Edge Detection

Inspired by classic Sobel filtering edge detection which labels the outlines of objects containing color difference in adjacent pixels, we come out with a similar idea to detect and draw an outline around an object with respect to the change of depth values in “Normalized Device Coordinates”. The whole process is illustrated in Figure 1.

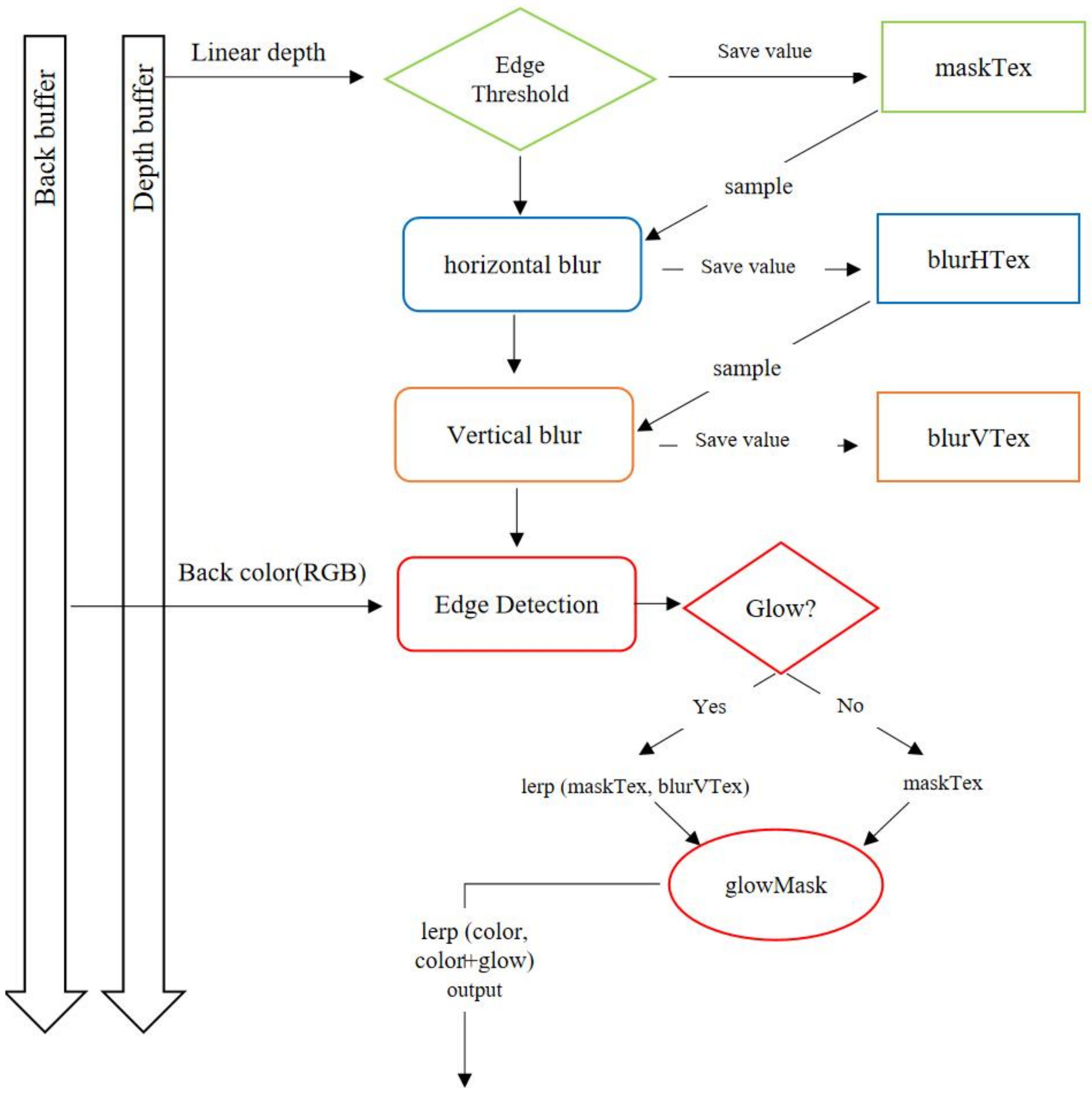
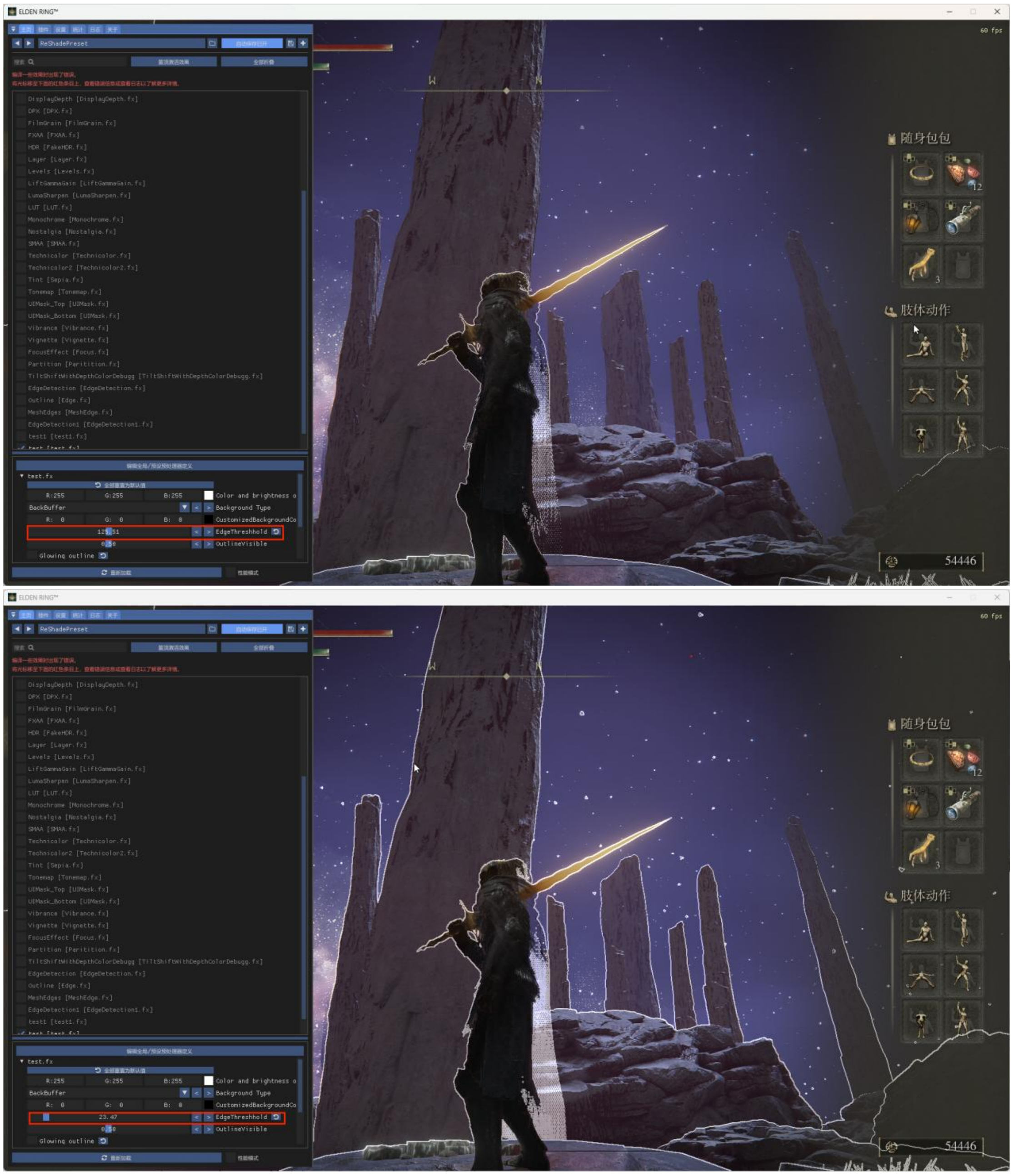


