Aurora Jitrskul
Dr. Nima Kalantari
CSCE 748 Section 600
April 11th, 2025

Color Quantization & Dithering for Pixel Art - CSCE 748 Final Project

Description

The purpose of this project was to replicate and experiment with a popular post-processing (and sometimes pre-processing) effect in video games to pixelate the scene. This effect is most well-known from the award winning game Dead Cells where it was utilized to render out 3D models and animations as 2D pixel art sprites and sprite sheets for efficient animation however, it is also useful as a stylistic post-processing effect in several other games as well. As such, our goal was to replicate this effect not only for its original intended use, but also to serve as an artistic finish for photographs, most notably those of scenic landscapes and backgrounds.

Methods

To create the stylized pixel art look that is reminiscent of the iconic games that I drew inspiration from, I knew that the core of my algorithm would rely on downsampling in such a way that color value differences are accentuated between pixels. For this reason, I decided to utilize point filtering in order to sample every other pixel at different increments depending on a constant value and then upsample back up using these selected pixel values to the original image size. For this to work most effectively for every possible image dimension it is best if this resize factor constant value is maintained at a power of two. Once this simple color quantization algorithm was implemented, I also had intended to implement color remapping of the pixel values to a user inputted color palette. To do this, I first converted my image to grayscale using the luminance method which takes into account the human visual system's natural sensitivities towards certain color values. The equation is as follows:

$$grayscale = (0.299 * red) + (0.587 * green) + (0.114 * blue) \quad (1)$$

Now that the image was converted to grayscale (meaning it is 2-dimensional at this point), I could then round each value found within the image which was within the range of [0, 1] to their corresponding nearest neighbor based on a constant parameter for the number of colors which is inputted by the user. In this way, if *n* is the number of colors desired by the user, I used the following equation to remap the image.

$$color = \frac{\lfloor grayscale * (n-1) + 0.5 \rfloor}{n-1} \quad (2)$$

After this remapping function, there are only n-many color values left in the image which I can easily loop through and index using NumPy's array indexing to change the values of. It is important to note that on this step the color palette is mapped in from left to right or starting from index zero if inputted as a list where the first color is mapped to the darkest regions and the last color is mapped to the brightest regions. If I simply use this mapping however, an artifact called color banding can sometimes occur which is where colors appear to create band-like shapes in an image that look unnatural and can disturb the overall image aesthetically. Therefore, to combat this I implemented intentional noise in the form as ordered/bayer dithering before the grayscale operation. For my purposes I used the 2 x 2 and 4 x 4 Bayer Matrix which I had the algorithm select based on image size for efficiency and have included below.

$$\mathbf{M_2} = \frac{1}{4}\begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix} \qquad \mathbf{M_4} = \frac{1}{16}\begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

Figure 1. 2x2 & 4x4 Bayer Matrices for Ordered/Bayer Dithering

Then, following ordered/bayer dithering I created a noise 2-dimensional matrix which is equal in dimension to that of the image with only one color channel and mapped each coordinate of this noise matrix to a value in the Bayer Matrix by modding the x and y coordinates by the dimensions of the Bayer Matrix. Then, this noise matrix was subtracted by 0.5 uniformly across all pixels and multiplied by a user-defined parameter called spread which controlled the level of noise introduced to the image. Finally, the noise matrix was added to the final image which was then normalized to the [0, 1] range. This final image was then put through the aforementioned grayscale and color remapping functions in order to achieve a color-banding free image.

The last thing I implemented as part of this project was a feature I added in hindsight because of the loss of certain semantic information. When there were many objects with harsh yet small edges in the scene or just a very busy scene in general these details usually became lost as a result of the downsampling and could not be shown in the final pixel art. In order to accentuate these details, I decided to add sharpening as the very first stage of this pipeline. In this way, I used a simple approximation of the laplacian operator which had the following kernel.

0	-1	0
-1	5	-1
0	-1	0

Figure 2. Laplacian Operator Sharpening Kernel

Which I then convolved with the image, using symmetric image replication for the boundary conditions. This convolution occurred on each color channel individually which, with normalization, produced the initial sharpened image. As such, the final pipeline for my pixel art algorithm including all of these processes was as follows.

Figure 3. Final Pixel Art Algorithm Pipeline

Results

The input images that produced all of my resulting images are as follows.

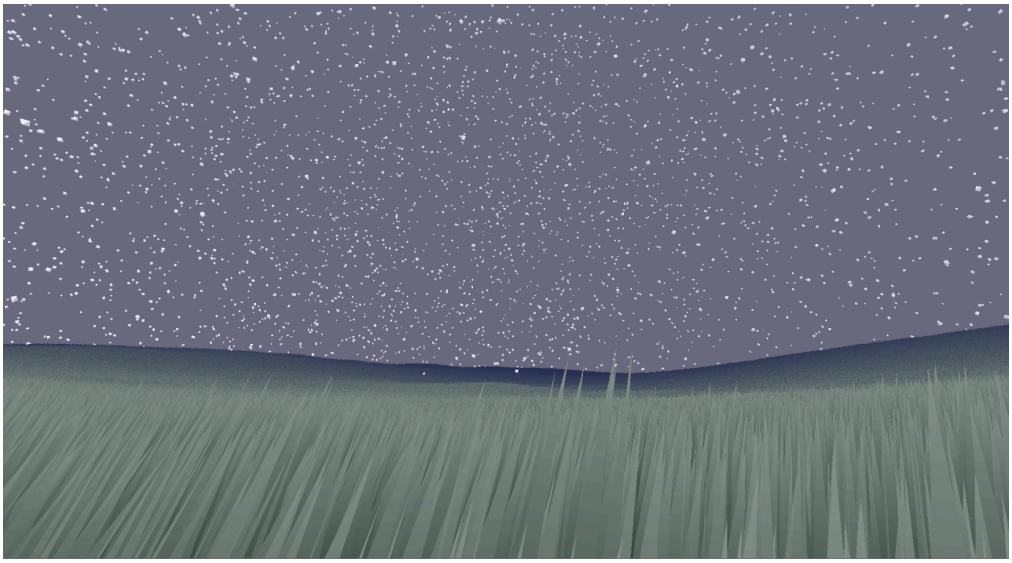

Figure 3. Plain Input Image

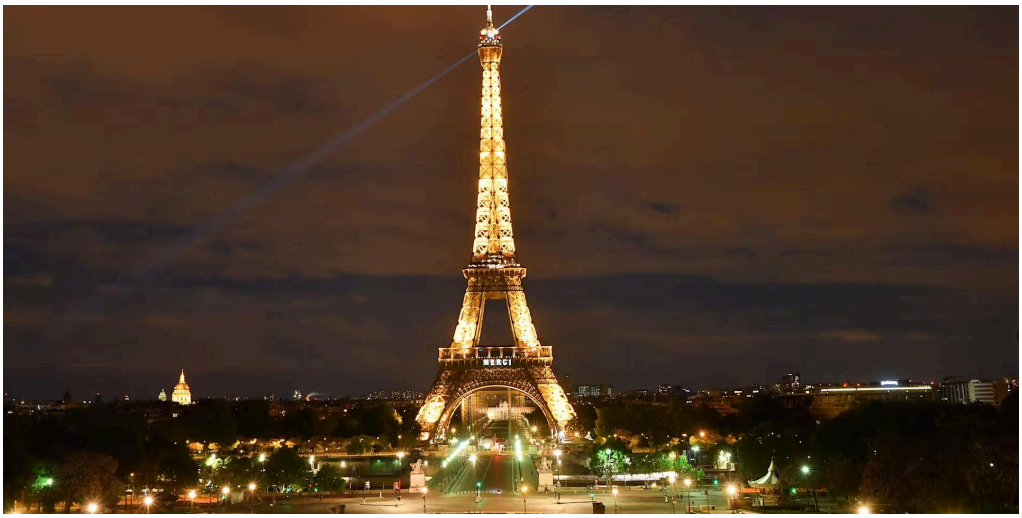

Figure 4. Eiffel Tower Input Image

Figure 5. Wave Input Image

To start with, I am going to just display the different operators at different parameter values on the base eiffel tower image to show their different effects. Starting with the first operation in this algorithm pipeline, sharpening, using the discrete laplacian operator I ended up only using a constant parameter factor for sharpness of ≤ 0.3 because higher values tended to saturate the colors too much towards white. The results for the eiffel tower image are below.

