CPSC 4040 Cartoon/Painting effect using Bilateral Filtering with Sobel Operators

I created an image converter that takes a given input image and converts the image into a cartoon/painting. This effect is achieved by first applying a bilateral filter to the image then the Sobel operator. After applying these two, a little bit of artifacting occurs, so we can then reduce this by applying a smoothing operation to the image to reduce the harshness of the artifacts while simultaneously making the artifacts look like imperfections in the painting/cartoon/.

How it works

The Sobel operators are a good way of extracting edge data in a given image while, for the most part, keeping the borders somewhat intact. You can take two 3x3 kernels, one for horizontal, one for vertical, and apply convolution with each distinct kernel to each pixel. You can then combine them with the following formula to create an image that mainly shows edges (source https://en.wikipedia.org/wiki/Sobel_operator)

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

The bilateral filter comes in by flattening out the data in the image while also, for the most part keeping the borders intact. It helps preserve sharp edges. However, a side effect of applying bilateral filtering is something called the staircase effect, which can cause images to appear as cartoons, which is the desired result. This side effect can also be amplified when bilateral filtering is applied with the Sobel operator. Below is the general formula for Bilateral filtering. (source https://en.wikipedia.org/wiki/Bilateral_filter)

$$I^{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|),$$

How it was Made

The project was made using OpenGL to render the images in real-time to see the effect. The design is very modular, with most operations separated into functions that achieve precisely one task. The code is not designed as a library since most functions have dependencies and are very order-based. Meaning you do not want to run one function until the dependency function has already been run. The modular design was for readability and ease of debugging. The codebase is also designed to be converted into a website using Doxygen if desired. (/*! Comments are turned into documentation with doxygen */

How To Run

The project was written and created with a tool called CLion, developed by a company called JetBrains. Essentially it is an IDE with many debugging tools. CLion runs and compiles code with CMake; however, in the project directory, there are a lot of CLion files that are not necessarily useful. I left them there in the event of compatibility issues; CLion is multiplatform so that it can be very quickly run in the CLion environment on any operating system with minimal issue in the event of compatibility issues outside of CLion.

Instructions

Make sure you are in the project directory (Designed on Mac)

```
> cmake CMakeLists.txt
> make
> ./Program
```

Troubleshooting

1. If it fails to compile changing the headers for openGL to (GL/gl.h) should resolve the issue
2. Make sure cmake is also installed. If you type cmake and it returns a command not found then cmake is not installed. Here are the instructions incase https://cmake.org/install/

Input:

After the program is running it's going to ask you for three inputs. Input each one by one and press enter after each input

1. Input Image Name
2. Output Image Name
3. Optimal Divisor Value

The optimal divisor value is more of a "hack" to the smoothing operator to fix some by-products artifacting. Since the image has a random amount of black pixels from artifacting, the image begins to darken after multiple filter passes. A solution to this is when averaging pixels, instead of dividing by the total pixels used as you normally would when averaging; you reduce the value just a little bit to make the pixel a tad bit brighter + reduce the effect of the black pixel being inputted. You could also ignore black pixels, but sometimes the pixel is not entirely black; sometimes, it is a very dark gray—the general rule of thumb. If the image is dark, you would want to use 7-8 or any value in between. If the image has a lot of white (clouds, for example), you want to use 9 (the original value).

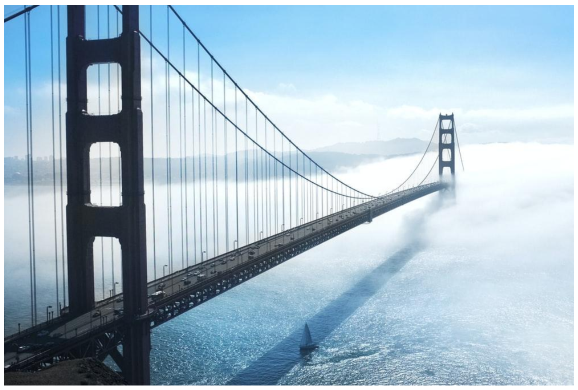
Now you should see the image being displayed on OpenGL. Here are the following options to apply
b : Bilateral Filter
s : Sobel Operator
w : Writes image with given output name that was provided during run time
ESC, Q, q : Exits the program without saving

The program is not designed for the Sobel operator to be applied before the Bilateral Filter so that the behavior may be unexpected in that scenario. For best results, press ‘b’ then let the filter load. It will produce an image that is a bit dark and spotty. It will be resolved once the Sobel Operator is applied. Then press ‘s’. If you choose to save the image, press w, and it will write the image in the executables directory

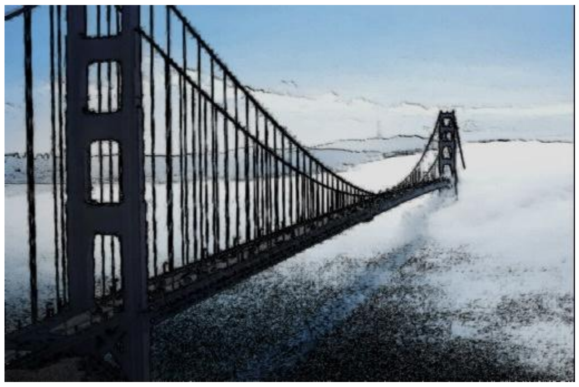
Example Output

Bridge.jpeg with Divisor Value of 9

Before



After



Issues and Modification that may be necessary to resolve in the future.

- Run time for 4K images is a tad slow (2-7 seconds)
- 4K Images also take many bilateral filter passes to achieve cartoon effect
- Animals with fur can confuse the edge detector and paint 50% of the animal with black dots

For optimal Results:

- Use on landscape, People, Cars, and avoid images that contain lots of texture, fur or noise (Carpet/Animals/Waterfalls)