Kernel-Based Image Processing

ANDREW PRATA, JR. | 12/04/23

Rensselaer

AGENDA

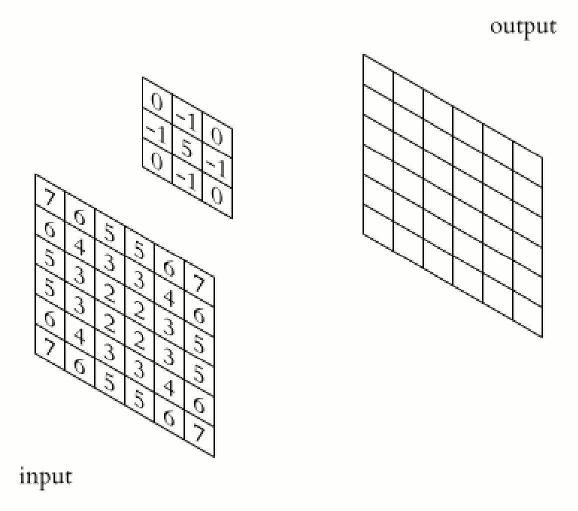
- Project Overview
- Kernel Convolution Explanation
- Functionalities Implemented
  - Mean Blur
  - Gaussian Blur
  - Inversion
  - Sobel Operator
  - Laplacian Operator
  - Histogram Equalization
  - Median Filtration
- Conclusions and Lessons Learned

PROJECT OVERVIEW

- Initial scope: implement traditional edge-detection algorithm
- Scope grew over time, standard edge detection is not exceedingly difficult to design
- Experimented with other kernel-based processing techniques (blurs, filters, etc.)
- Designs implemented:
  - Gaussian blur, mean blur, Sobel operator, Laplacian operator, inversion, histogram equalization, median filtration

KERNEL CONVOLUTION

- Basis of many image processing techniques: convolution
- Kernel: “grid” that is *convolved* with the input image (also just a “grid”)
  - Contains filter information
- Basic explanation:
  - Kernel “overlaid” on input image, centered on one pixel
  - Each value in the kernel is multiplied with the value it overlaps from the input, these are then summed
  - This sum is the output pixel; repeat for all input pixels



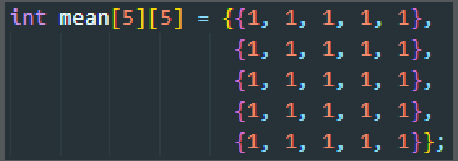
0	-1	0
-1	5	-1
0	-1	0

7	6	5	5	6	7
6	4	3	3	4	6
5	3	2	2	3	5
5	3	2	2	3	5
6	4	3	3	4	6
7	6	5	5	6	7

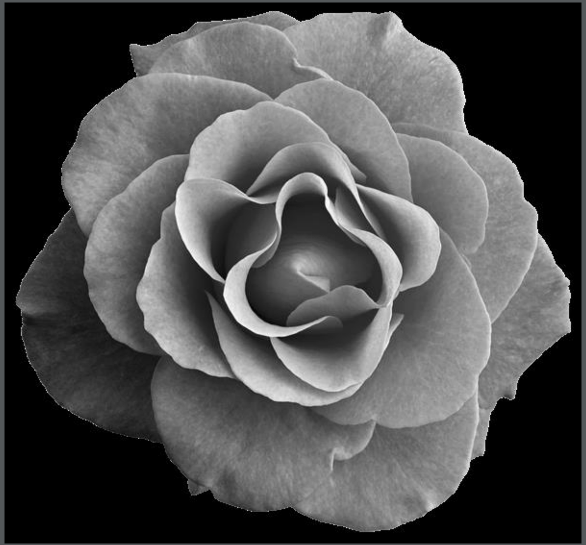
Convolution Visualized
Source: Wikipedia