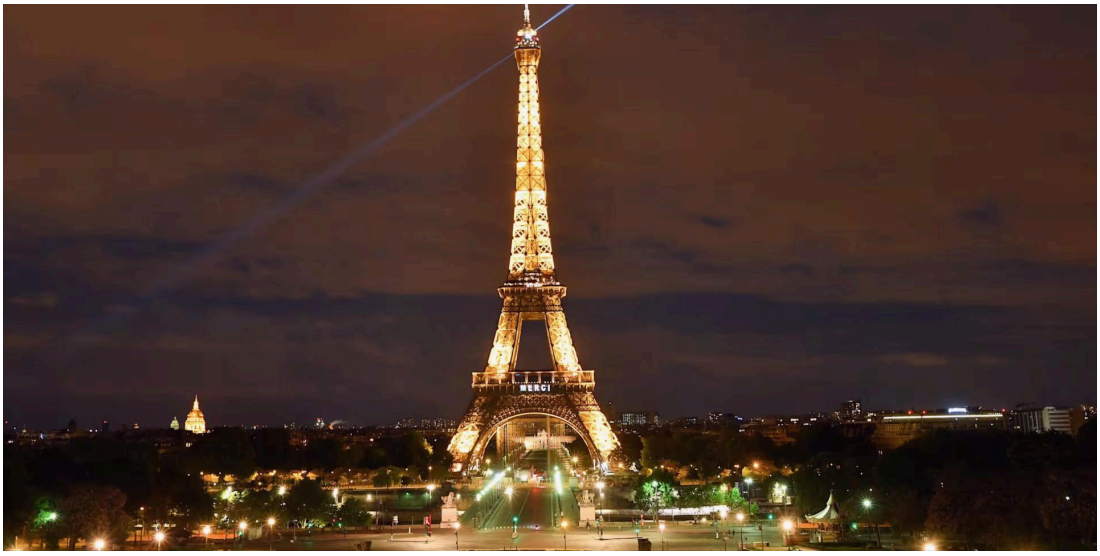

Figure 6. 0.1x Sharpness on Eiffel Tower

Figure 7. 0.3x Sharpness on Eiffel Tower

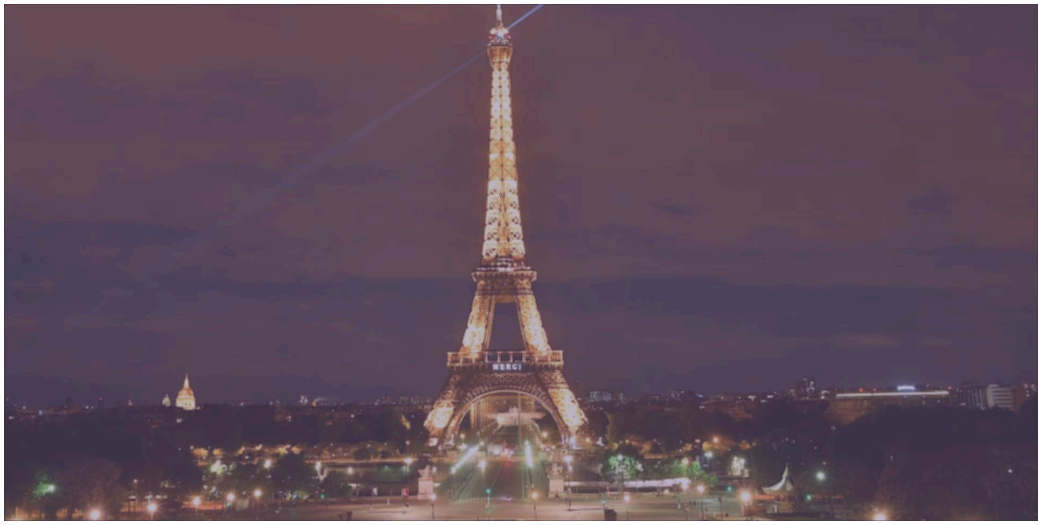

Figure 8. 0.6x Sharpness on Eiffel Tower

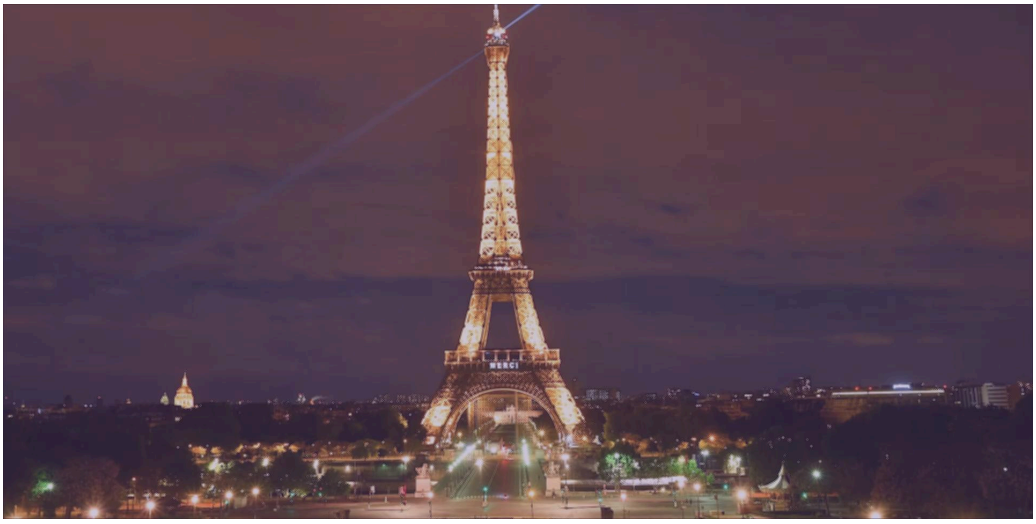

Figure 9. 1.0x Sharpness on Eiffel Tower

The next operation in the pipeline was point filtering in order to downsample and rescale the image back to its original size in order to achieve the actual pixel art effect. The effect was accentuated at higher resize factors however, this led to a loss of semantic information and made the images less recognizable. I noticed the best results at resize factors of 8 or 16 (indicating sampling only every 8 or 16 pixels of the image) however, I have included several factors below and this can vary slightly based on image size. It is important to note that this down sampling method only works well for powers of two.

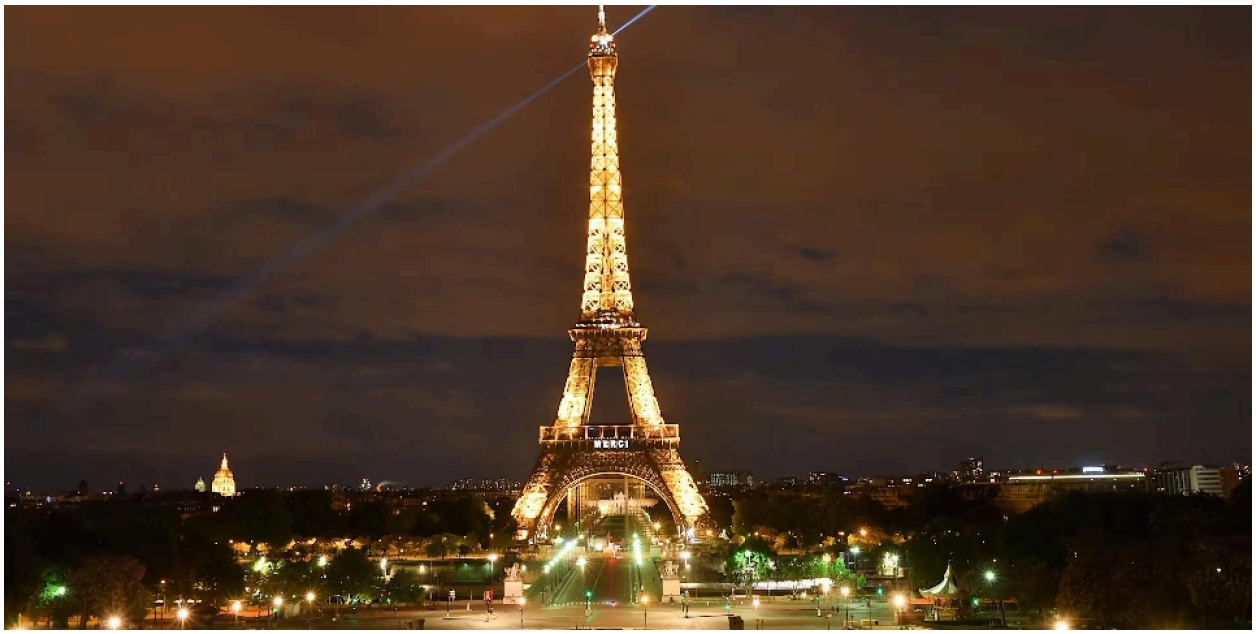

Figure 10. 2x Resize Factor on Eiffel Tower

Figure 11. 4x Resize Factor on Eiffel Tower

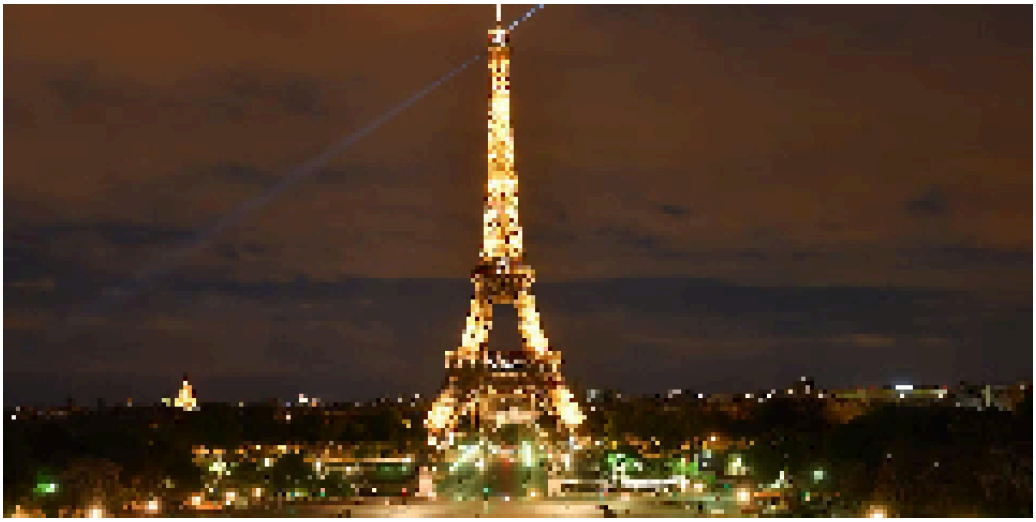

Figure 12. 8x Resize Factor on Eiffel Tower

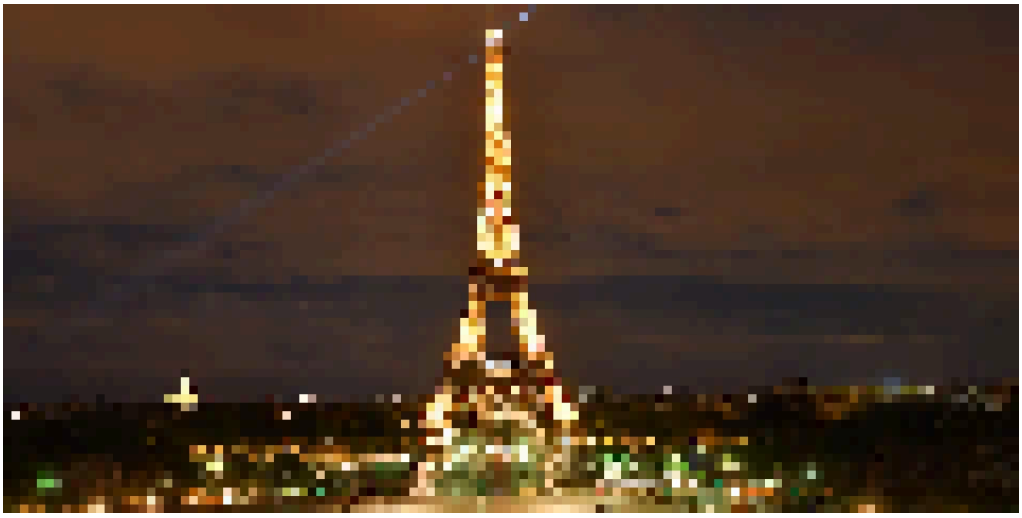

Figure 13. 16x Resize Factor on Eiffel Tower

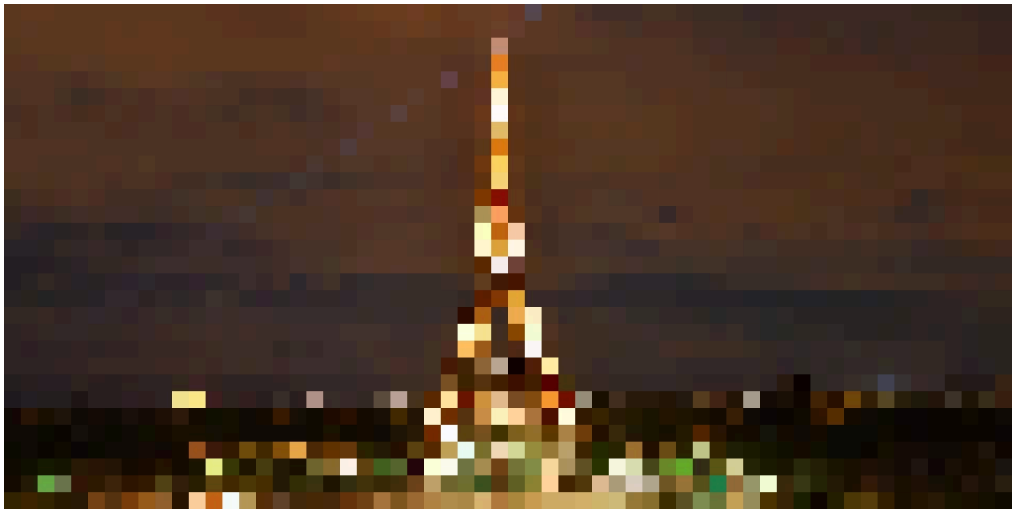

Figure 14. 32x Resize Factor on Eiffel Tower

Figure 15. 64x Resize Factor on Eiffel Tower

After applying this pixel art filter the next operation to be applied was Ordered/Bayer Dithering in order to add intentional noise to prevent color banding artifacts in the final output image. This process included a spread factor which essentially influenced the magnitude of the dithering effect on the image. The best experimental spread factor I discovered was around 0.2 to prevent color banding artifacts without too heavily influencing the final output too greatly over the other effects. The direct output of dithering on the input image at different spread values are displayed below.