Figure 1 Overall process of edge detection

In order to implement the edge detection based on depth, we got linear depth information from the depth buffer and applied both horizontal and vertical ways of Sober kernel onto it. The size of this value represents the intensity of depth change in the scene.

We then set an uniform value “EdgeThreshold”, letting users to determine how much change in depth would allow outlines to display. The initial effect is shown in Figure 2



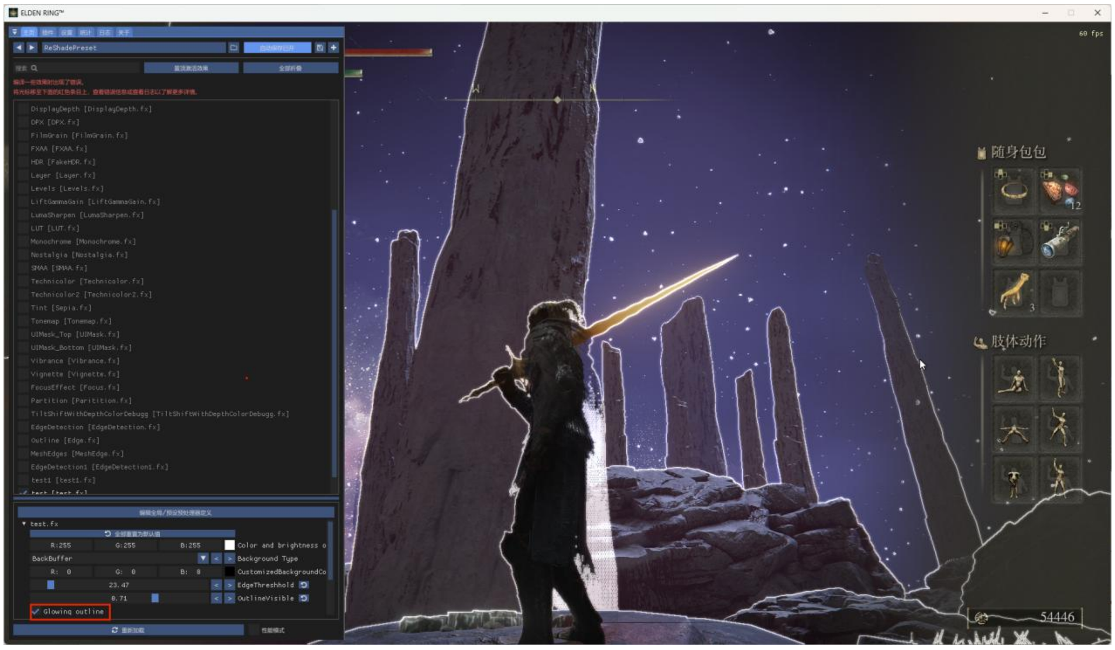
```
▼ test.fx
R:255      G:255      B:255      Color and brightness o
BackBuffer                       Background Type
R:  0      G:  0      B:  0      CustomizedBackgroundCo
125.51                           EdgeThreshhold
0.50                             OutlineVisible
Glowing outline
```

```
▼ test.fx
R:255      G:255      B:255      Color and brightness o
BackBuffer                       Background Type
R:  0      G:  0      B:  0      CustomizedBackgroundCo
23.47                            EdgeThreshhold
0.50                             OutlineVisible
Glowing outline
```

Figure 2 Basic edge detection based on depth variance

After the basic function works, we started to be not satisfied with the straight outline. What we wanted next was a glowing effect where the greatest changes in depth shined most and attenuated gradually till the change in depth was below the detection threshold. Though the current method naturally contained this concept, its effect was not very visible. It occurred to us that blurring may imitate a more natural light source diffusion effect.