MEAN BLUR

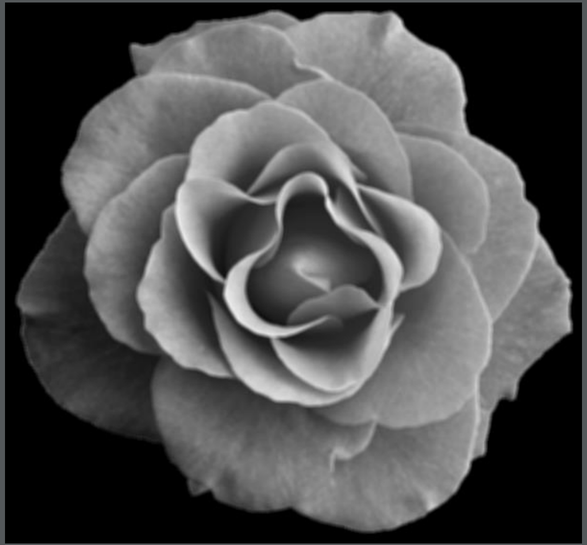
- Kernel of all “1”, equal weighting of all pixels in kernel region
- Causes a true averaging of all pixel values



```
int mean[5][5] = {{1, 1, 1, 1, 1},
                  {1, 1, 1, 1, 1},
                  {1, 1, 1, 1, 1},
                  {1, 1, 1, 1, 1},
                  {1, 1, 1, 1, 1}};
```



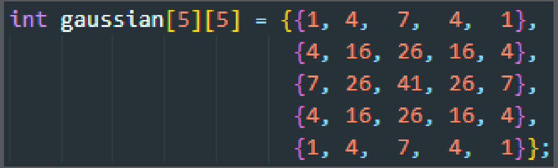
Raw Grayscale Image



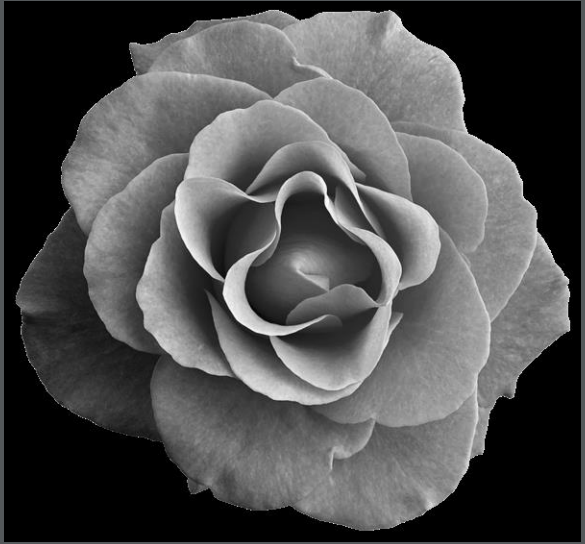
5x5 Mean Blur

GAUSSIAN BLUR

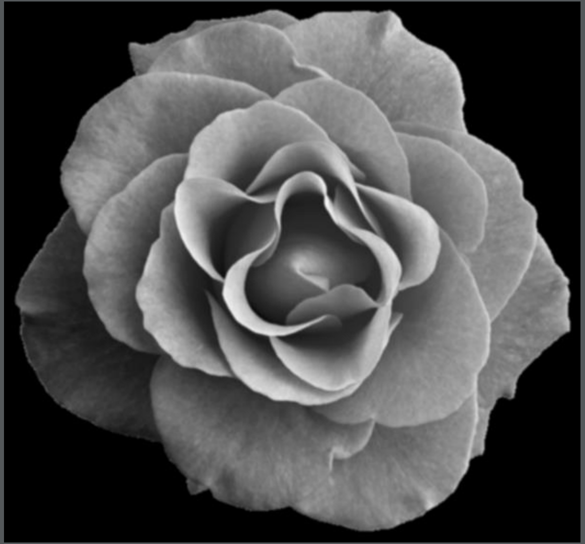
- Kernel constructed from discrete Gaussian distribution
- Weights origin pixel heavier than neighbors



```
int gaussian[5][5] = {{1, 4,  7,  4,  1},
                      {4, 16, 26, 16, 4},
                      {7, 26, 41, 26, 7},
                      {4, 16, 26, 16, 4},
                      {1, 4,  7,  4,  1}};
```



