

Convolution Project

Brandon Rodriguez



Input Files

Input Files - File Type

- Images in “Portable Bitmap Format”
 - One of the most basic image types.
 - No compression.
 - Header has 3 parts:
 - File type, such as P6 (rgb) and P5 (greyscale).
 - Image width (in pixels)
 - Image height (in pixels)
 - Body is raw rgb pixel data.



Input Files - Image Content

- 38 Images total.
- Images taken myself, with my phone.
 - Various assorted images taken over the past years.
- Images are various content, split into 6 categories.
 - Meant to test effectiveness of code with wildly varying input.
- Tried to have approximately 5 to 8 images per category.



Input File Categories - Nature



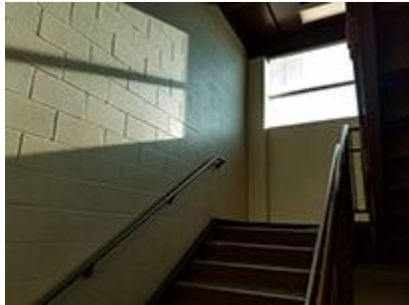
Input File Categories - Nature Fuzzy



Input File Categories - Objects



Input File Categories - Hallways



Input File Categories - Blurry



Input File Categories - Other



Input Images - Format

- Rescaled to varying sizes
 - 13 x 10 pixels
 - 400 x 300 pixels
 - 2000 x 1500 pixels
 - 4000 x 3000 pixels (original file format)
- Two sets of every image, per size group
 - Original RGB (full color) format
 - Desaturated Greyscale (black-and-white) format

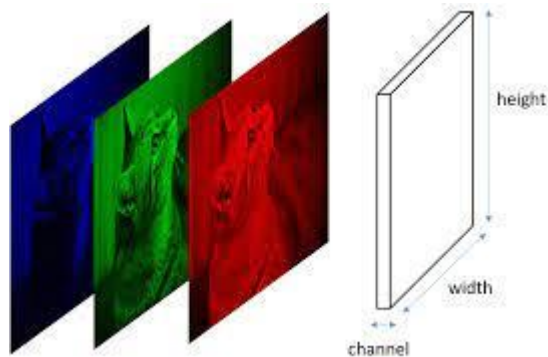


Input Images - Format

Reminder for image file format.

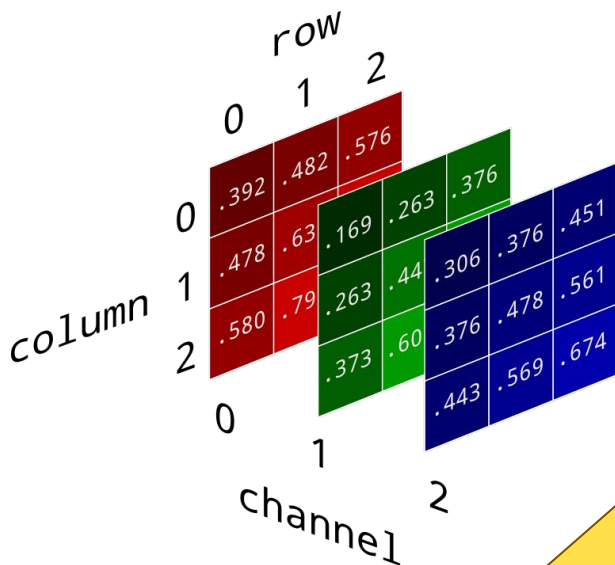
The most basic implementation of color image will have:

- 3 different “Channels”.
- Each channel represents a “base” color (red/green/blue).



Input Images - Format

- Each pixel has a numerical value for each channel.
- Together, the channels create the final color we physically see.





Code Setup

Code Setup

- C++ with CUDA for GPU computations.
- Only used standard C++ libraries.
- Images were read into an 1D array of structs.
 - Each struct represented an individual pixel.
 - Struct contained a float for each of:
 - Red pixels
 - Green Pixels
 - Blue Pixels



Code Setup

- Convolution masks were also represented with 1D struct arrays.
- Mask channel values (red/green/blue) generally matched, for a given pixel.
 - Aka, mask [red == green == blue], for each pixel.





