User-Centered Non-Photorealistic Rendering

Cole Bateman
COMP-SCI 175: Computer Graphics
Spring 2021 Final Project

Abstract

This project constructs a user-friendly and robust non-photorealistic rendering application. A great deal of cutting edge methods are present when dealing with non-photorealistic rendering technologies, but as a whole these techniques are both not accessible to the average artist, and not in a centralized location. This tool explores the interactions between the artistic methods of tone mapping, image filtering, L0 image smoothing, and neural-network models with options for combining rendering techniques and adjusting relevant image processing parameters.

1. Introduction

Non-photorealistic rendering (NPR, and also known as rotoscoping) is a technique which allows artists to convert real-world photos and videos into a realistic, cartoonish effect for use in their artwork.

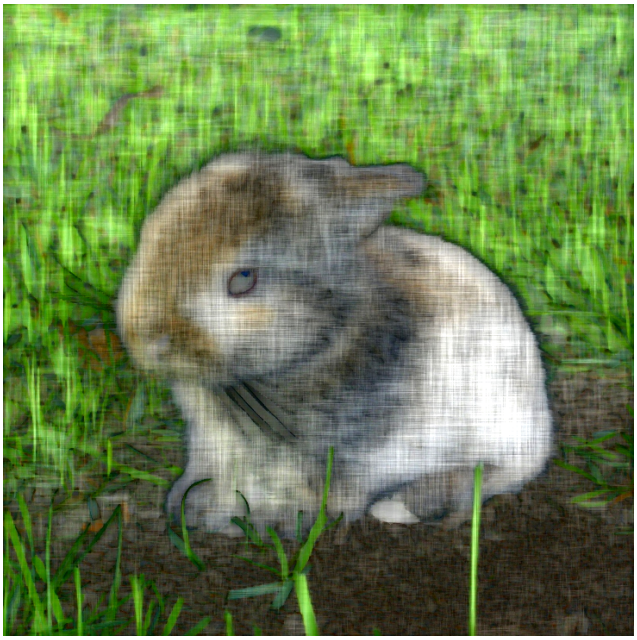

Figure 1. Stylized image using a combination of methods described throughout the report.

This project aims to prioritize user friendly interaction, variability, and modularity by pulling from HCI, Computer Graphics, and Computer Vision. Thus the project goal is to build an application that will allow users to upload photos and videos, and then apply a wide variety of photo effects in addition to art rotoscoping effects. In regards to CS 175 specifically, this project falls under the category of image processing using convolution and feature detection. Much of the functionality is implemented using OpenCV and Python 3.8.5.

The project implements numerous automatic rotoscoping techniques alongside color mapping operations, L0 image smoothing and neural-network-based foreground/background serparation, all unified by the user interface. While there are many available techniques to perform NPR-relavent effects, most of these are not accessible to the average user with control over specific modifiable parameters. The user is given the ability to select various parameters from the implemented rendering options, and returned the processed NPR image.

1.1. To Be Noted

It's important to note that this project is a continuation of the work I had done in a previous course: ES 143, Computer Vision. I made sure to receive permission from other team members before extending the functionality of this project.

I also wanted to make it clear what parts were done last semester and what was completed for CS 175 specifically. For this project, I made the following modifications and additions:

1. Redesigning the user interface, making it both more usable and more stable.

2. Implementing image smoothing using L0 Gradient Minimization [6].

3. Implementing a foreground-background separator function utilizing convolutional neural networks [3].

4. Small modifications to the other methods used to optimize the program.

Many of the other methods - such as color mapping/transforming, gamma correction, or edge detection - had already been implemented. That being said, in this report I will still be explaining each of the various methods, so as to give you the best understanding of this work as a whole.

1.2. Coding Methods

Our photo and video rendering pipeline was implemented using Python3 and can be run in the command line. We have included a README file which guides new users through the set up process; upon first run, be sure to follow the README carefully to ensure proper setup. We have also included a setup.sh file that ensures the directory's organization is correct.

