coursework 3 - report

sc23y2m - Yanni Ma

May 2024

1 Render-To-Texture Setup

1.1 resources:

I created two intermediate textures: baseImage and brightImage, and created corresponding ImageView(baseView, brightView) and DescriptorSet(baseTexDesc, brightTexDesc). Create an intermediate Framebuffer through two intermediate texture ImageViews. In addition, I also created two renderpass(renderpassA, renderpassB) and two corresponding Pipeline(pipe, finalPipe). In renderpassA, the pbr calculation result and the bright calculation result(rgb component greater than 1) are output to two intermediate textures(intermediate framebuffer) respectively. Then in renderpassB, these two intermediate textures are input to finalPipe as uniform sampler2D, the two textures are sampled in the fragment shader and the sampling results are added and output to the screen(swapchain framebuffers).

1.2 synchronization:

I recorded and submitted the command twice. The first time is record_intermediate_commands() and submit_intermediate_commands(), the second time is record_final_commands() and submit_final_commands(). I created a Fence(tcbFence) and a CommandBuffer(tcbuBuffer) for the intermediate process. After recording and submitting intermediate commands, use tcbFence to ensure synchronization.

1.3 intermediate texture formats:

I'm using VK_FORMAT_R16G16B16A16_SFLOAT as the format for the intermediate texture. This format is suitable for color information of float value, has high color accuracy and range, and is suitable for intermediate calculation results. Here it helps us complete Tone Mapping.

2 Tone Mapping

In finalPipe shader, the results of the two intermediate texture samples are added. Then use the added value as a, b=a/(1+a), and output b to the screen.

Figure 1: without tone mapping

Figure 2: with tone mapping

3 Bloom

I added two RenderPass (bhPass and bvPass) and two corresponding Pipelines (bhPipe and bvPipe) to process horizontal Gaussian and vertical Gaussian respectively. In addition, I also created two new intermediate textures (bhImage and bvImage) and the corresponding ImageView and DescriptorSet. Then I created two framebuffers using bhView and bvView respectively.

First, in the fragment shader, I calculated 22 weights through the one-dimensional discrete form of the Gaussian function and normalized them. Then I linear sampled the offset by the following formula, reducing the number of iterations to 11.

$$weight_L(t_1,t_2) = weight_D(t_1) + weight_D(t_2)$$

$$offset_L(t_1,t_2) = \frac{offset_D(t_1) \cdot weight_D(t_1) + offset_D(t_2) \cdot weight_D(t_2)}{weight_L(t_1,t_2)}$$

Figure 3: linear sampled Gaussian

```
float weights[22];
CalculateWeights(weights);

float LinearWeights[11];
for(int i = 0; i < 11; ++i)
{
    LinearWeights[i] = weights[2*i] + weights[2*i+1];
}
vec2 texOffset = 1.0 / textureSize(brightTex, 0);

//22 - texOffset*0 texOffset*1 texOffset*2 ...texOffset*21
float offset[11];
for(int i = 0; i < 11; ++i)
{
    offset[i] = ( texOffset.x*(2*i)*weights[2*i]+
          texOffset.x*(2*i+1)*weights[2*i+1] )/ LinearWeights[i];
}
```

The final result is as follows:

Figure 4: with Bloom

Figure 5: with Bloom