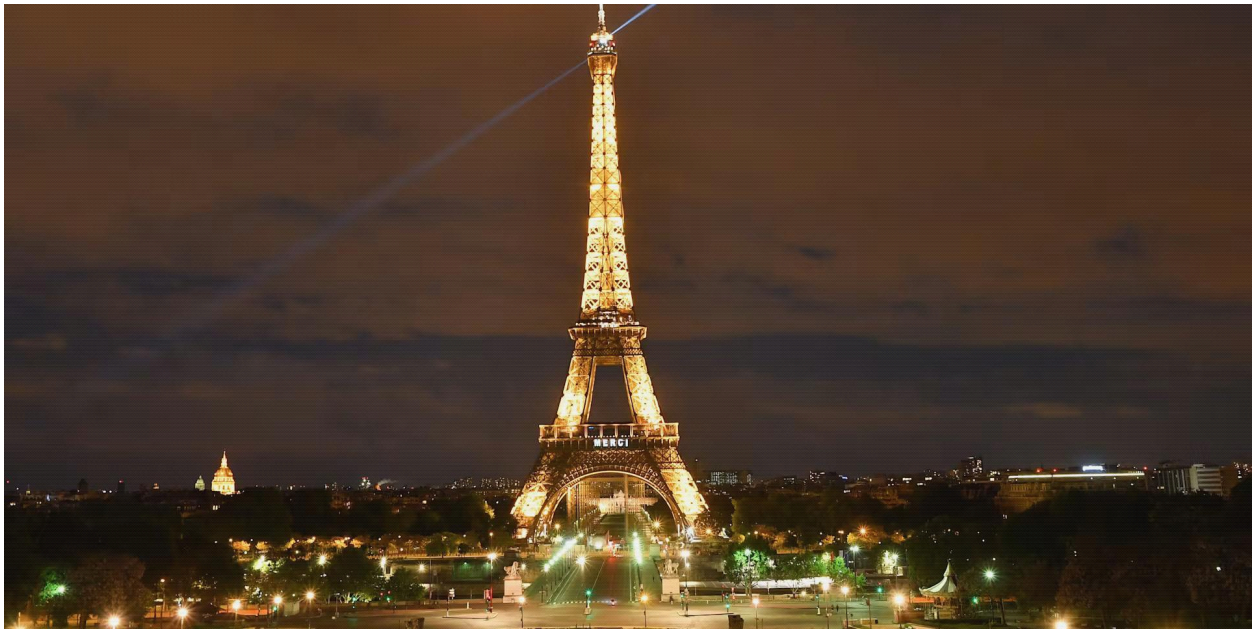

Figure 16. 0.1x Spread Factor on Eiffel tower

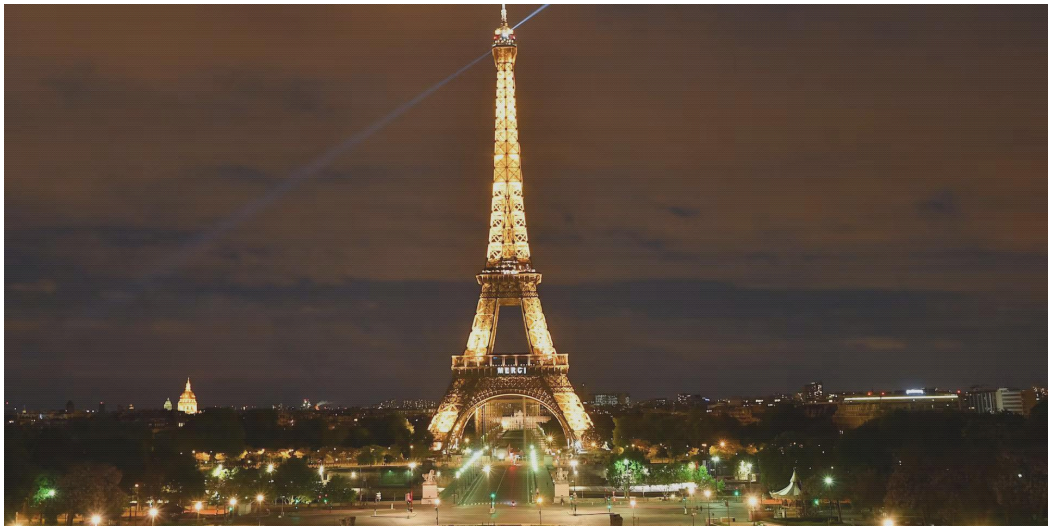

Figure 17. 0.3x Spread Factor on Eiffel Tower

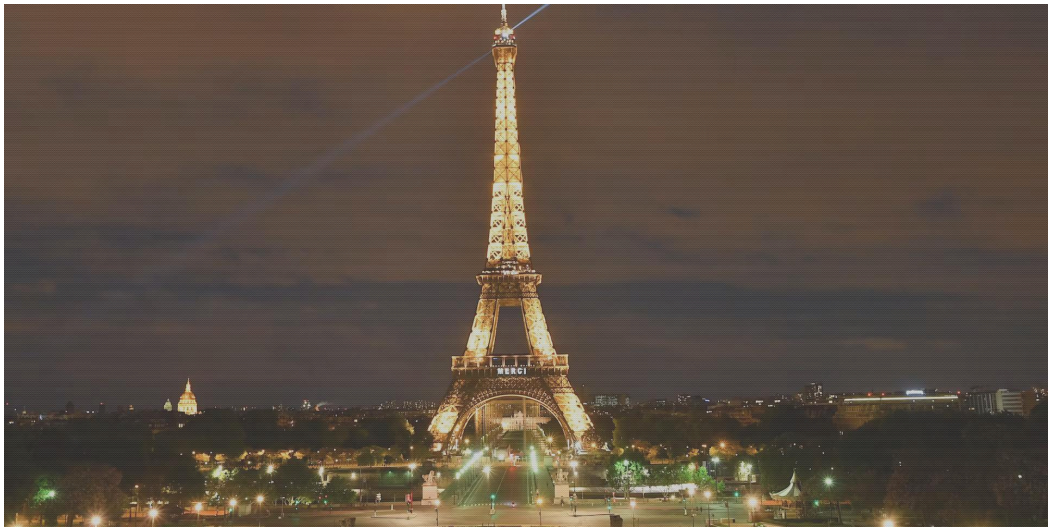

Figure 18. 0.6x Spread Factor on Eiffel Tower

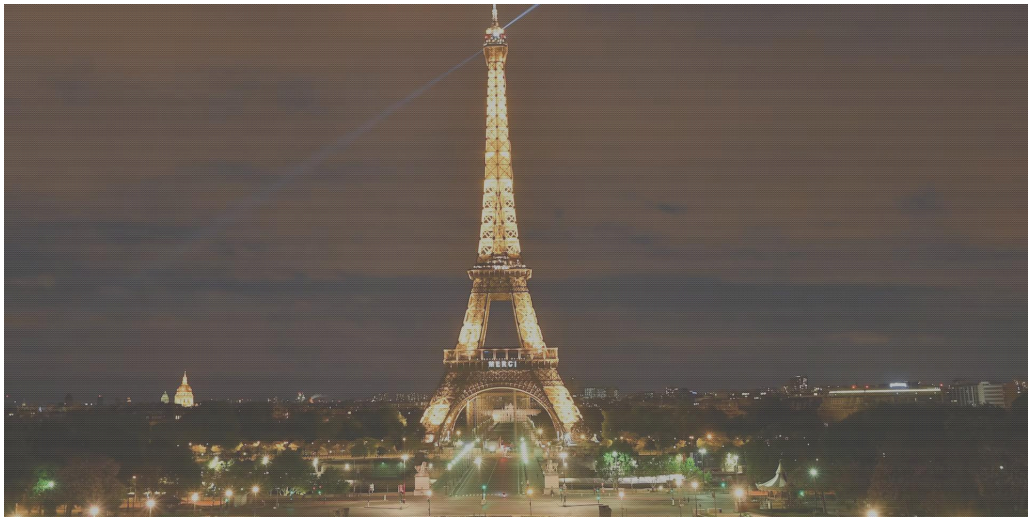

Figure 19. 1.0x Spread Factor on Eiffel Tower

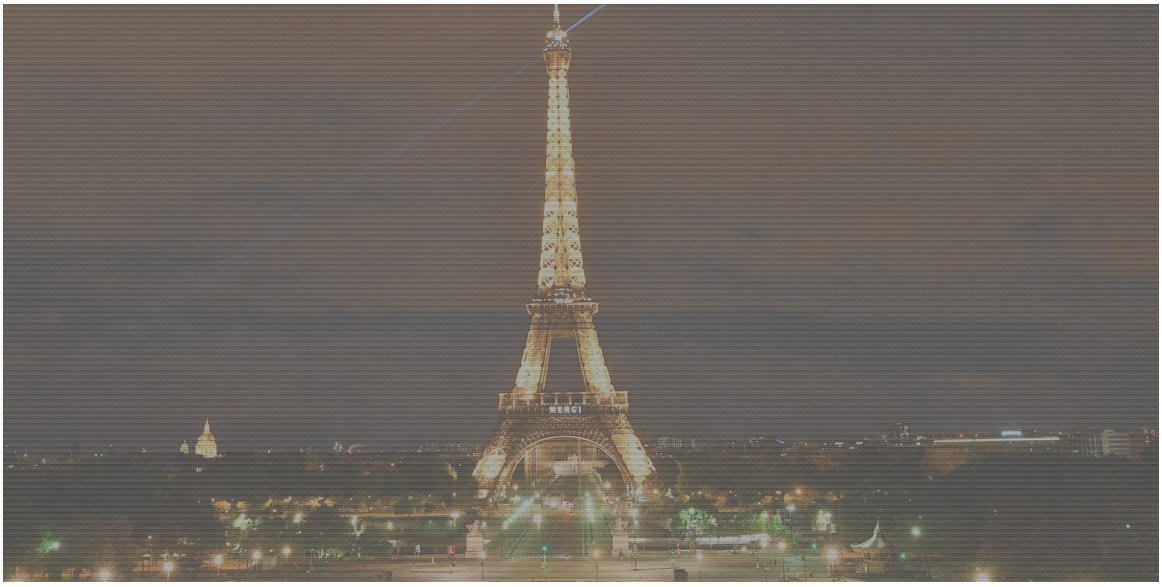

Figure 20. 2.0x Spread Factor on Eiffel Tower

Next after dithering occurs the grayscale conversion. Again, as mentioned above I used the luminance method for doing this in order to match the human visual system color sensitives most optimally. As there is no constant parameter for this I have just included the grayscale image of the eiffel tower using the luminance method below for demonstration purposes.

Figure 21. Luminance Grayscale Conversion on Eiffel Tower

Finally, after grayscale conversion, the final color palette swap is computed on the grayscale image in order to swap the colors of the image to any desired color palette. To show this, I will include the color palette (which also shows the number of colors the image is being reduced to) as well as the output image computing this step directly from the original image of the eiffel tower. It is also important to note that the order of the colors in the color palette matters

as they are mapped from left to right or from index 0 to the last index starting from the darkest color to the brightest color. As such, I will also show shuffled versions of the same color palette to display how this ordering can shift the overall output.

