Praktikum Echtzeit-Computergrafik
Assignment 7 – Post Processing

Technische Universität München
Institut für Informatik
Lehrstuhl für Computergrafik & Visualisierung

Christoph Neuhauser, Simon Niedermayr – SS 24

In this assignment, we will apply different image filters to our rendered image before displaying it in in ShaderLabWeb (https://vmwestermann10.in.tum.de/). This is a process called post processing.

Tasks

Task 1: Image Blurring using Render to Texture

This task is an adapted form of an assignment originally written by Bernhard Kainz for a course offered by the London Imperial College.

For this task you will use the render-to-texture (R2T) 2D fragment shader instead of the base fragment shader. These shaders are applied to a screen-aligned quad that is rendered in front of the camera. The quad is textured with the scene you have been working with so far and serves as an intermediate representation to allow image-based operations.

The framework renders the scene first into a so called framebuffer. A framebuffer is basically a texture image similar to the one that you used in the previous exercise. However, this object has the additional capability to capture the output of your render window. This function is currently one of the most important ones in applied Computer Graphics because many different image processing algorithms can be applied to this 2D texture image as post processing steps.

The render-to-texture 2D fragment shader and render-to-texture 2D vertex shader are available in the editor and act in their plain version as pass-through shader for the screen-aligned textured quad.

Your task is to extend the render-to-texture 2D fragment shader, so that it produces a simple blur effect.

Radial blur can be achieved by sampling the available texture in the direction towards the image center. In this example we work with normalized texture coordinates which means, for WebGL, that every position within the input texture is encoded within [0, 1]. Therefore the image center is located at $c = (0.5, 0.5)^T$ and the vector to the image center from any position p can be calculated by $\vec{p} = c - p$. By accumulating color values from the input texture along to the normalized $\vec{p}$ you can define a blurred color value for the current pixel according to its distance d to the current pixel position p:

$$rgb_{blur} = \frac{1}{n} \sum_{i=0}^{n} tex(p + \vec{p}d_i) \tag{1}$$

where d can be limited to a maximum range d_{max} and sampled within this range by fixed distances s_i. Therefore,

$$d_i = s_i \cdot d_{max} \tag{2}$$

You should use the following $n = 12$ factors s_i to determine your samples within d_{max}:

```
float s[12];
s[0] = -0.10568;    s[1] = -0.07568;     s[2] = -0.042158;
s[3] = -0.02458;   s[4] = -0.01987456;  s[5] = -0.0112458;
s[6] = 0.0112458;  s[7] = 0.01987456;   s[8] = 0.02458;
s[9] = 0.042158;   s[10] = 0.07568;     s[11] = 0.10568;
```

When defining `d_max = float(textureSize(textureRendered, 0).x) / 750.0;`, which is a factor depending on the resolution of the rendering area, the resulting scene should look similar to the figure below.

Figure 1: Result using Blinn-Phong shading and the radial blur.

Task 2: Color Space Conversion

"sRGB is a standard RGB (red, green, blue) color space that HP and Microsoft created cooperatively in 1996 to use on monitors, printers, and the World Wide Web. [...] sRGB is the current defined standard colorspace for the web, and it is usually the assumed colorspace for images that are neither tagged for a colorspace nor have an embedded color profile."[1]

The sRGB color space is the de-facto standard color space for image storage. It is nonlinear, which means that doubling a color value in this color space does not necessarily result in twice the brightness. When rendering images in computer graphics, we often want to do lighting calculations in a linear RGB color space and convert colors to sRGB at the end of the rendering loop. In this case, textures need to be converted from sRGB to linear RGB when accessing them in a shader. This is usually done automatically by setting the correct image format in the CPU side code of the graphics API (i.e., WebGL, OpenGL, Vulkan, DirectX, ...).

Your task is to use the Blinn-Phong shading code from assignment 5 and convert the final image from linear RGB to sRGB using the equation[2]

$$C_{sRGB} = \begin{cases} 12.92 C_{linear}, & \text{for } C_{linear} \leq 0.0031308 \\ 1.055 C_{linear}^{1/2.4} - 0.055, & \text{for } C_{linear} > 0.0031308 \end{cases} \tag{3}$$

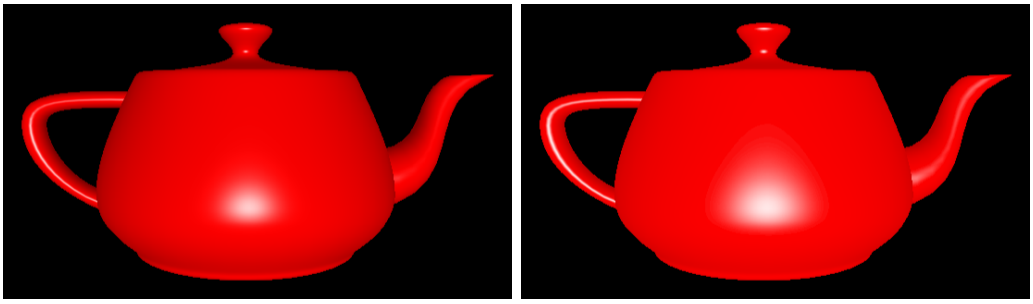

Figure 2: Left: No color space conversion. Right: Conversion of final image from linear RGB to sRGB.

As can be seen in the images above, the conversion increases the brightness in low image intensity areas.

[1] https://en.wikipedia.org/wiki/SRGB
[2] https://en.wikipedia.org/wiki/SRGB