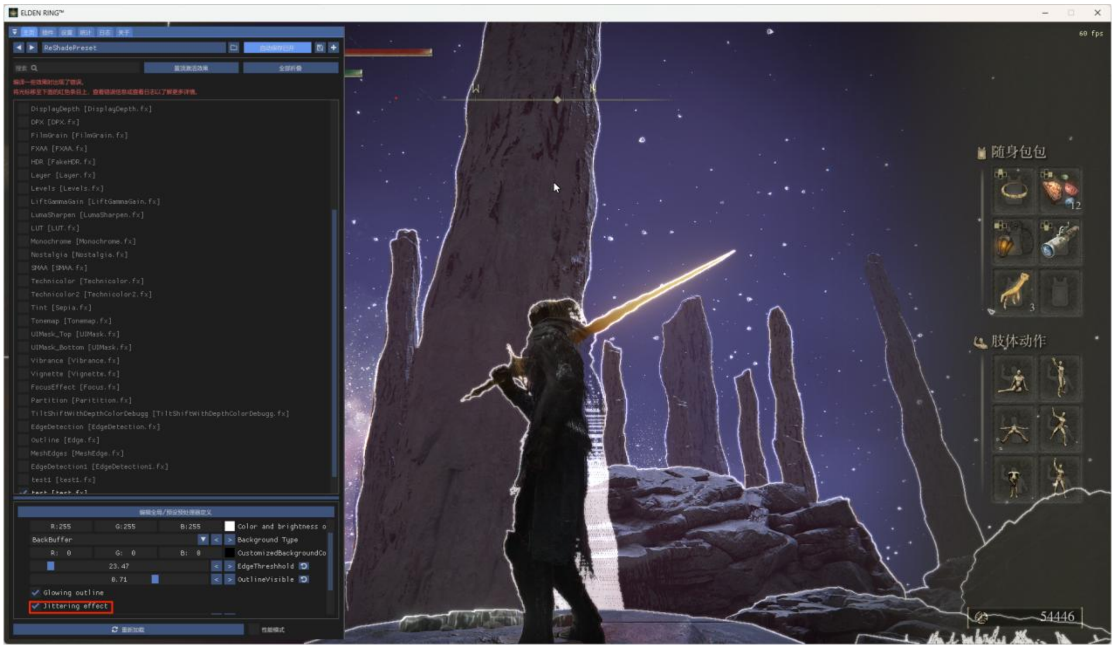
Instead of using Gaussian blur directly which combines 25 neighboring values, We first did a weighed average along horizontal direction and then did it along vertical direction. Compared to 25 calculations in classic Gaussian blur, we only introduced 10 more, thus accelerating the running speed. Each blurred data was stored temporally as a texture in one pass and was extracted as input in the later passes. By the way, the reason why we need to interpolate between non-blurred texture and blurred texture was pure blurred texture was quite fuzzy and we expected a glowing effect neither too sharp nor too vague. The glowing effect is shown in Figure 3.



```
test.fx
R:255   G:255   B:255   Color and brightness o
BackBuffer              Background Type
R: 0    G: 0    B: 0    CustomizedBackgroundCo
23.47                   EdgeThreshhold
0.71                    OutlineVisible
✓ Glowing outline
```

Figure 3 Glowing outlines

Finally, what is not shown in the flow chart is another vivid effect that allows the outlines to jitter along the horizontal axis. It looks like the distortion of a distant object under the hot weather due to the refraction of light. Considering the fixed coordinate of depth change value, we need to sample depth information of neighboring x axis instead of itself to mimic a waving effect. Intuitively, we introduced simple sin function to let sampling point oscillate in specified amplitude as well as frequency along horizontal direction. The final effect is shown in Figure 4.



```
ReShadePreset

DisplayDepth [DisplayDepth.fx]
DPX [DPX.fx]
FilmGrain [FilmGrain.fx]
FXAA [FXAA.fx]
HDR [FakeHDR.fx]
Layer [Layer.fx]
Levels [Levels.fx]
LiftGammaGain [LiftGammaGain.fx]
LumaSharpen [LumaSharpen.fx]
LUT [LUT.fx]
Monochrome [Monochrome.fx]
Nostalgia [Nostalgia.fx]
SMAA [SMAA.fx]
Technicolor [Technicolor.fx]
Technicolor2 [Technicolor2.fx]
Tint [Sepia.fx]
Tonemap [Tonemap.fx]
UIMask_Top [UIMask.fx]
UIMask_Bottom [UIMask.fx]
Vibrance [Vibrance.fx]
Vignette [Vignette.fx]
FocusEffect [Focus.fx]
Partition [Paritition.fx]
TiltShiftWithDepthColorDebugg [TiltShiftWithDepthColorDebugg.fx]
EdgeDetection [EdgeDetection.fx]
Outline [Edge.fx]
MeshEdges [MeshEdge.fx]
EdgeDetection1 [EdgeDetection1.fx]
test1 [test1.fx]
✓ test [test.fx]

R:255        G:255        B:255        Color and brightness o
BackBuffer                             Background Type
R:  0        G:  0        B:  0        CustomizedBackgroundCo
              23.47                    EdgeThreshhold
              0.71                     OutlineVisible
✓ Glowing outline
✓ Jittering effect
```

Figure 4 Jittering effect of outlines