Figure 22. Autumn Leaves 4-color palette

Figure 23. Autumn Leaves Color Swapping on Eiffel Tower 1st Ordering

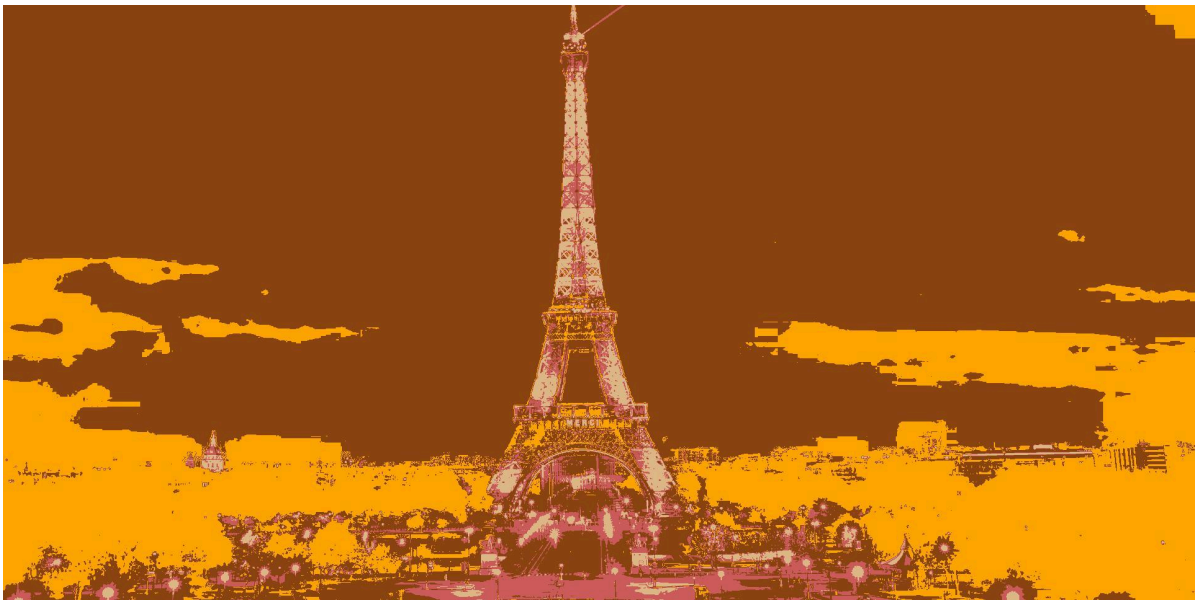

Figure 23. Autumn Leaves Color Swapping on Eiffel Tower 2nd Ordering

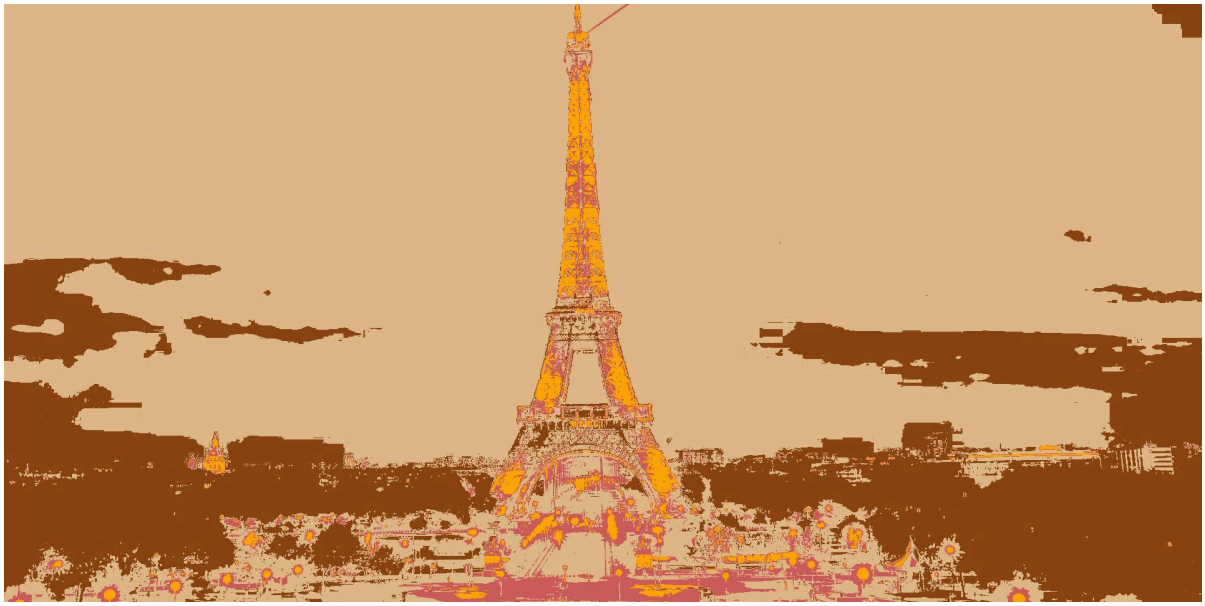

Figure 24. Autumn Leaves Color Swapping on Eiffel Tower 3rd Ordering

Figure 25. Cyberpunk 8-color palette

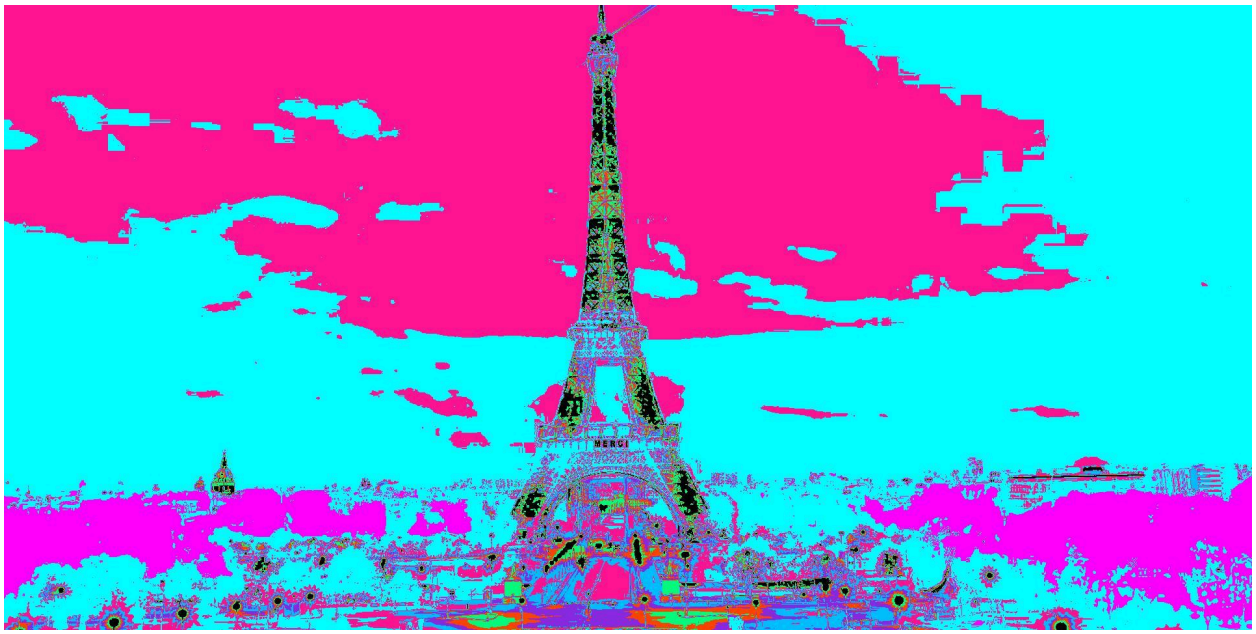

Figure 26. Cyberpunk Color Swapping on Eiffel Tower 1st Ordering

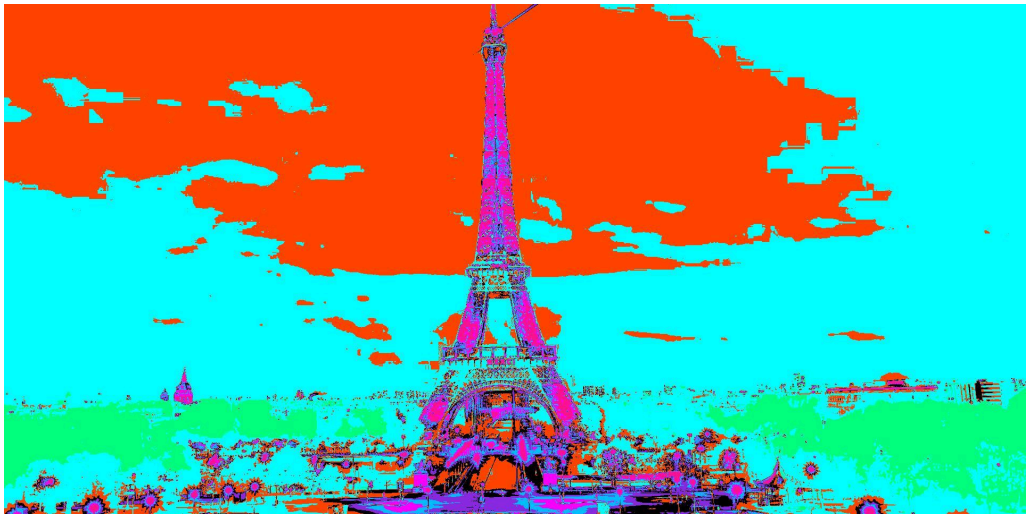

Figure 27. Cyberpunk Color Swapping on Eiffel Tower 2nd Ordering

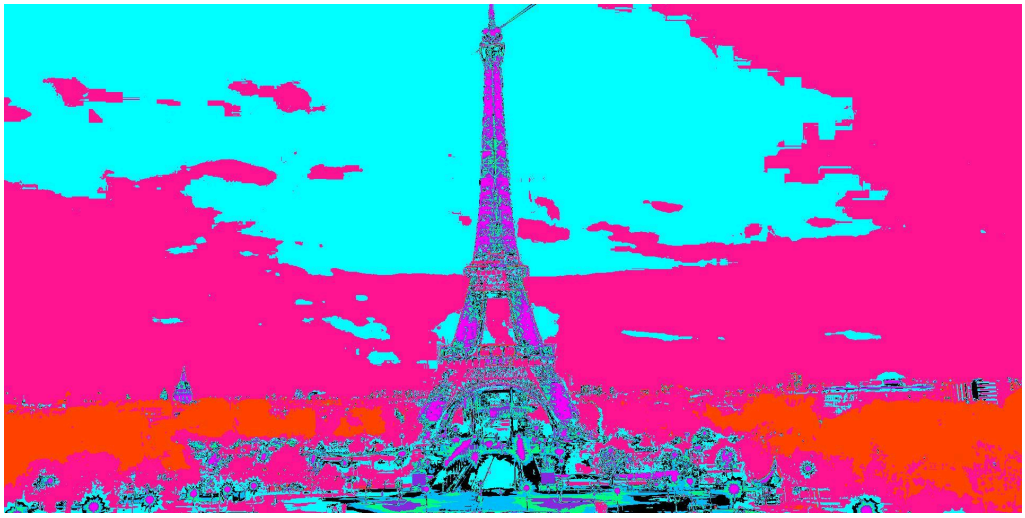

Figure 28. Cyberpunk Color Swapping on Eiffel Tower 3rd Ordering

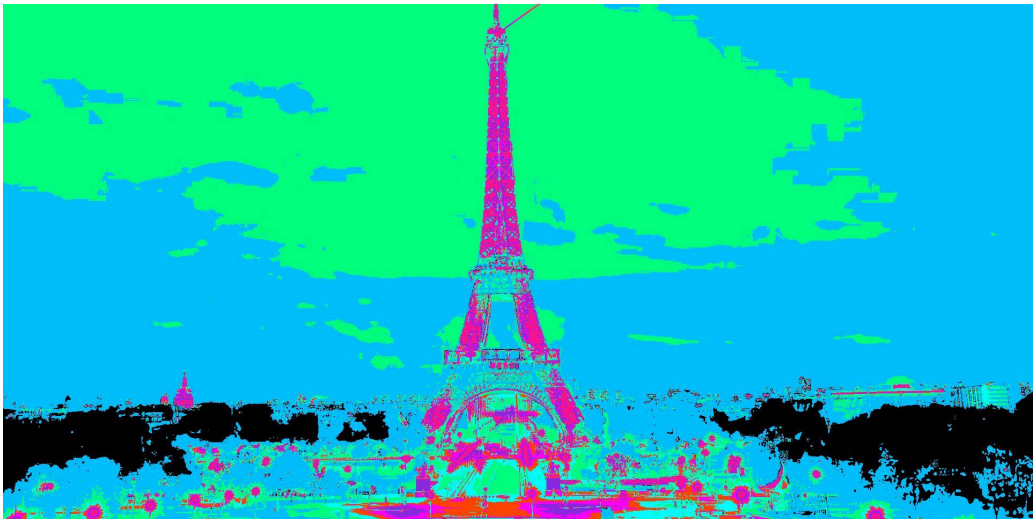

Figure 29. Cyberpunk Color Swapping on Eiffel Tower 4th Ordering

Figure 30. Beachside Breeze 16-color palette