The following packages are required to run the application: tensorflow, PIL, cv2, numpy, io, sys, datetime, and PySimpleGUI. tensorflow was used to implement a deep-learning image foreground-background separator and PySimpleGUI was used to implement the front end.

1.3. Computational constraints

A main focus of this project was to allow for the ability to add various filters and effects to images and videos; I found that the most interesting images came from mixing a multitude of methods together. An unfortunate result of this functionality is that the application can become difficult to run as more effects are added. This is especially true when rendering video, which requires an extremely high computational cost and might not be possible on lower end hardware. To combat this, all images and videos are scaled so that they are no wider than 900 pixels, while maintaining the original aspect ratio. Content will also be output at that scale.

2. Methods

The image processing pipeline (ignoring user-interface construction) consists of three distinct parts: color mapping operators, selected image filtering techniques, and machine learning image convolution. The final interface allows users to choose multiple image transformations within each of these categories and adjust relevant parameters for different operators.

2.1. Color mapping operators

Some of the most fundamental image processing transforms are known as point operators, where each output pixel's value corresponds to the input pixel value, potentially alongside some global information about the entire image. Examples of such operators include contrast adjustments (gamma correction), color transformations, and intensity rescaling operations (such as histogram equalization and contrast stretching). All of these operations were coded into the database to provide the users more parameters to modify output NPR images by enhancing certain features or making images more non-realistic.

2.1.1 Color transform

In the interest of providing users with an additional aspect of non-realism for NPR, we provide users with the ability to transform images using pseudocoloring. Based on the specified color map, the intensity of every pixel in a given image is mapped to a certain color. We provide users with 12 different color maps, which allow users to transform the appearance of their image. Users are further allowed to scale how much of this pseudocolored image is averaged with the original image.

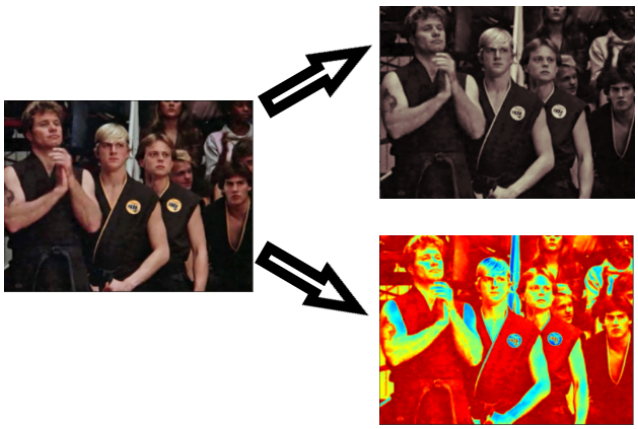

Figure 2. Example of original image (left) being mapped to 2 different pseudocolored images (right) based on different color maps.

2.1.2 Gamma correction

Gamma correction corresponds to the relationship between a pixel's intensity and its perceived luminance. Since human perception of brightness is nonlinear (such that doubling brightness on screen will not double intensity of pixels human perceive), a nonlinear gamma function adjusts the brightness of an image to match our expected change in brightness.

Image brightness corresponds to pixel intensity, and gamma correction modulates pixel intensity by nonlinear parameter γ according to the relationship $I' = I_(max) * (I/I_(max)))^{\gamma}$

Figure 3. Sample NPR compositing example. The user's input image (left) is transformed by applying a color map (middle). The user then additionally decides to add both style and sketch domain transformation filtering for a finished cartoon rendering.

2.1.3 Histogram equalization

Histogram equalization is an alternative method to maximize ideal pixel intensities in a global sense. Histogram equalization constructs a histogram of pixel intensities in a given image, and uses an equalization function to flatten this distribution of intensities in the image. By effectively spreading out the most frequent intensity values, this allows for areas of lower contrast to be emphasized. The equalization function corresponds to integrating the intensities of the histogram $h(I)$ to obtain the cumulative distribution of intensities, where we can look up the a given pixel's intensity I in order to determine its equalized intensity $c(I)$, given as follows by Szeliski 3.1.4 [4]:

$$c(I) = (1/N)\sum_{i=0}^{I} h(i) = c(I-1) + (1/N)h(I) \quad (1)$$

Histogram equalization works well for intensities values for RGB images, which corresponds to equalization for grayscale outputs. Histogram equalization for colored images can be performed by histogram equalization over the Y value of an image converted to the YUV color space, and then reverting the image back to RGB.

