CSc 471 - Fall 2019
Fundamentals of Computer Rendering
Assignment 2
University of Victoria

Due: October 6, 2019 at 23:55. **Late assignments will not be accepted.**

1 Overview

A look-up table (or LUT) is a table representation of a function that maps one value to another. In terms of image editing, a *colour LUT* is then a table that applies a colour transform to an image. Their importance lies in their ability to represent complex colour grades while still maintaining performance. In film and special effect production, it is very important for the artists to maintain interactive feedback when creating colour corrections. LUTs provide a simple and efficient way of implementing this.

A LUT can be represented in OpenGL as a 3D texture that is then evaluated in the fragment shader to compute the final colour of an image. The size of the LUT is generally *not* the same as the size of the image itself. As you might imagine, this would cause problems with very large images. Instead, LUTs are typically $32 \times 32 \times 32$, which means that the values are then interpolated using trilinear interpolation.

Your task is to write a single C++ program that does the following:

- Loads the provided image and renders it on a quad,
- Creates two 3D textures that represent two colour LUTs from the list shown in Section 4,
- Correctly applies the colour LUTs to the image.

You must choose *any two* LUTs from the list shown in Section 4. In addition, you must provide a way of switching between each of the LUTs and the original image.

2 Specification

For full marks, your submissions **must** have the behaviour specified below. Failure to implement this functionality will result in marks being deducted.

- Load the provided image file. This can be done at program startup as shown in the labs.

- Apply the texture to a quad and render it to the screen.
- Implement *any two* colour LUTs as 3D textures and apply them to the image.
- Provide a way to toggle between each of the two LUTs and the unmodified image.

You will be provided with an assignment bundle that will contain the following:

1. A full CMake setup for the assignment including code to generate the headers like the one seen in the lab,
2. an image file `input.jpg`[1],
3. a `main.cpp` file with a function to create the quad vertices with texture coordinates and an empty `main`,
4. a blank `assignment.hpp` header, and
5. a blank `UniformLocations.glsl` file.

Your submission must contain the following:

1. One `main.cpp` containing all the appropriate C++ code to perform the specified tasks.
2. One `assignment.hpp` containing the definitions of any auxiliary data structures, functions, etc that you may require. Note that you may leave this file blank.
3. Any vertex and fragment shaders that you require, along with the `UniformLocations.glsl` file (you may leave this blank if you wish). Note that these *must* observe the file extension convention specified in the labs.
4. In the submission comments, document how to switch between each of the LUTs and the unmodified image.

Please note that the full assignment bundle does not need to be provided in your submission, only the individual files listed above.

3 Evaluation

Your code must compile and run correctly in ECS 354. If your code does not compile or crashes at runtime, you will receive a mark of zero. This assignment is worth 8% of your final grade and will be marked out of 10.

[1]Image obtained from NASA's Image of the Day. URL: `https://www.nasa.gov/image-feature/seeing-the-world-with-a-drone`

4 Colour LUTs

This section provides a list of 4 possible colour LUTs for your assignment. You *must choose two* of them for your submission. Note that the values shown here are suggestions and you are free to change them as you see fit. For all equations shown below, c represents the input colour, and L is the LUT function.

- **Inverse LUT:** This LUT will reverse colours. The equation for the table is:

$$L(c) = (255, 255, 255) - c$$

- **Black and White LUT:** This LUT will convert the colours provided in RGB values into a single gray value. Note that in this case, all coordinates in a table entry will be the same. The equation is:

$$L(c) = 0.3c_r + 0.59c_g + 0.11c_b$$

  Where c_r, c_g, c_b are the red, green, and blue channels, respectively.

