#endif

}
```

- 对窗口的一半做了Blur处理
- 做作业时引擎的UI和场景直接用subPass连接，不能访问相邻的像素，只能做但像素处理
- 自己添加了两个buffer，将framebuffer作为输入的贴图，传到blur pass中，使用时切换不同的

flag，最终实现blur效果

- 中间的黑色界限是根据屏幕uv坐标来判断的