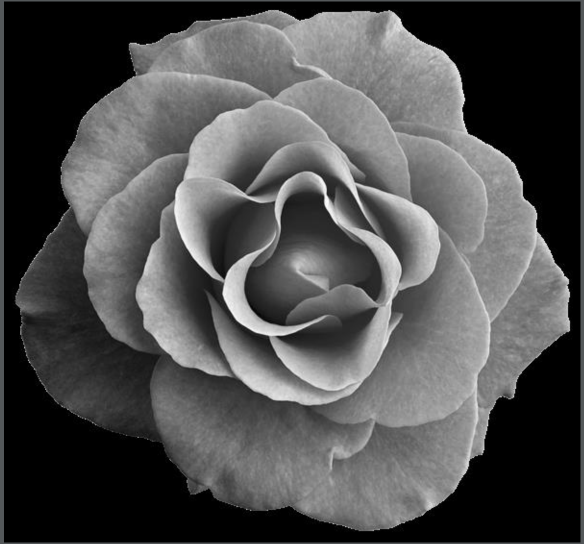
Raw Grayscale Image



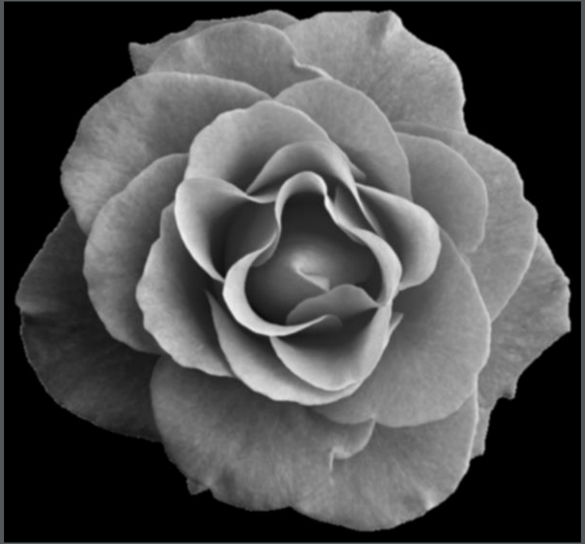
5x5 Gaussian Blur

BLUR COMPARISON

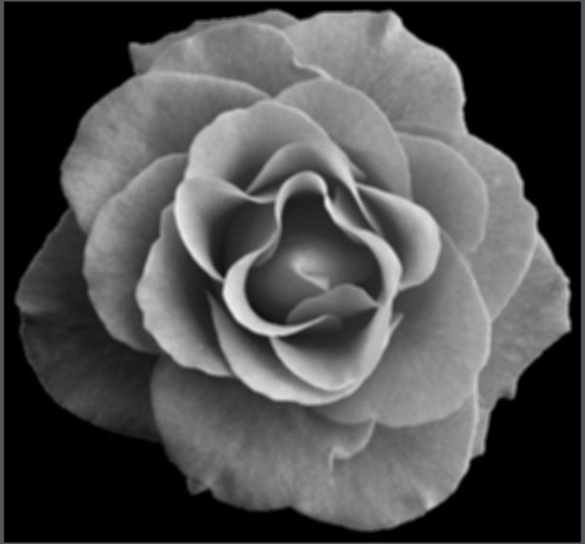
- Gaussian blur preserves edges, but still softens the image
- Mean blur is generally *not* edge preserving, and simply blurs the entire image



Raw Grayscale Image



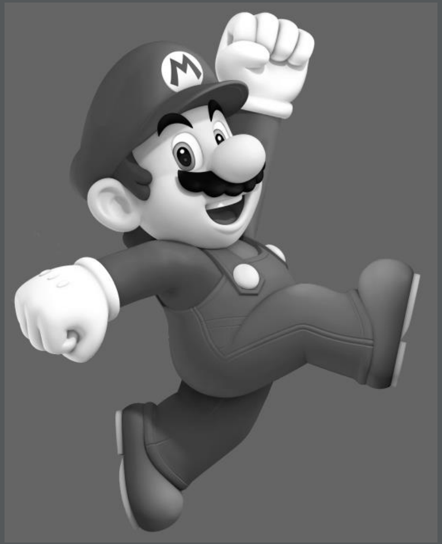
5x5 Gaussian Blur



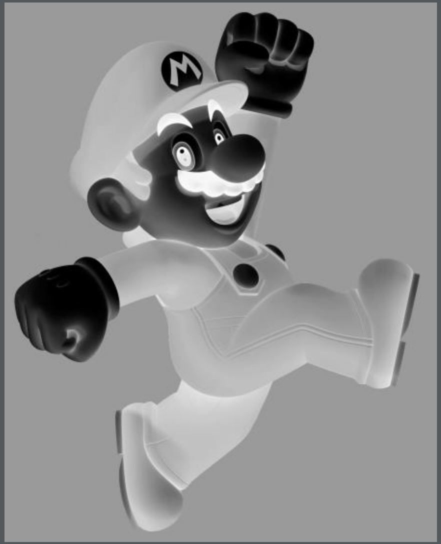
5x5 Mean Blur

INVERSION

- Does not require kernel-based processing, simple subtraction: *newVal = 255 - val*
- Used to make other functions easier to see