Initial Experiments

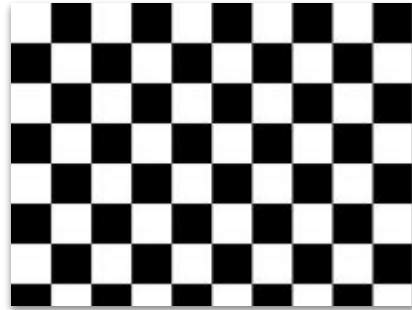
01 - Pixel Manipulation Test

- Most basic possible manipulation.
- Set alternating pixels to black/white, regardless of original image content.
- Result was a boring grey color for larger images.



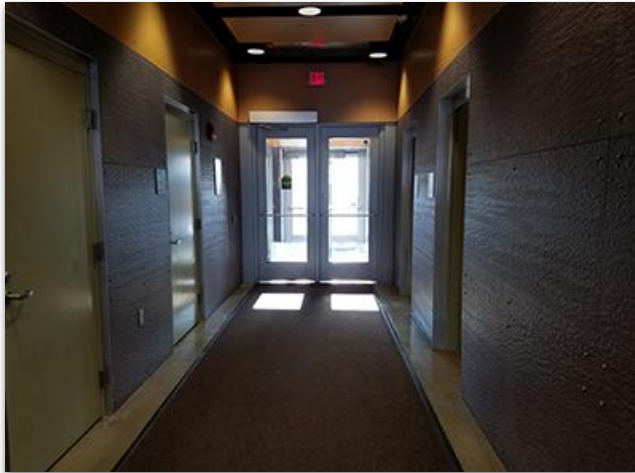
02 - Checkerboard

- Basic manipulation that takes rows/columns into account.
- Creates groups of pixels into squares and sets each group to alternating black/white.



03 - Adjusting Individual Channels

- Test to manipulate each color channel (red/green/blue) separately.





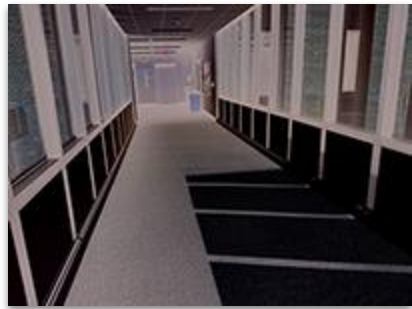
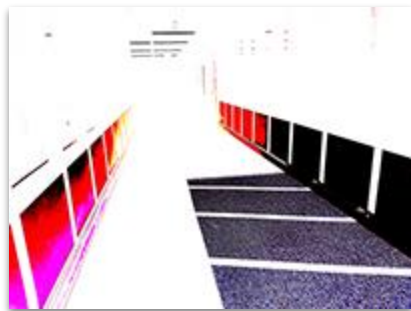
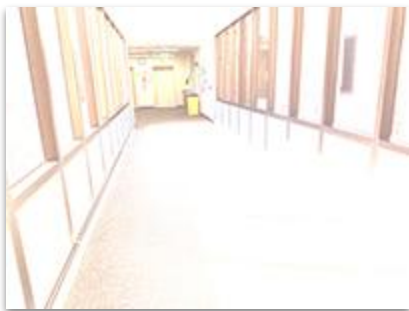
Convolution Implementation

Gaussian Blur

Importance of Mask

- Initially, I thought the mask was simply a “weight”.
- Assumed most convolution logic occurred after weights.
- Attempted to “guess” the initial mask values and got...interesting results.





Gaussian Mask

- Mask Center should be largest value.
- Values gradually get smaller towards edges.
- All mask values should add up to a total of 1.