2.1.4 Contrast stretching

An alternative method which allows for contrast optimization in an image is contrast stretching. In contrast stretching, the highest intensity pixels are mapped to the maximum allowed intensity, whereas the lowest intensity pixels are mapped to minimum allowed intensity. This makes the input image use the entire range of values available to them, similar to histogram equalization. However, for contrast stretching, there exists a one-to-one relationship of the intensity values between the original image and stretched image. Contrast stretching and histogram equalization both produce slightly different outcomes, with different tone final results.

2.2. Filtering techniques

Filtering is an extremely common way of altering the information stored in images. Many technologies exist where users can apply and customize image filters, (Photoshop, Snapchat, etc) but the next generation of image processing tools will also include these methods as they both create cool effects, and are the fundamentals of more advanced methods.

2.2.1 Convolution and correlation

The fundamental idea behind filtering is the creation of a 'kernel'. The kernel represents a mapping of pixel values to a weighted average of their neighboring pixels. Convolutional filtering is the process where pixels are weighted as the mean of their neighboring pixels, and correlation filters also use a kernel-style mapping but are not necessarily the weighted average of the neighboring pixels and can apply different weightings (ex. Convolutional filter rotated by 180 degrees). Mathematically the discrete convolution operator is given as:

$$f[x] * g[x] = \sum_{n=-\infty}^{\infty} f[k]g[x-k] \quad (2)$$

Where the function $f[x]$ is the image processing function we are trying to apply, and $g[x]$ is the function that is defined by pixel values of the source image, and the result is a filtered image. Some examples of functions are explored below.

2.2.2 Gaussian blurring

One of the most common image processing functions is a Gaussian Filter, it is used to create a smoothing effect throughout the image by removing noise that is assumed to come from a Normal distribution. The distribution used for sampling weights in this case is the 2D 0-Mean discrete Gaussian function

$$g[i,j] = e^{\frac{-(i^2+j^2)}{2\sigma^2}} \quad (3)$$

Here, σ determines the strength of smoothing: gaussian blurring replaces a pixel's value with the weighted sum of its neighbors, where the size of this neighborhood (and thus how much impact neighbors have in "averaging" a pixel value) is controlled by σ. Evidently, the gaussian acts as a low-pass filter in the spatial domain. This is easy to implement, and creates solid results that you can see in the gallery below.

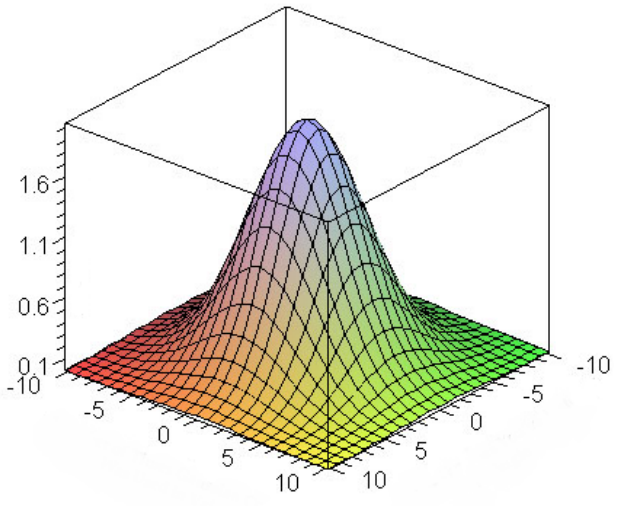

Figure 4. Visualization of a general 2D Gaussian Filter.