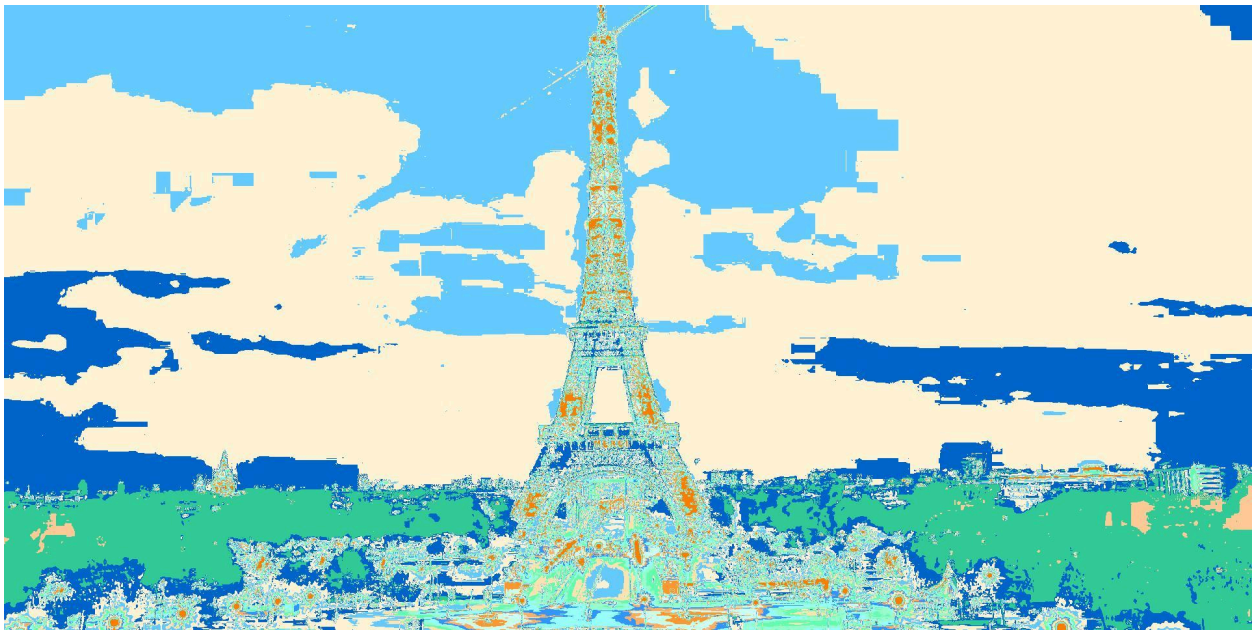

Figure 31. Beachside Breeze Color Swapping on Eiffel Tower 1st Ordering

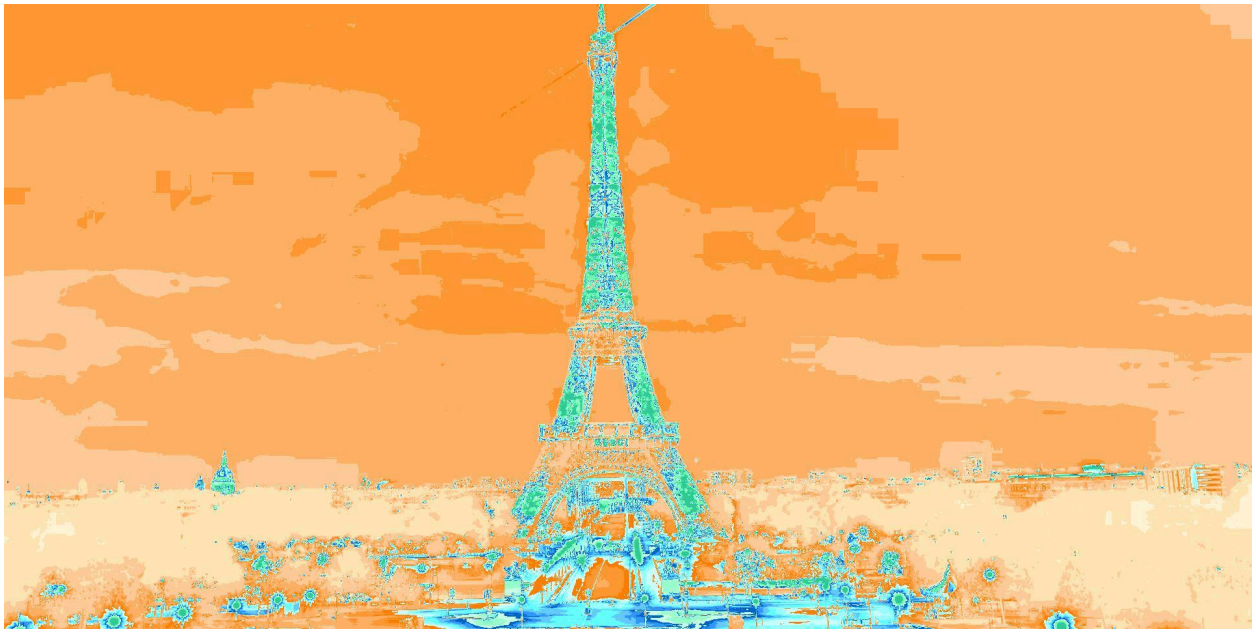

Figure 32. Beachside Breeze Color Swapping on Eiffel Tower 2nd Ordering

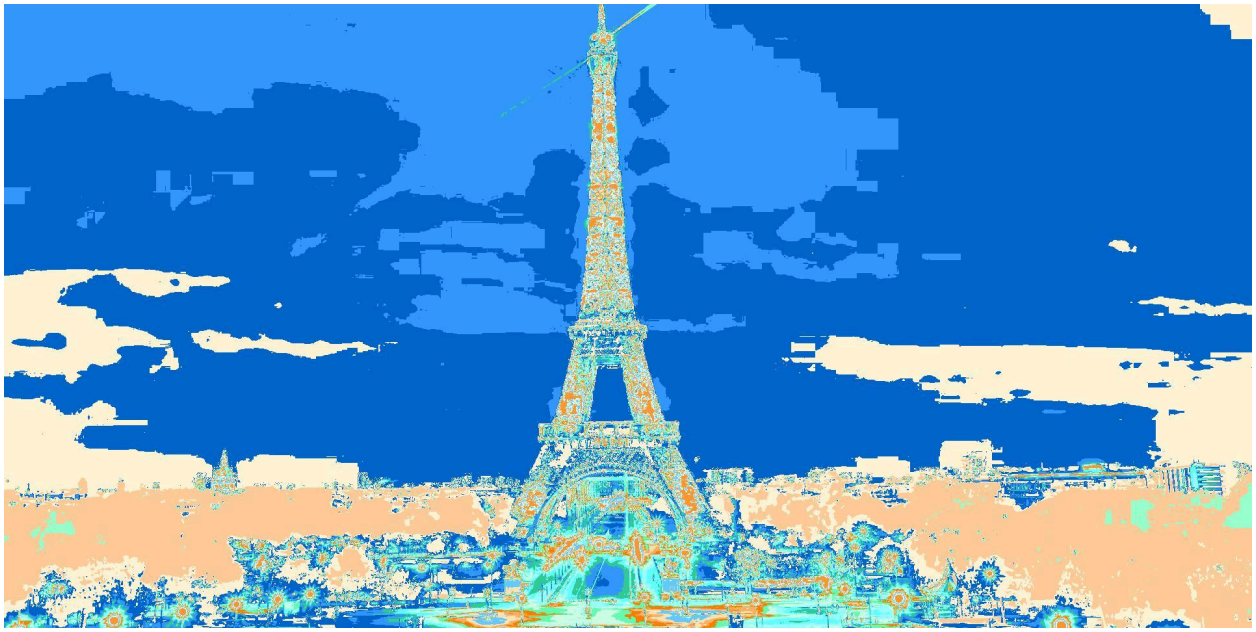

Figure 33. Beachside Breeze Color Swapping on Eiffel Tower 3rd Ordering

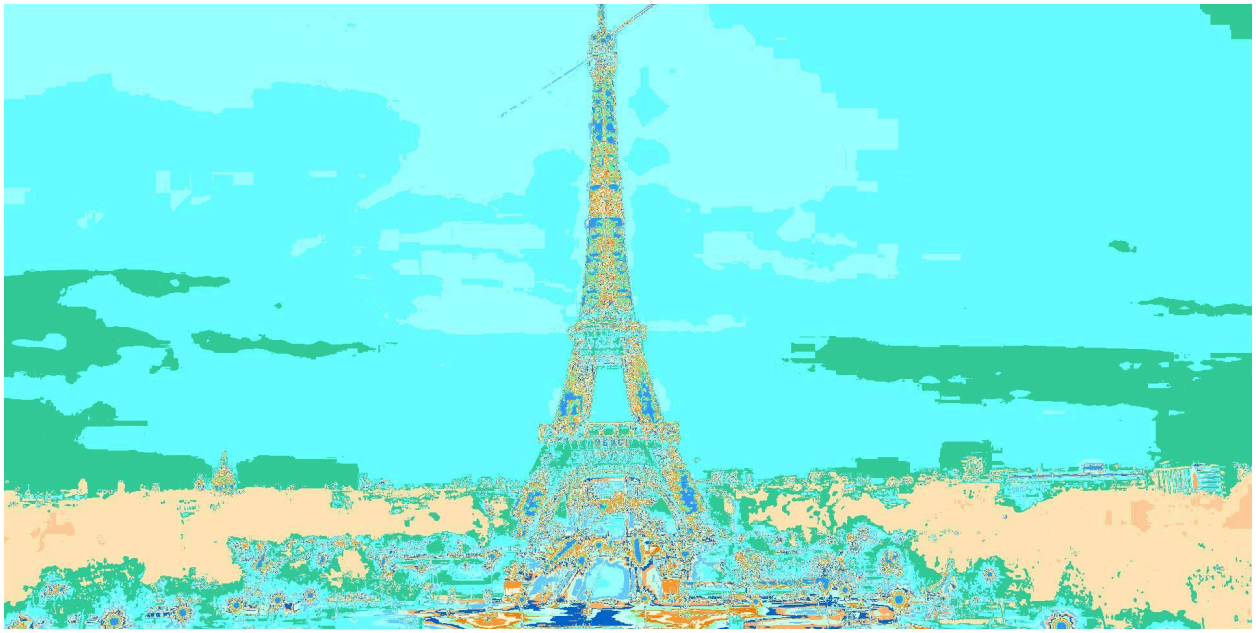

Figure 34. Beachside Breeze Color Swapping on Eiffel Tower 4th Ordering

Now that each stage of the algorithm's pipeline has been showcased along with the effects of each parameter I will now display the results of the complete pipeline (as well as intermediary steps) for each of the three original input images below. The parameter values used are a sharpness of 0.3x, a resize factor of 8x, a spread factor of 0.2, and a 16-color palette which I have shown first below.