$$\frac{1}{273}$$

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

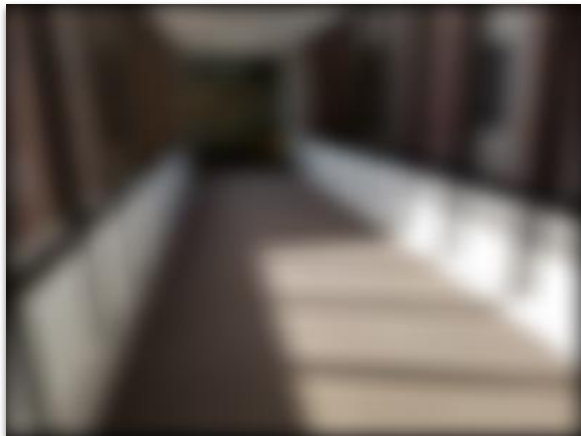


Gaussian Mask Size

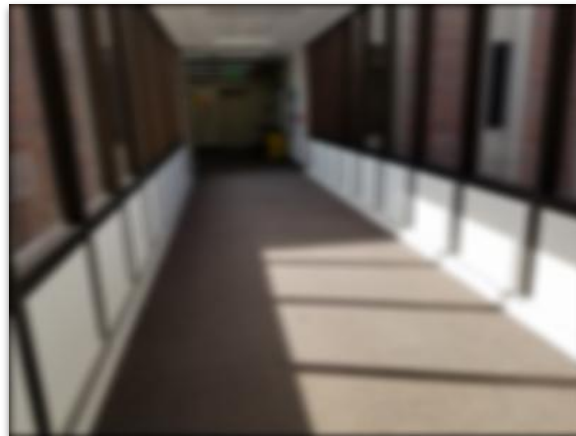
- Size of image didn't seem to affect anything.
 - Excluding computational cost.
- But size of mask (proportional to image) matters.
 - Larger mask means:
 - More blur.
 - Higher computational cost.



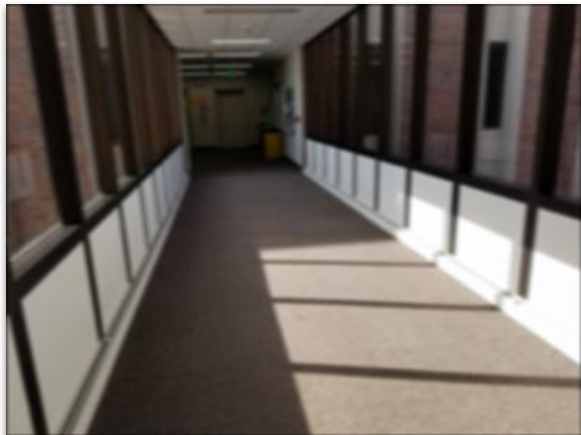
Mask Size: 10%



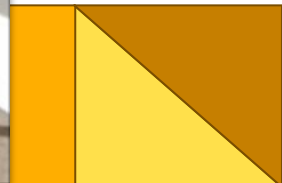
Mask Size: 4%



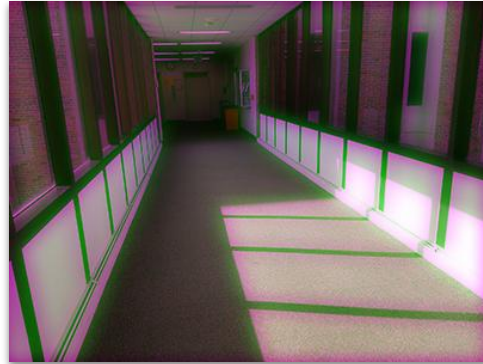
Mask Size: 2%



Mask Size: 1%



Effect of Channels in Gaussian



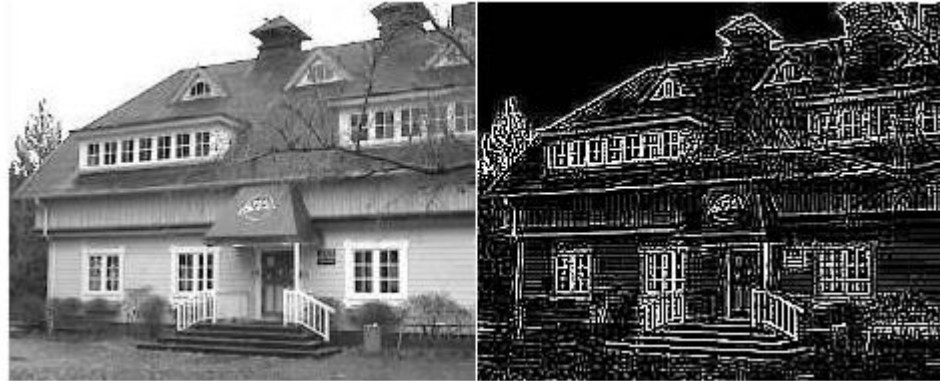


Edge Detection

Importance of Masks (Revisited)

- Convolution Mask is very important.
- Simply changing the mask changes what the convolution can do.