2.2.3 Edge detecting filter

When looking to determine features of an image, the best way to determine points of interest is to look at the edges. The filter that is used in this implementation of is centered around a filter called the "Laplacian". It can be used in conjunction with Gaussian Smoothing to reduce high frequency noise that would have unwelcome effects on the overall filter and it is implemented generally following:

$$LoG(x,y) = \frac{1}{\pi\omega^2}(1 - \frac{x^2+y^2}{2\omega^2})e^{\frac{-x^2-y^2}{2\omega^2}} \quad (4)$$

This operator essential calculates the second derivative of the image space. An 'edge' corresponds to a change in intensity between neighboring pixels, and the second derivative format will create a really high output result after being applied.

2.2.4 L0 Gradient Minimization

My smoothing functionality was inspired by Xu et al [6] in their implementation of a L0 gradient minimization algorithm. The effect is achieved by simultaneously sharpening major edges and eliminating a manageable degree of low-amplitude structures within the image; essentially, we

Figure 5. An example of the smoothing, and only the smoothing, in action.

Figure 6. With the addition of other effects, we get a really stylized image.

calculate a gradient. If we denote S as the computed result and I as the input image, we solve for the gradient $\nabla S_p = (\partial_x S_p, \partial_y S_p)^T$ at each pixel p where p is calculated as the color difference between neighboring pixels in the x and y directions. The problem can be broken down into two subproblems:

1. Computing S.

2. Computing (h, v), which are auxiliary variables that correspond to $\partial_x S_p$ and $\partial_y S_p$ respectively.

Subproblem 1 corresponds to the minimization of

$$\left\{ \sum_p (S_p - I_p)^2 + \beta((\partial_x S_p - h_p)^2 + (\partial_y S_p - v_p)^2) \right\} \quad (5)$$

while for subproblem 2 we calculate

$$\min_{h,v} \left\{ \sum_p (\partial_x S_p - h_p)^2 + (\partial_y S_p - v_p)^2 + \frac{\lambda}{\beta} C(h,v) \right\} \quad (6)$$

The implemented algorithm is thus:

Algorithm 1: L_0 Gradient Minimization

Input: image I, smoothing weight λ, parameters β_0, β_{max}, and rate k
Initialization: $S \leftarrow I$, $\beta \leftarrow \beta_0$, $i \leftarrow 0$
Output: image S
while $\beta \geq \beta_{max}$ **do**
  With $S^{(i)}$, solve for $h_p^{(i)}$ and $v_p^{(i)}$ using subproblem 1 eq.
  With $h^{(i)}$ and $v^{(i)}$, solve for $S^{(i+1)}$ using subproblem 2 eq.
  $\beta \leftarrow k\beta_{max}$
end

2.2.5 Bilateral filter

Bilateral filtering can be used to create sharper cutoffs between regions of intensity for a cartoonish effect, effectively combining the smoothing of pixel values from gaussian blurring while preserving edges/color boundaries to retain image definition. This generation of "false edges" is sometimes considered a limitation of bilateral filtering, but serves well for NPR-driven image transformation. The derivation of this filter (from Szeliski) is as follows. The output pixel value is formed by first taking a weighted sum of neighboring pixels.

$$p[x,y] = \frac{\sum^{k,l} f[k,l]g[i,j,k,l]}{\sum^{k,l} g[i,j,k,l]} \tag{7}$$

The weights are uniquely determined by using a combination of two more distributions, here being Gaussian distributions, taking the form of range and spatial kernels. The range kernel smooths the intensity change between the neighboring pixels given by:

$$r[i,j,k,l] = e^{\frac{(i-k)^2+(j-l)^2}{2\sigma_r^2}} \tag{8}$$

This is then multiplied against another function that smooths the difference of coordinates, which is called a spatial kernel (again Gaussian).