Figure 35. Deep Space 16-color palette

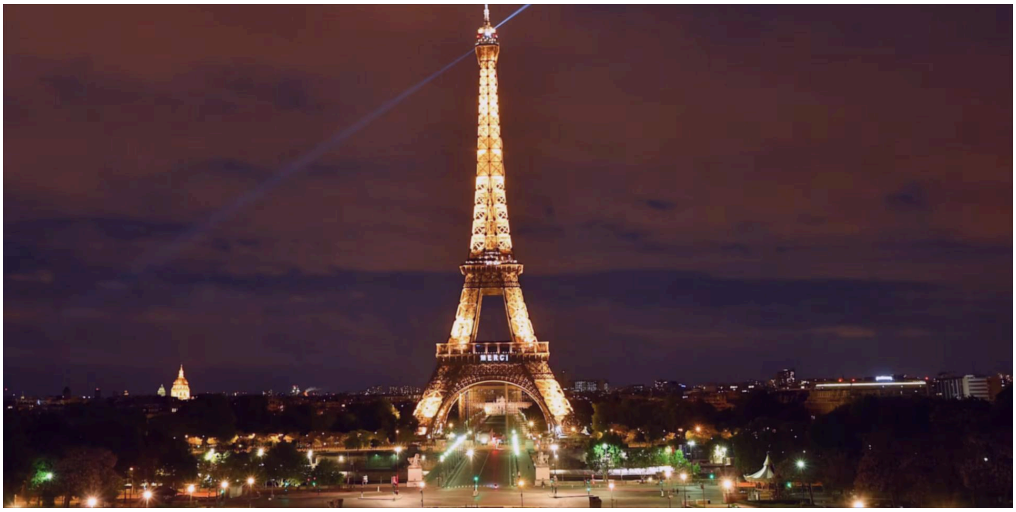

Figure 36. 0.2x Sharpness on Eiffel Tower Pipeline Sharpening

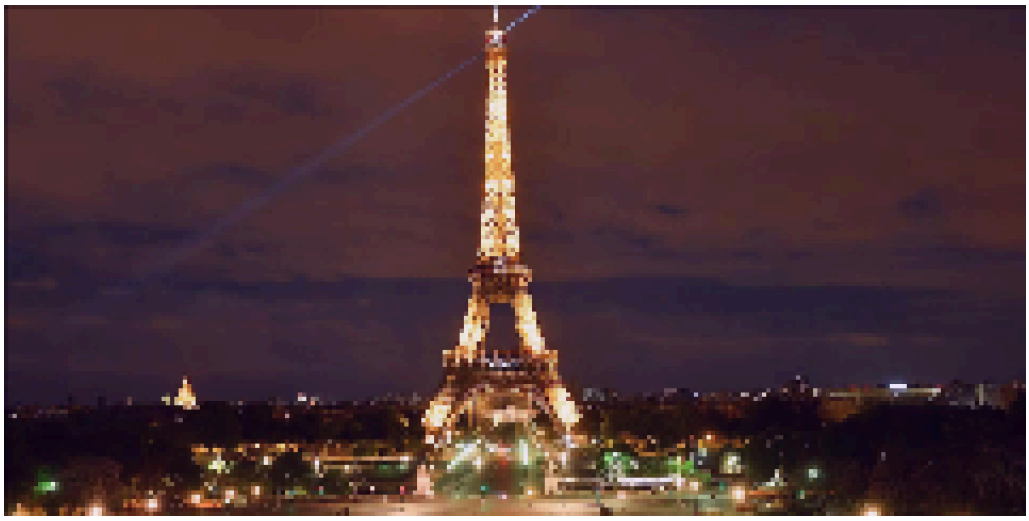

Figure 37. 8x Resize Factor on Eiffel Tower Pipeline Resizing

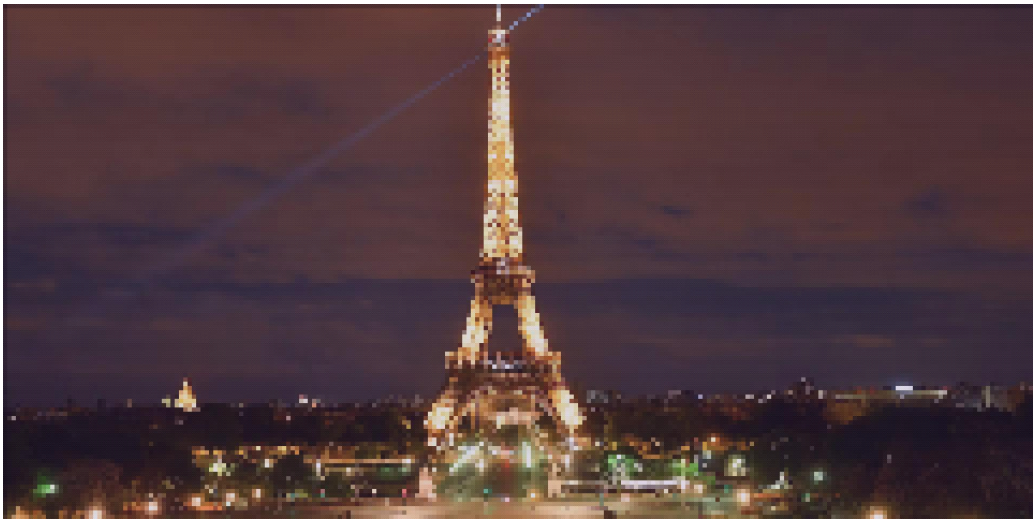

Figure 38. 0.2x Spread on Eiffel Tower Pipeline Dithering

Figure 39. Pipeline Grayscale on Eiffel Tower

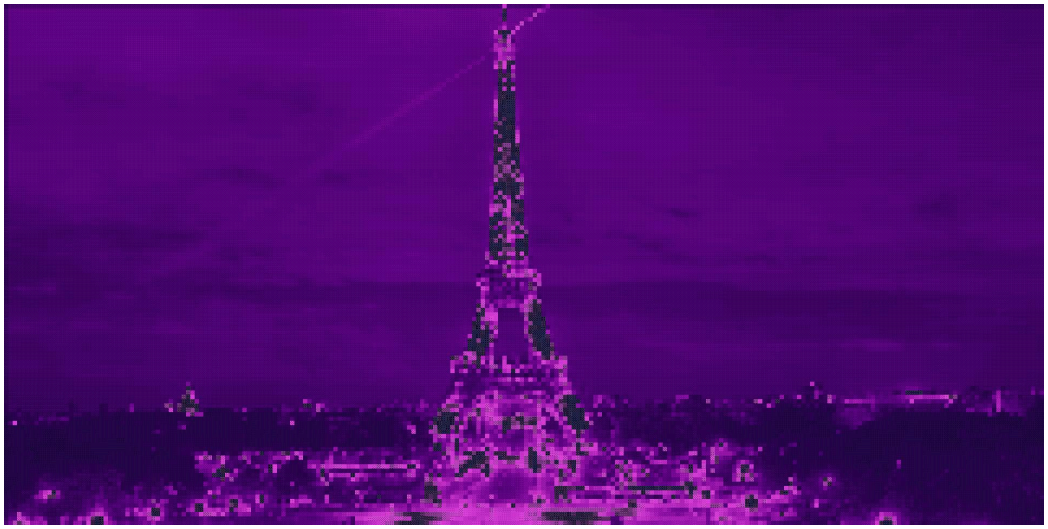

Figure 40. Deep-Space Pipeline Color Palette Swapping on Eiffel Tower 1st Order

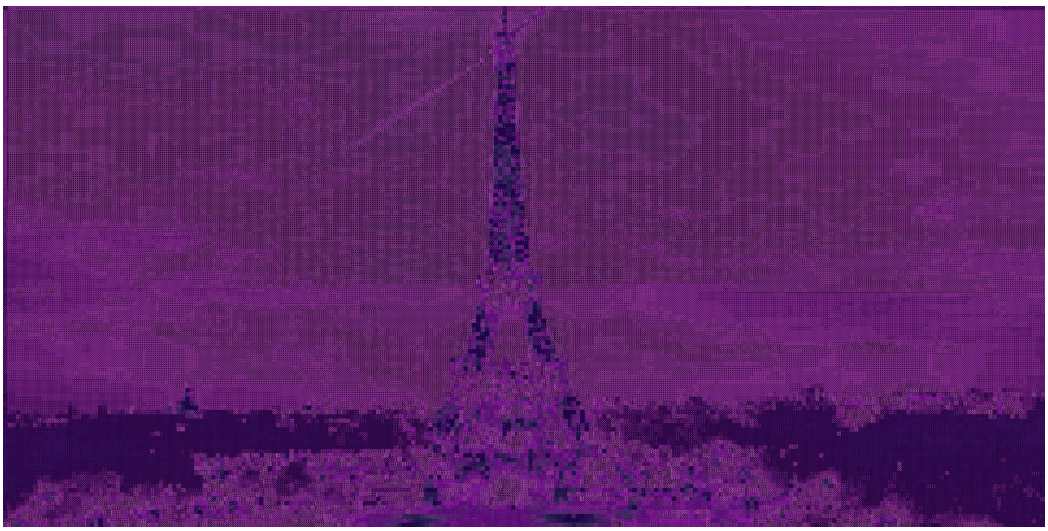

Figure 41. Deep-Space Pipeline Color Palette Swapping on Eiffel Tower 2nd Order

Figure 42. Deep-Space Pipeline Color Palette Swapping on Eiffel Tower 3rd Order

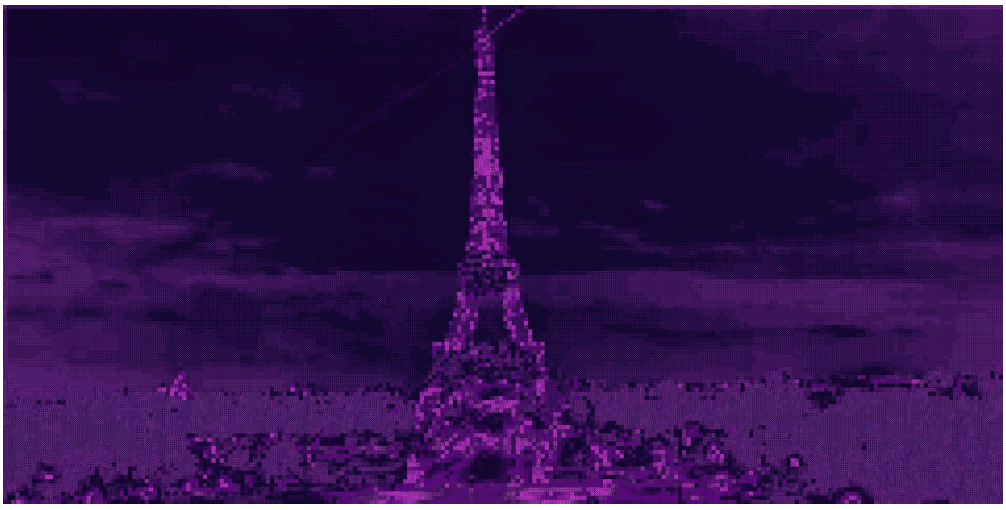

Figure 43. Deep-Space Pipeline Color Palette Swapping on Eiffel Tower 4th Order

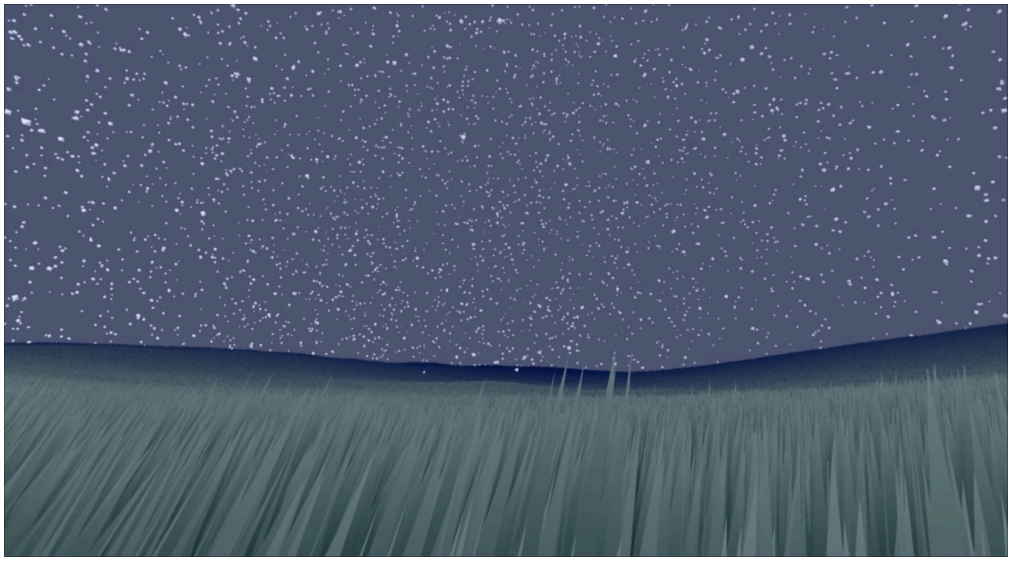