-1	-1	-1
-1	8	-1
-1	-1	-1



Laplacian of Gaussian Mask

- Possible to mathematically include complex concepts into the mask.
- The “**Laplacian Operator**” examines the second derivative values of data.
 - Due to dealing with derivatives, can be sensitive to noise.
- “**Laplacian of Gaussian**” fixes this by combining Gaussian Blur into the mask.



Mask Properties

- Smaller masks worked best:
 - Ideal seemed to be 7x7 pixels or 9x9 pixels.
 - 9x9 better for “man made” objects, or with clear object separation.
 - 7x7 better for “natural”/“plant-like” objects, or with very textured images.
- Image size seemed to directly affect how accurate line detection was:
 - Small images had rough/approximate detection.
 - Large image had extremely accurate detection.



Edge Detection - Image Size

Image Size: 4000x3000



Image Size: 2000x1500

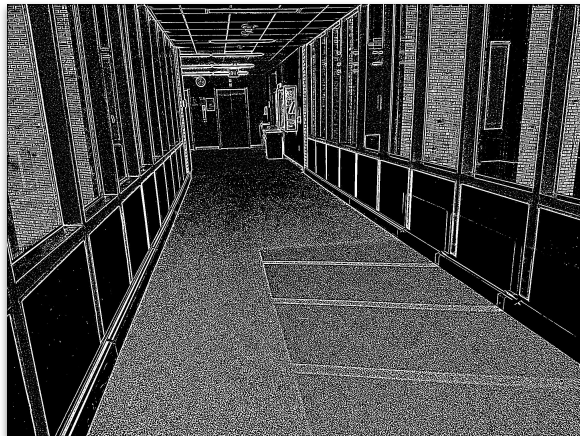


Image Size: 400x300



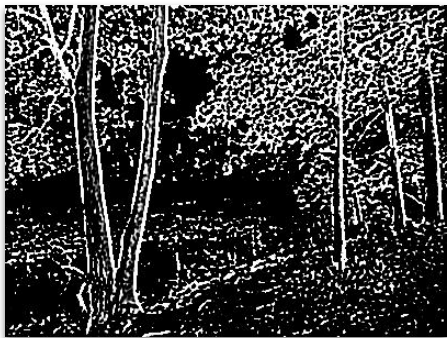
Due to lack of detail, smaller may be better for most “computer vision” applications.



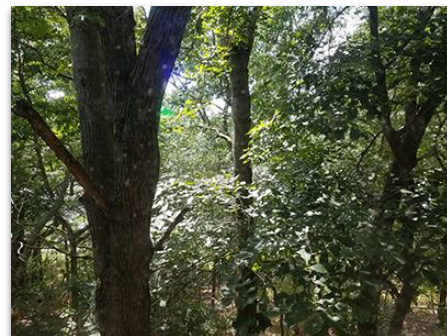
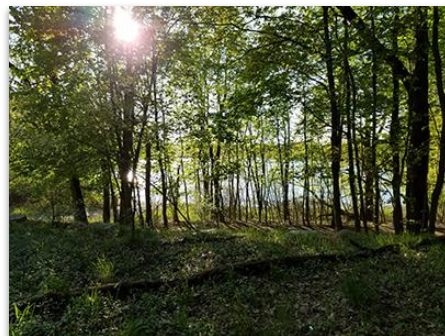
Edge Detection - Nature (Hi Res)