Raw Grayscale Image

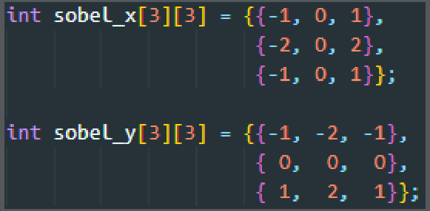


Intensity Inversion

SOBEL OPERATOR (EDGE DETECTION)

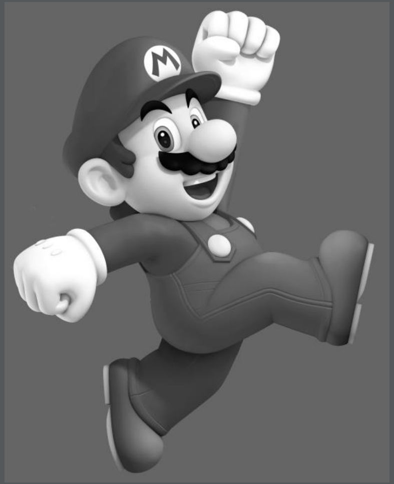
- Emphasizes difference between {top, bottom} of origin (y-direction), and {left, right} of origin (x-direction)

- Compute magnitude of both x and y convolution sums for output

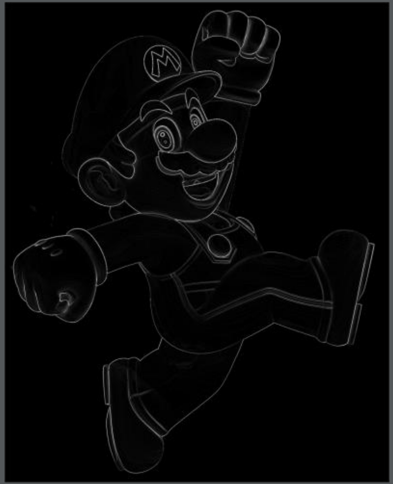


```
int sobel_x[3][3] = {{-1, 0, 1},
                     {-2, 0, 2},
                     {-1, 0, 1}};

int sobel_y[3][3] = {{-1, -2, -1},
                     { 0,  0,  0},
                     { 1,  2,  1}};
```