Figure 44. 0.2x Sharpness on Plain Pipeline Sharpening

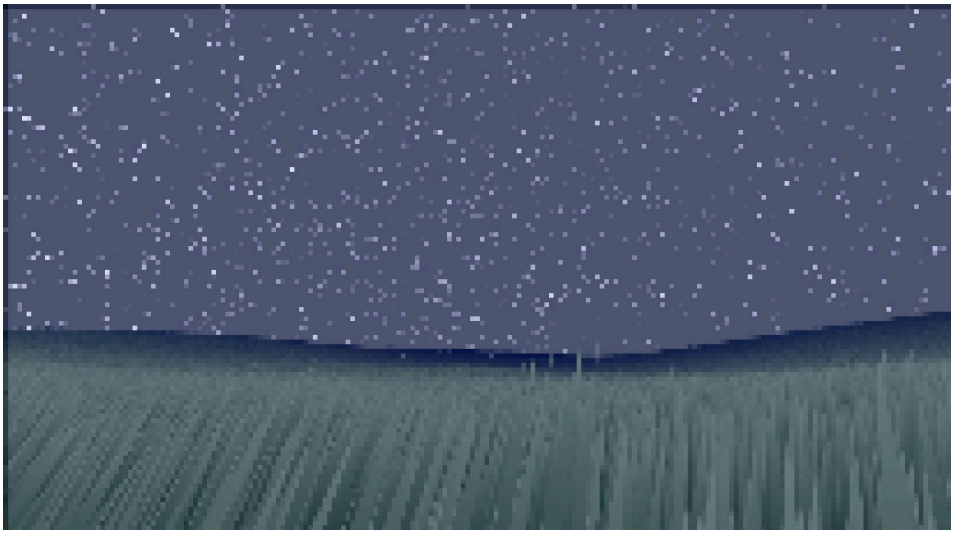

Figure 45. 8x Resize Factor on Plain Pipeline Resizing

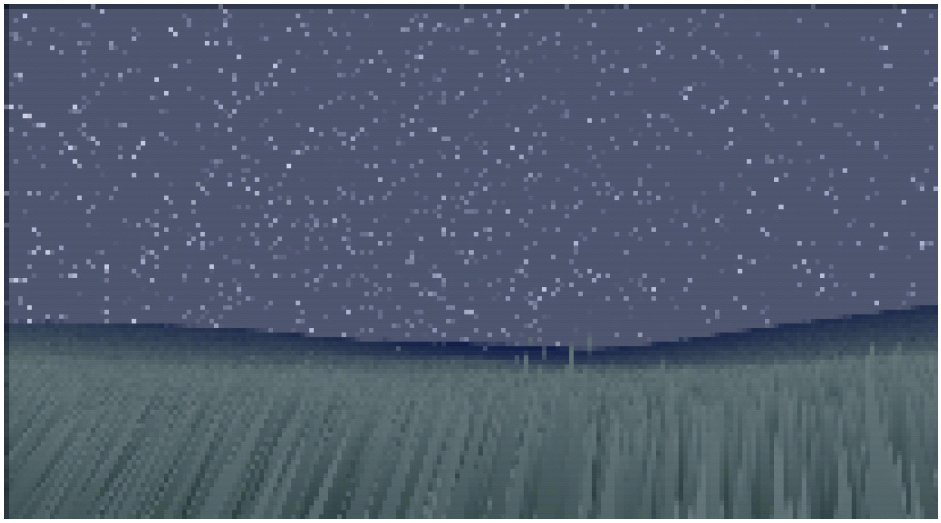

Figure 46. 0.2x Spread on Plain Pipeline Dithering

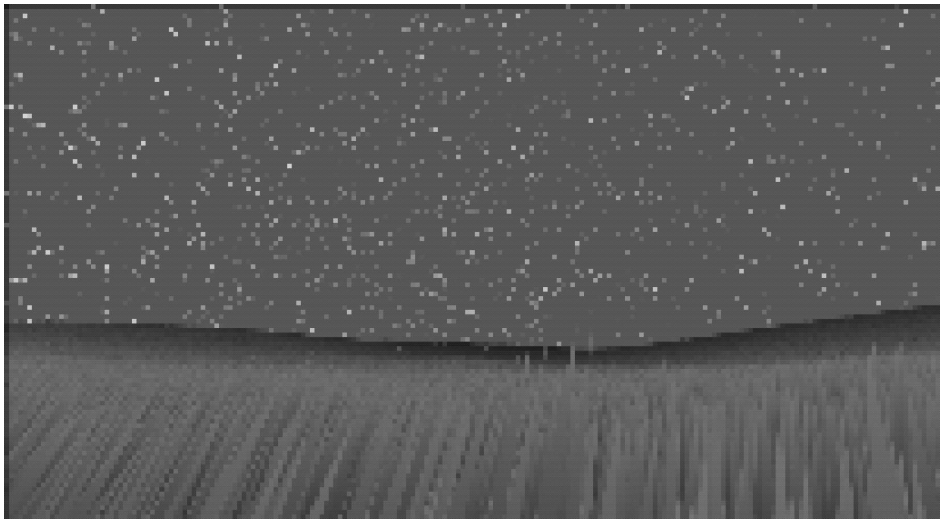

Figure 47. Pipeline Grayscale on Plain

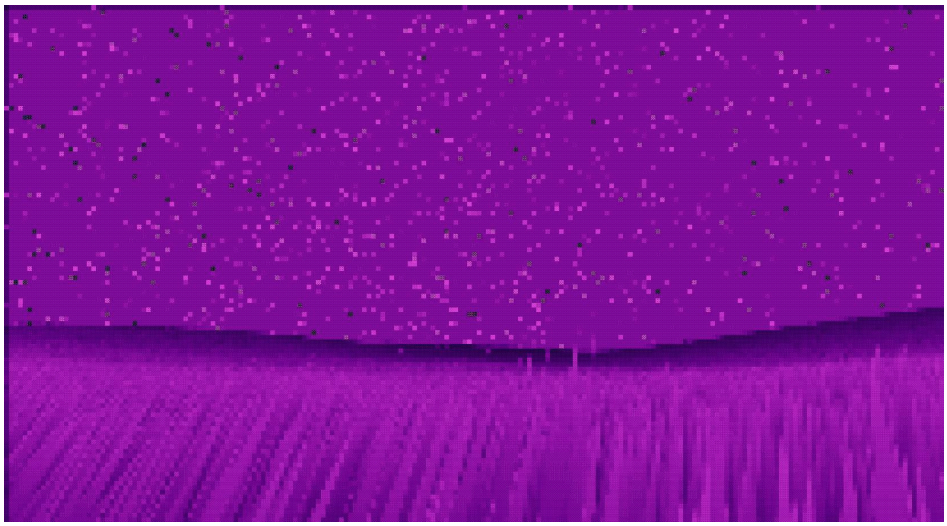

Figure 48. Deep-Space Pipeline Color Palette Swapping on Plain 1st Order

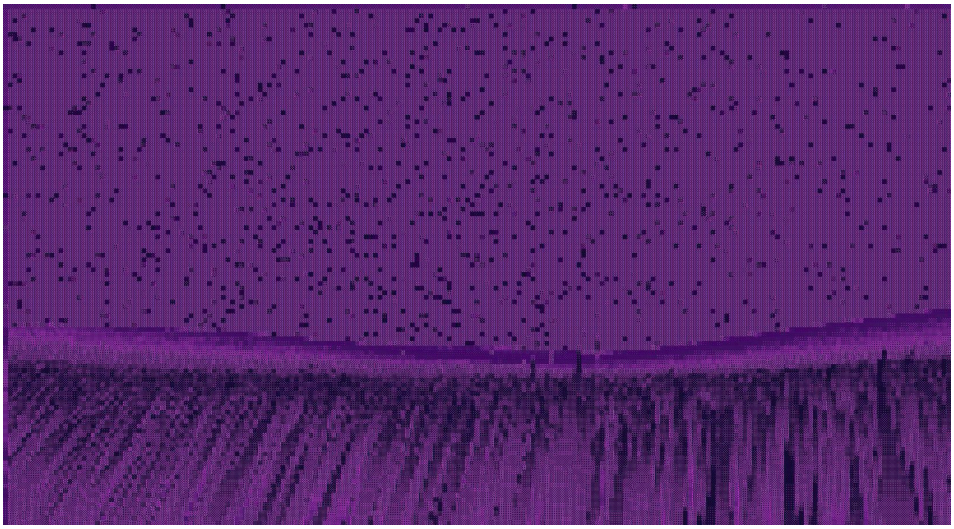

Figure 49. Deep-Space Pipeline Color Palette Swapping on Plain 2nd Order

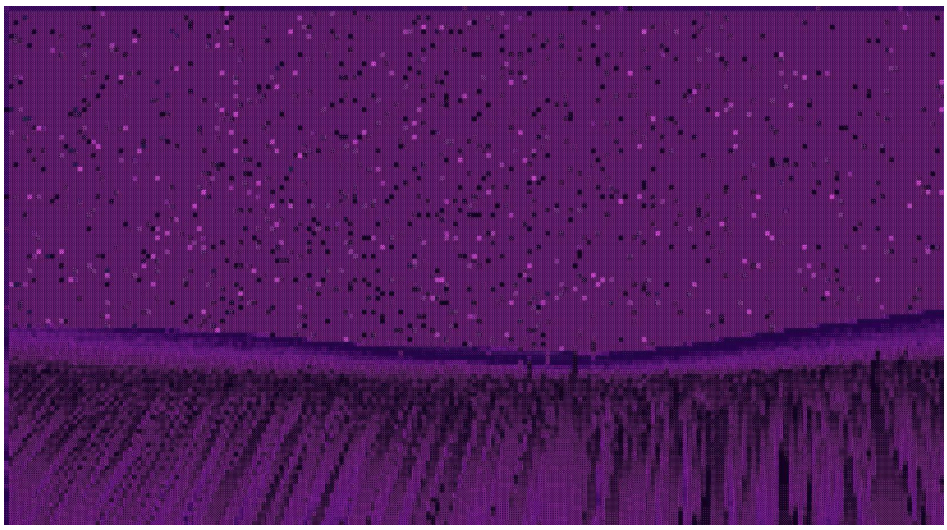

Figure 50. Deep-Space Pipeline Color Palette Swapping on Plain 3rd Order

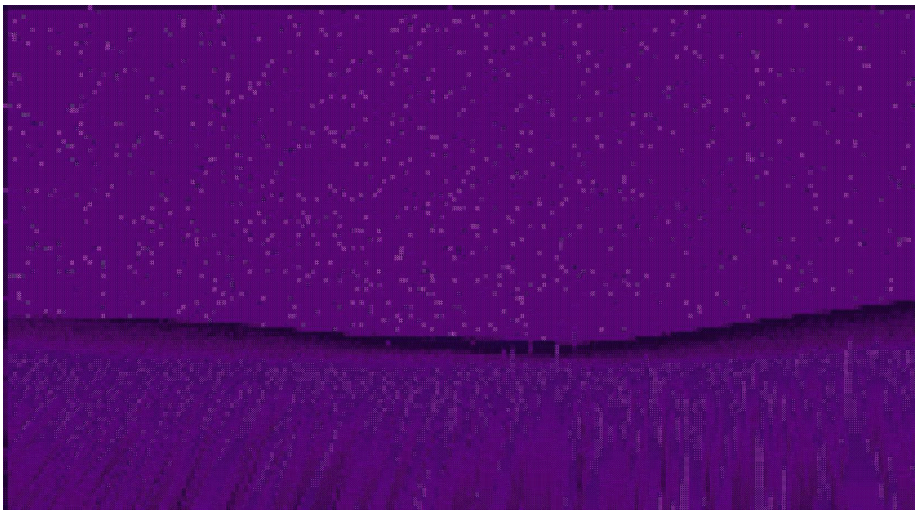

Figure 51. Deep-Space Pipeline Color Palette Swapping on Plain 4th Order

Figure 52. 0.2x Sharpness on Wave Pipeline Sharpening

Figure 53. 8x Resize Factor on Wave Pipeline Resizing

Figure 54. 0.2x Spread on WavePipeline Dithering