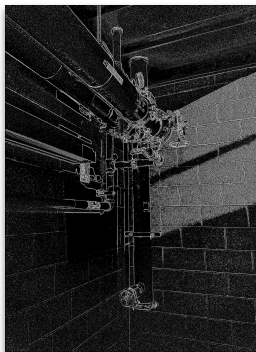
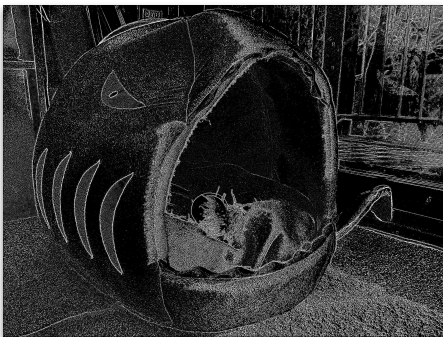
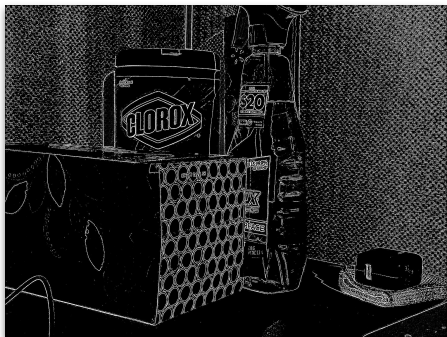
Edge Detection - Nature (Low Res)



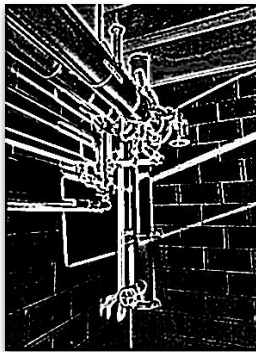
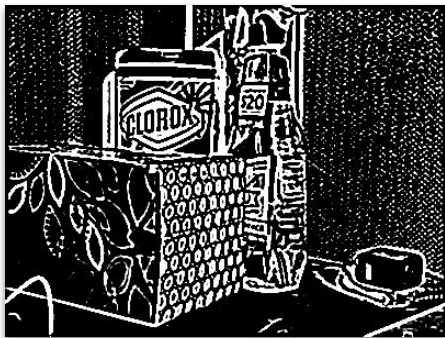
Edge Detection - Nature Fuzzy



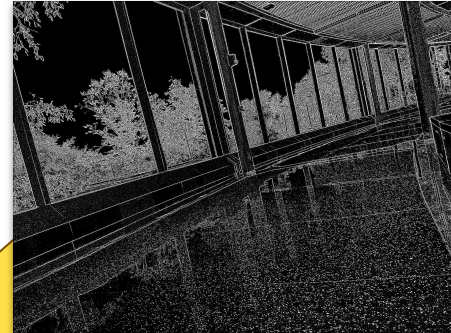
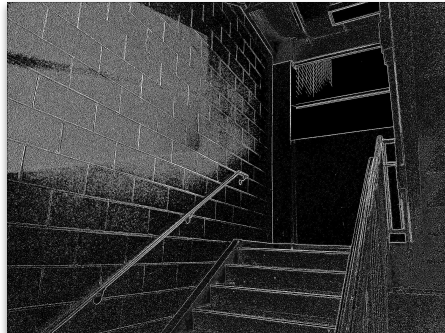
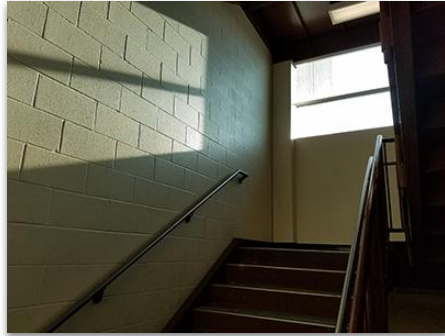
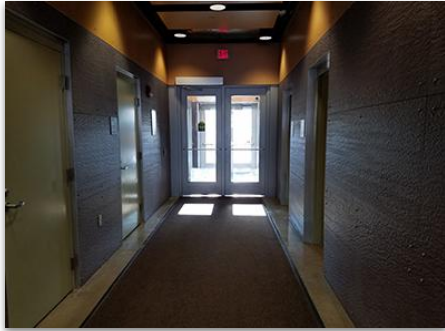
Edge Detection - Objects (Hi Res)