$$s[i,j,k,l] = e^{\frac{|f[i,j]-f[k,l]|^2}{2\sigma_s^2}} \tag{9}$$

Which sum to the weight function. We now arrive at the complete definition of the filter. For coordinate x of the current pixel to be filtered, we have:

$$B[x] = \frac{1}{W_p} \sum^{|x|<1} p[x_i]r[|p[x_i]-p[x]|]s[|x_i-x|] \tag{10}$$

Where the normalized term of W_p is equal to

$$W_p = \sum^{|x|<1} r[|p[x_i]-p[x]|]s[|x_i-x|] \tag{11}$$

This is the definition of the entire filter, and the parameters of σ_r and σ_s present in the weighting term provide the degrees of freedom for the user. These parameters smoothen larger features, and flatten the Gaussian respectively. We encourage the reader to give them a try using the tool to create their desired effects. This bilateral filter effectively uses

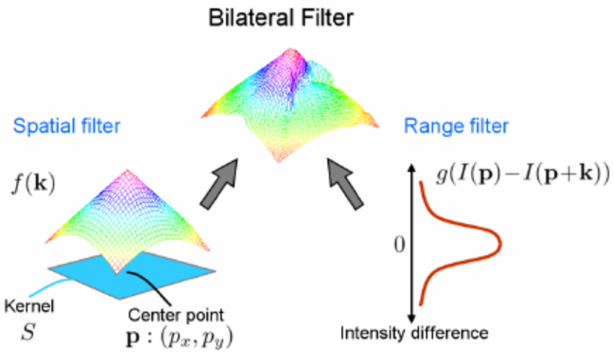

Figure 7. Visualization of a Bilateral Filter

a spatial Gaussian to ensure that only neighboring pixels are considered for blurring and a range Gaussian of intensity difference makes sure that only those pixels with similar intensities to the central pixel are blurred, in order to ultimately preserve sharp edges while blurring (since image edges present large variance in intensity). [5]

2.2.6 Domain Transform for edge-aware filtering

For 2D images, bilateral filtering can be interpreted as a 5D spatially-invariant kernel (3 color channels alongside pixel coordinates), whose response decreases as the distances among pixels increase in 5D. Given the high dimensionality of the filtering process, bilateral filtering presents high computational costs. In interest of a user-focused interface, we have included an additional faster edge-preservation filtering technique which can provide NPR relevant effects in linear time to the user (the user can see the filtered image result in real time so that they have better control over implementing their NPR vision). Gastal et al. provide a domain transformation for edge preserving smoothing which holds close results to bilateral filtering but is significantly faster [2], providing multiple stylization options relevant to NPR.

The domain filter defines an isometry (transformation where distances are preserved) in a space of lower dimensionality, such that many spatially-invariant filters are still edge-preserving but the computational cost of determining filter strength has decreased. For a given 1D signal $I : \Omega \rightarrow \mathbb{R}^2$, I defines a curve C in $\mathbb{R}^2$ by the graph $(x, I(x))$, for $x \in \Omega$. For any two points u and w in Ω, $w \geq u$, we define the distance between them in the domain

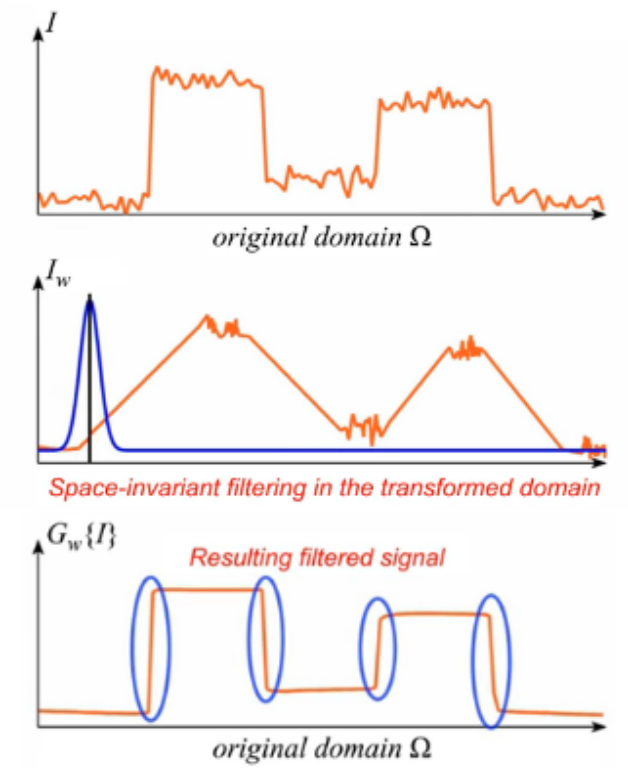

Figure 8. Pipeline demonstrating the image transform from original Ω domain to transformed domain for filtering, and then reverted back to original domain.