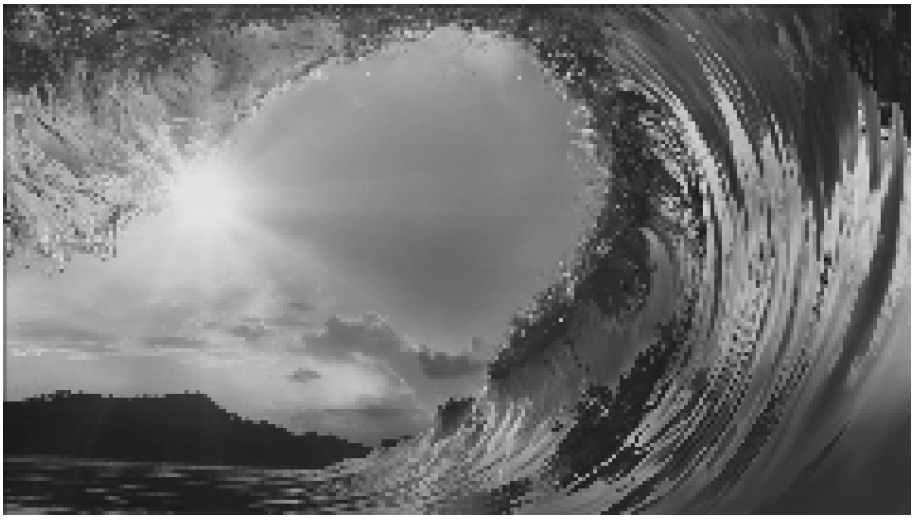

Figure 55. Pipeline Grayscale on Wave

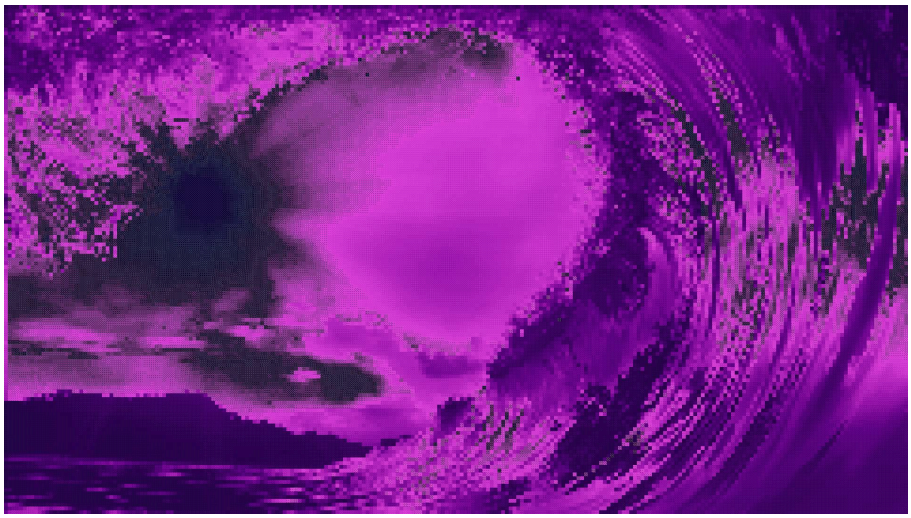

Figure 56. Deep-Space Pipeline Color Palette Swapping on Wave 1st Order

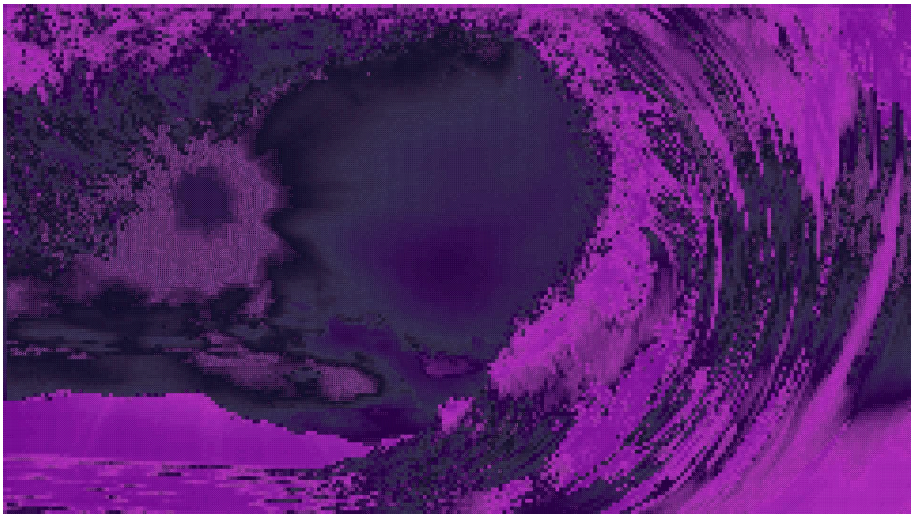

Figure 57. Deep-Space Pipeline Color Palette Swapping on Wave 2nd Order

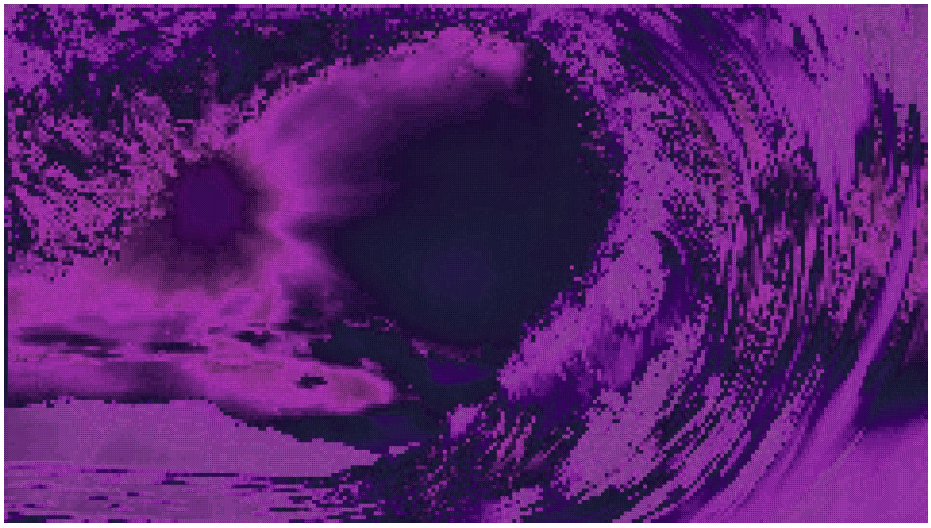

Figure 58. Deep-Space Pipeline Color Palette Swapping on Wave 3rd Order

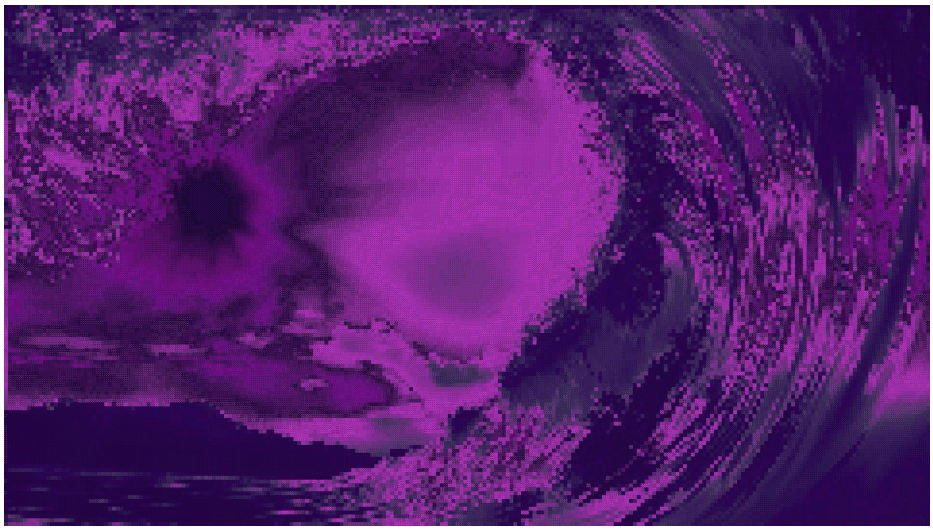

Figure 59. Deep-Space Pipeline Color Palette Swapping on Wave 4th Order

References

Gunnell, Garrett. “Color Quantization and Dithering.” *YouTube*, 2022, youtu.be/8wOUe32Pt-E?si=XNe3qiIK_sgqqSgI.

“Ordered Dithering.” *Wikipedia*, Wikimedia Foundation, 9 Feb. 2025, en.wikipedia.org/wiki/Ordered_dithering.

“Discrete Laplacian Operator.” *Wikipedia*, Wikimedia Foundation, 9 Feb. 2025, en.wikipedia.org/wiki/Discrete_Laplace_operator