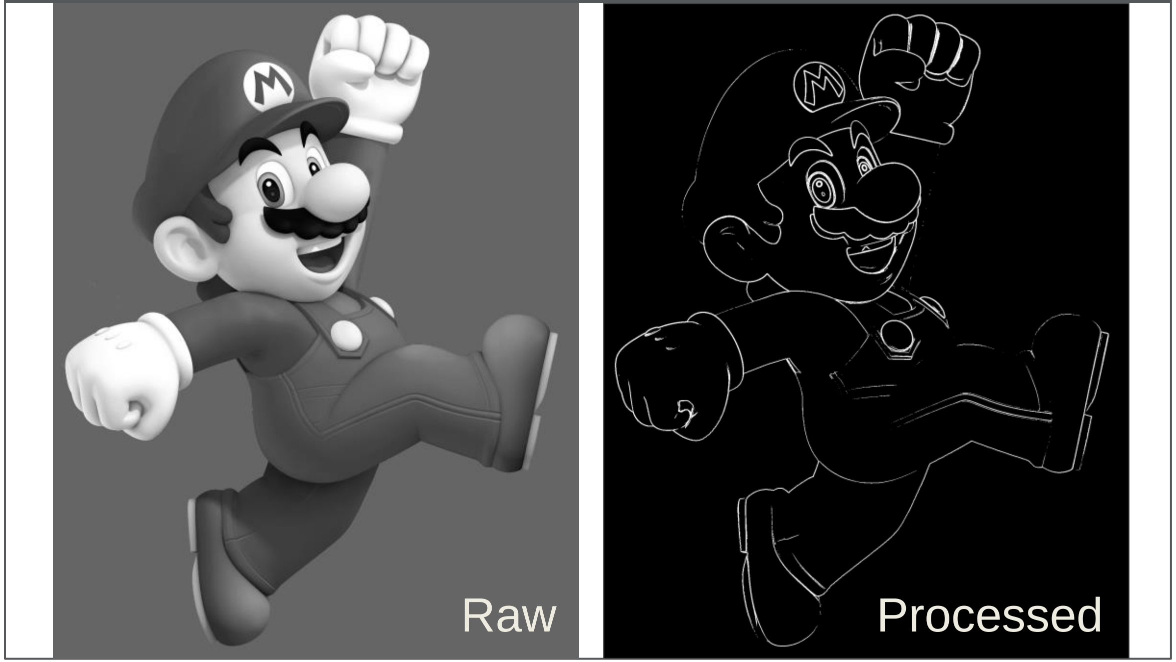
Raw Grayscale Image



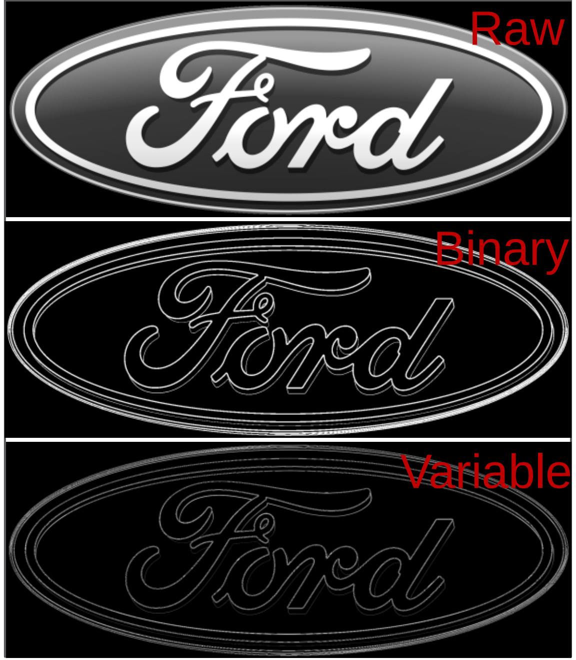
Sobel Operator Result (No Tuning)

SOBEL OPERATOR (TUNING ENHANCEMENT)