transform ct as:

$$ct(w) - ct(u) = \int_u^w |1 + I'(x)|\, dx \tag{12}$$

such that the geodesic distance between all points on the a curve of C is preserved even following dimensionality reduction. Scaling this finding to 2D RGB images, the domain transform for all channels is:

$$ct(u) = \int_0^u (\sigma_r/\sigma_s) \sum_{k=1}^{c} |1 + I'_k(x)|\, dx \tag{13}$$

Ultimately, by applying the domain transformation to high dimensional space, we protect detected edges in the frequency domain, allowing us to then efficiently smooth over the signal without disturbing the edges. Reverting the signal to the original Ω domain gives a similar but faster result compared to bilateral filtering.

This domain transform provides various real-time applications. While the domain transformation filtering itself offers similar rotoscoping to bilateral filtering, the subtraction of this module with the original image provides "detail enhancement" capabilities. Meanwhile, the gradient of the filtered image superimposed to the filtered image itself produces additional high-contrast edges around important features. Additionally, by assessing the transform neighborhoods in the pixel domain, we can resemble pencil sketch functionality. All of these can provide useful NPR functionalities.

2.3. Machine learning methods

Artificial Neural Networks are well suited for image manipulation because they are able to take in and analyze the large amounts of data that represent pictures. They are made up of an array of connected 'neurons' which take input data and apply an activation function, then pass along the result. The network "learns" the optimal parameters to feed into the activation functions through a back-propagation process, and a prediction is made after a number of iterations. However, there are some problems with the regular neural

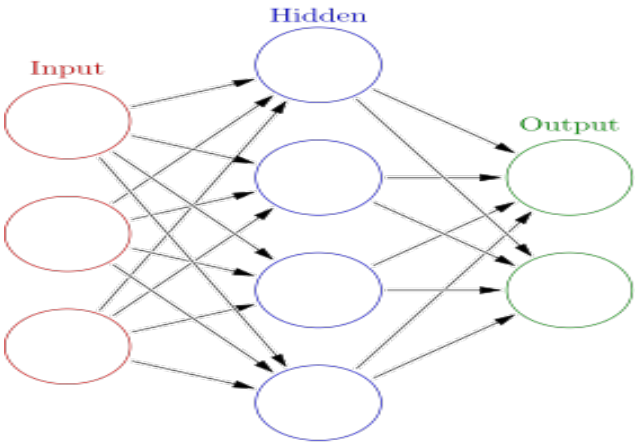

Figure 9. Artificial Neural Network

network model when it comes to image processing. Firstly, most interesting images contain at least hundreds of pixels and three channels, which will result in millions of parameters and a high likelihood of overfitting. Second, the process becomes increasingly expensive computationally, and is generally harder to work with along the lines of interpreting results and debugging.

2.3.1 Convolutional Neural Networks

For this project, I implemented a Convolutional Neural Network. This network solves the above problems by not having all of the neurons in a single connected system. Instead, the connection is divided up into three stages. The first stage is the convolutional operator that creates a map of probabilities that a goal feature is being selected for. The second layer consists of pooling, which usually consists of lowering the dimension of the data by just taking the highest value from each pixel area, a process called 'Max Pooling'. This allows us to retain the most important features and further