- **Brightness LUT:** This LUT will increase the overall brightness of the image. Note that any values exceeding 255 must be clamped to 255. The formula is:

$$L(c) = k \cdot c$$

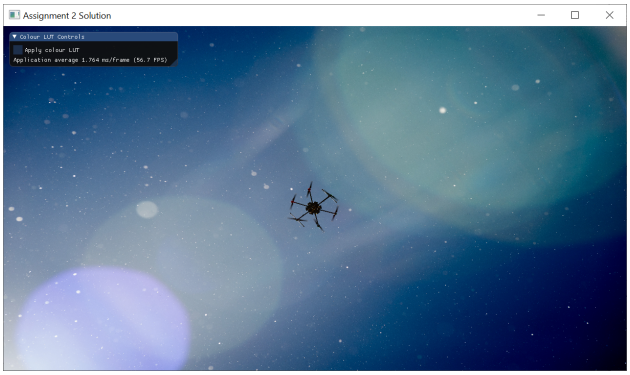
  Where k is the brightness coefficient.

- **Desaturation LUT:** This LUT will reduce the colour saturation of the image. The equation is the following:

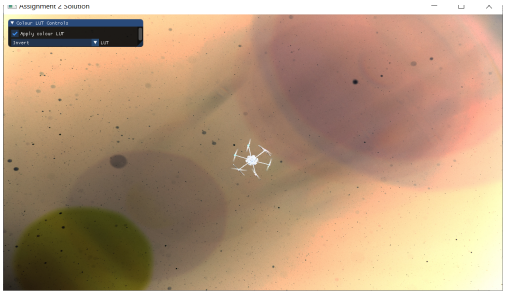
$$L(c) = c + f \cdot (l - c)$$

  Where f is the percentage of desaturation, and l is the luma value, computed with the same equation as the black and white LUT.

Figure 1 shows samples of the different LUTs along with their corresponding values.



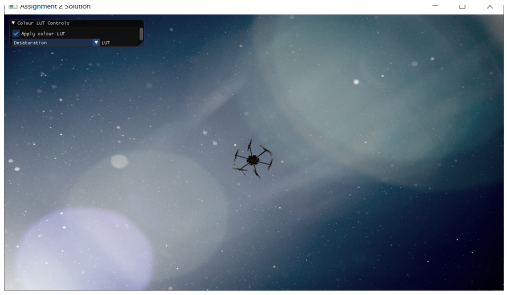
(a) Original image.



(b) Inverse LUT.

(c) Black and White LUT.

(d) Brightness LUT with $k = 2$.



(e) Desaturation LUT with $f = 0.5$.

Figure 1: Sample LUTs

5 Recommendations

5.1 Implementing the LUTs

Before you start implementing your choice of LUTs, it is a good idea to first implement the identity LUT, which simply maps every colour to itself. The main reason for doing this is that it will allow you to quickly check to see if your generation of the LUT itself is correct without having to worry about more complex formulae.

While the LUTs are 3D textures, they must be specified as a 1D buffer. The order in which you iterate over the 3 colour axes matters and will drastically change your result. For an RGB colour LUT, the buffer should be generated as a BGR buffer. This will ensure that the coordinates are correctly placed in the texture buffer. Alternatively, you may switch the order of the coordinates in the shader itself. You can use the same texture parameters that were shown in the labs, however you must ensure that your magnification filters are set to `GL_LINEAR` for the LUT to be correctly interpolated.

5.2 The Shaders

The vertex shader is a simple pass-through shader whose only task is to set the coordinates of the vertices of the quad and to forward the texture coordinates to the fragment shader. The fragment shader then requires the following inputs:

- The texture coordinates of the vertices,
- the image texture,
- the LUT texture, and
- the size of the LUT.

The fragment shader will first sample the image to obtain the colour coordinates to use in the LUT. Since trilinear interpolation is already performed when we uploaded the LUT, all we need to do is sample it. You will notice that we receive an additional input in the shader which contains the size of the LUT. The reason we need this is that the hardware texture-sampling algorithm will sample from one extreme of the data set to the other. While this is reasonable for an image, it doesn't work correctly for data sets. What we must do then is to query only the region between the outer sample's centers with the following equation:

$$\frac{s-1}{s} \cdot c + \frac{1}{2s}$$

Where s is the LUT size and c is the colour. This coordinate can then be used to sample the LUT and retrieve the correct colour.