Threshold Intensity 56, Binary Output

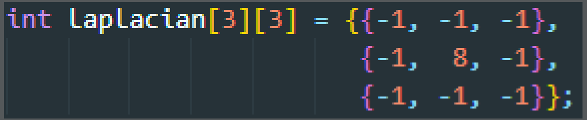


Threshold Intensity 48



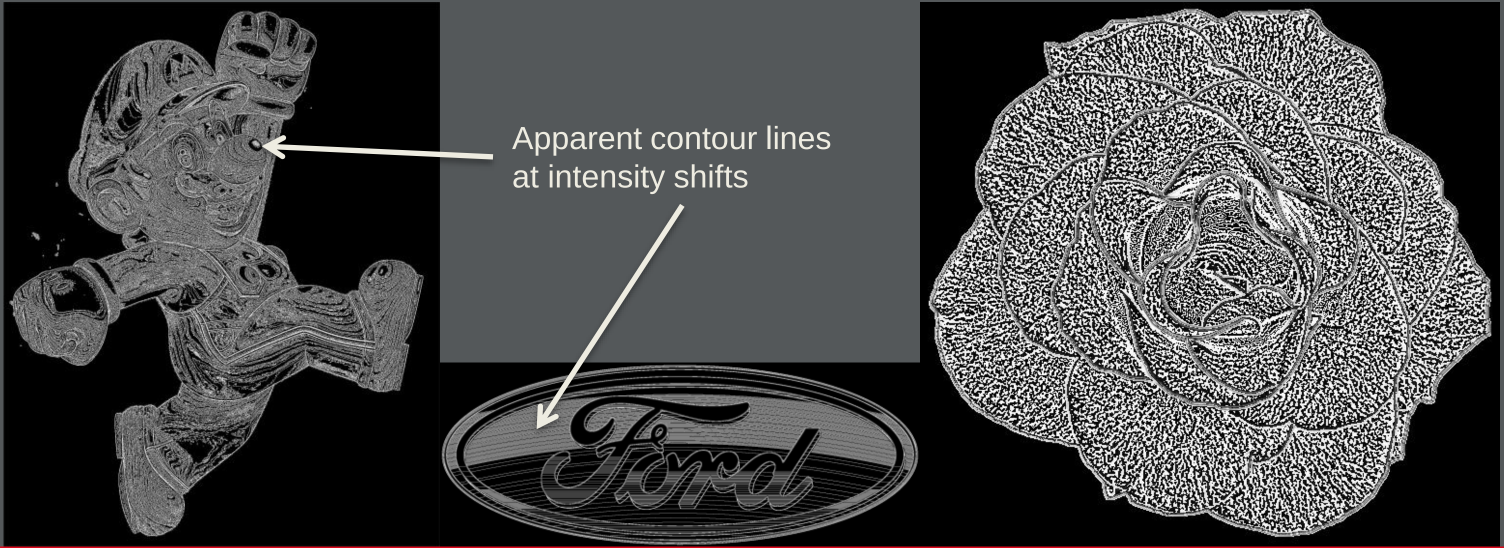
LAPLACIAN OPERATOR

- Detects changes in intensity from origin to neighbors (effectively edge detection)



```
int laplacian[3][3] = {{-1, -1, -1},
                       {-1,  8, -1},
                       {-1, -1, -1}};
```

- In my testing, seems to detect “contours” well - see samples:



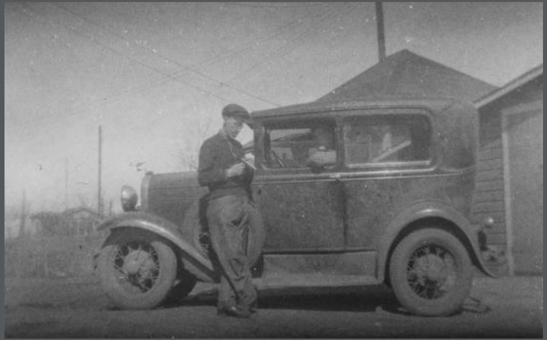
Apparent contour lines at intensity shifts

HISTOGRAM EQUALIZATION (SATURATION)

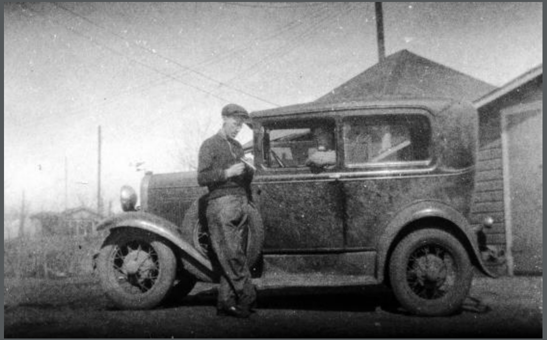
- Form a histogram of intensity values (count vs. intensity plot)

- Create an *accumulated histogram*, which is just the summation of the histogram at each point (Cumulative Distribution Function, or CDF)

- Normalize the accumulated histogram over the range of intensities (0-255)
  - Use to re-map intensities of original image