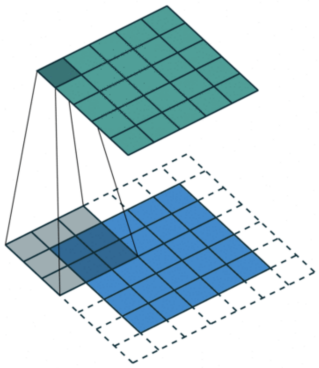

Figure 10. Convolution Stage

reduce the complexity. Finally, the results of the second

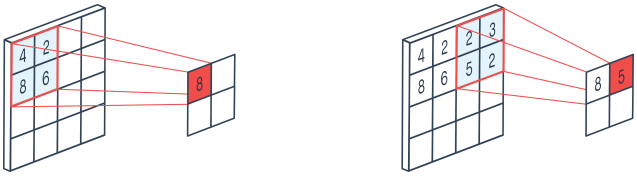

Figure 11. Pooling Stage

stage are fed into a smaller fully connected neural network as described above and a prediction is made about where the goal features are on the image if any are present [1].

Using this model, I implemented a foreground background separator function. There are many reasons why this would be desirable, and combining it with the other filtering techniques one the separated parts creates incredible visual effects.

The feature that this model is trying to extract is the edges around the foreground. The method selected is derived from a github author [3], which in the pre-processing uses two CNN models to detect boundaries, cut them out, and piece back foreground into the resulting image. A smoothing filter is applied that gives a cleaner result.

Figure 12. Example of the CNN's used to distinguish between foreground and background for compositing applications.

3. Results and Discussion

This application is comparable to mobile apps with image filtering functionality, like Instagram or Snapchat. It differentiates itself, however, in its flexibility, the ability to layer effects, and the cutting-edge nature of its implementation. For example, within Instagram's image editing suite, you are only able to change the color tone and structure of your image. This tool allows for the addition of one or more complex effects on top of similar functionality to Instagram.

I consider this project to be a successful attempt at implementing the stated goal of making image manipulation accessible. We were able to combine a variety of different tone mappings, image filters, and deep-learning tools under a usable interface, which makes unique NPR image editing accessible to anyone.

3.1. Limitations

Our application is not without its limitations. First and foremost, the processing power necessary for a smooth experience in the interface (with both images and video) is quite large. There are some combinations of effects that can cause the system to crash, namely combinations including Threshold, Canny, Hue, or Enhance. We believe this is due to the change in color channels these filters create in the input image. Finally, some functionality is not usable between both image and video processing, either due to processing constraints or unfinished implementation. For example, you cannot run the background-foreground separation utility on a video without the process taking extremely long.

3.2. Usability

One of the major goals within this project was to create a system that was usable and flexible for the end user. While more UI/UX testing is required, I believe the project accomplishes this goal due to three factors:

1. The ability to adjust the parameters on all filters and applicable tone mappings

2. The inclusion or exclusion of relevant menu buttons, depending on the input. This helps to avoid confusion concerning unimplemented features.

3. The simplistic layout, with no functionality hidden.

However, the processing power requirement can sometimes impact the immediacy of the program. For example, when using L0 gradient minimization, the responsiveness of the application is greatly affected.

4. Conclusions

In conclusion, this is a project that has allowed me to not only explore the world of computer graphics, but more read-

ily understand it. If you would like to watch some videos made with the program, you can do so at here and here.

I wanted to thank the CS 175 staff for an incredibly educational semester and for the opportunity to explore interesting and applicable topics.

References

[1] *Convolutional Neural Network: How to Build One in Keras amp; PyTorch*. URL: https://missinglink.ai/guides/neural-network-concepts/convolutional-neural-network-build-one-keras-pytorch/.

[2] Eduardo S. L. Gastal and Manuel M. Oliveira. "Domain Transform for Edge-Aware Image and Video Processing". In: *ACM TOG* 30.4 (2011). Proceedings of SIGGRAPH 2011, 69:1–69:12.

[3] Susheelsk. *susheelsk/image-background-removal*. URL: https://github.com/susheelsk/image-background-removal.

[4] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 1848829345.

[5] C. Tomasi and R. Manduchi. "Bilateral filtering for gray and color images". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 839–846. DOI: 10.1109/ICCV.1998.710815.

[6] Li Xu et al. "Image Smoothing via L0 Gradient Minimization". In: *ACM Transactions on Graphics (SIGGRAPH Asia)* (2011).