Raw Grayscale Image

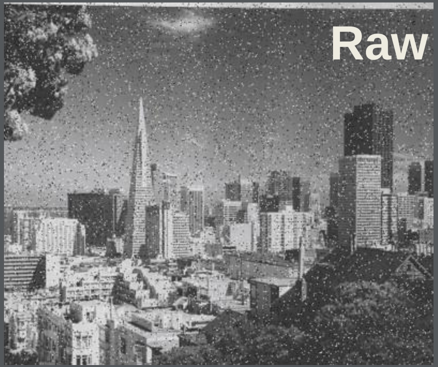


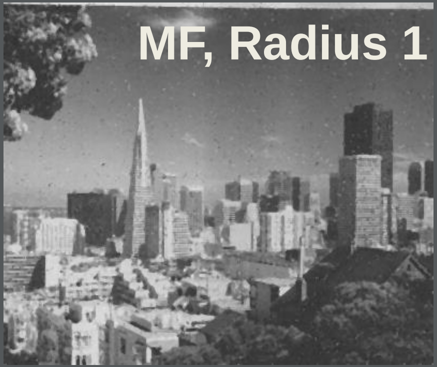
Histogram Equalization

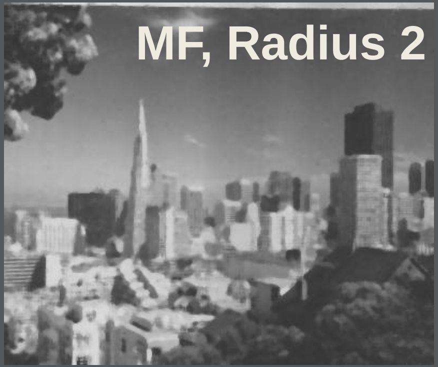
MEDIAN FILTRATION

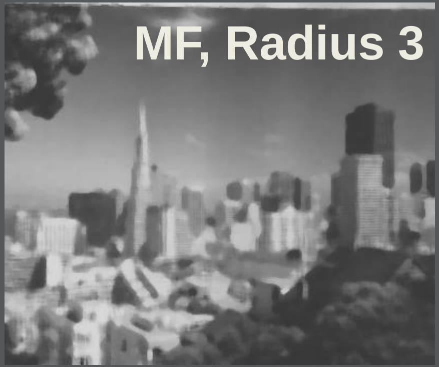
- The Median Filter establishes a neighborhood of size **[(2 * Radius + 1)^2]** centered on the origin pixel

- Determine and sort the intensities of this pixel neighborhood, and output is the median value (I wonder why it is called “Median Filter?”)
    - Destroys salt-and-pepper noise typically observed in old photographs, trade-off with sharpness



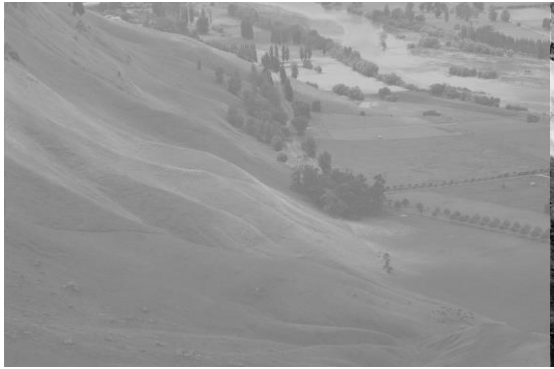




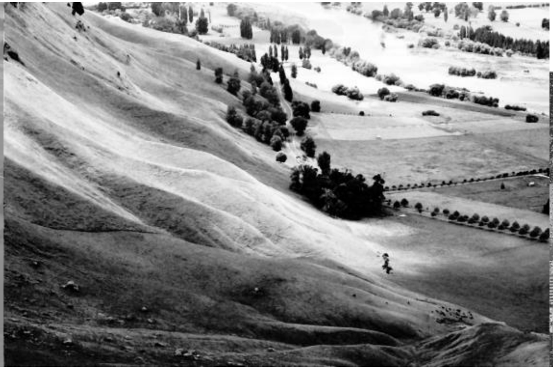


A POWERFUL COMBINATION…

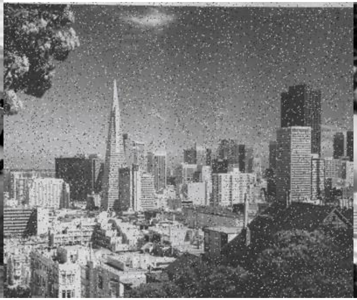
- Combining Median Filtration and Histogram Equalization can be powerful for restoration of corrupted images
  - Requires tuning: Which to apply first? What size median window? Can be successful: see examples.



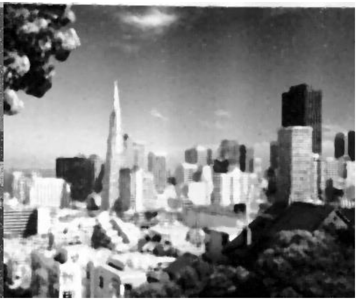
Landscape, Before



Landscape, After



City, Before



City, After

CONCLUSIONS

- Before this project, I had effectively no idea how image processing was done
  - I got to learn foundational elements based in kernel convolution
    - Blurs
    - Filters
    - Edge Detection
  - I also used a few algorithmic processing methods, from basic inversion to histogram equalization
- I was impressed how little code is required to completely transform an image for the better, or extract useful features like edges
- Tuning outputs and experimenting with different kernels was a rewarding and fun experience

Thank you for your attention!

Questions?

All source code, images, and references are available at
https://github.com/apratajr/AdvancedComputerSystems/tree/main/Final

RENSSELAER POLYTECHNIC INSTITUTE
KNOWLEDGE AND THOROUGHNESS
1824

Rensselaer

why not change the world?®