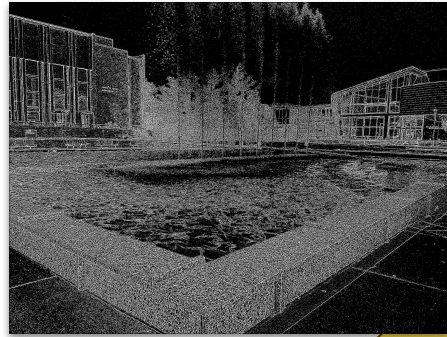
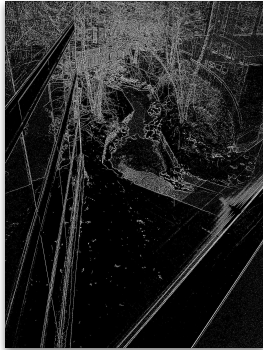
Edge Detection - Objects (Low Res)



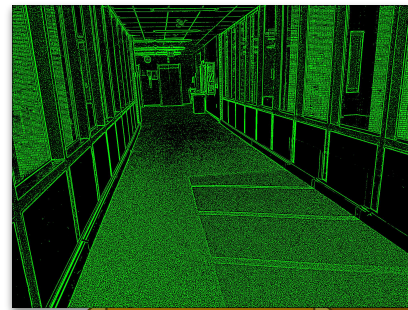
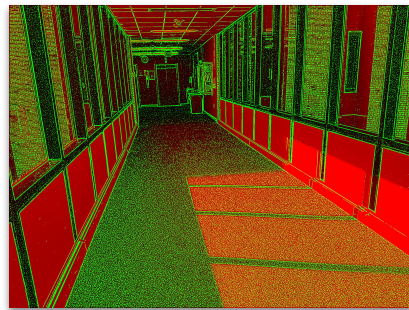
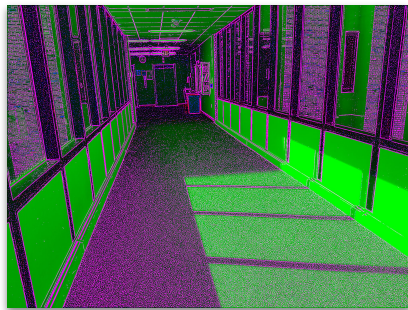
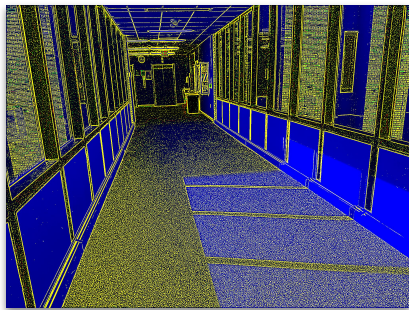
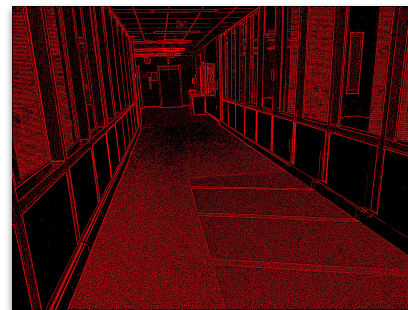
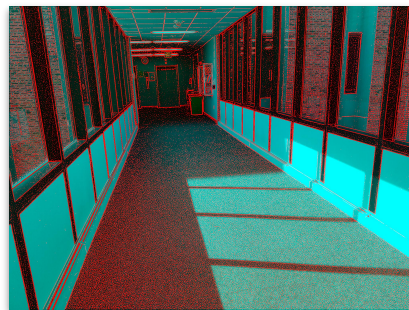
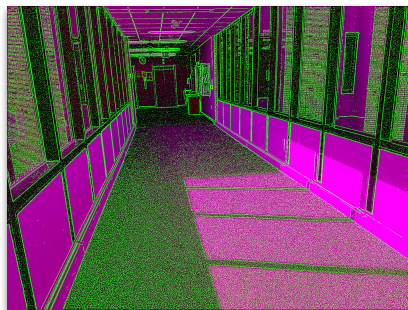
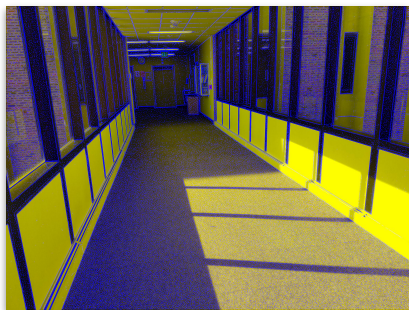
Edge Detection - Hallways



Edge Detection - Other



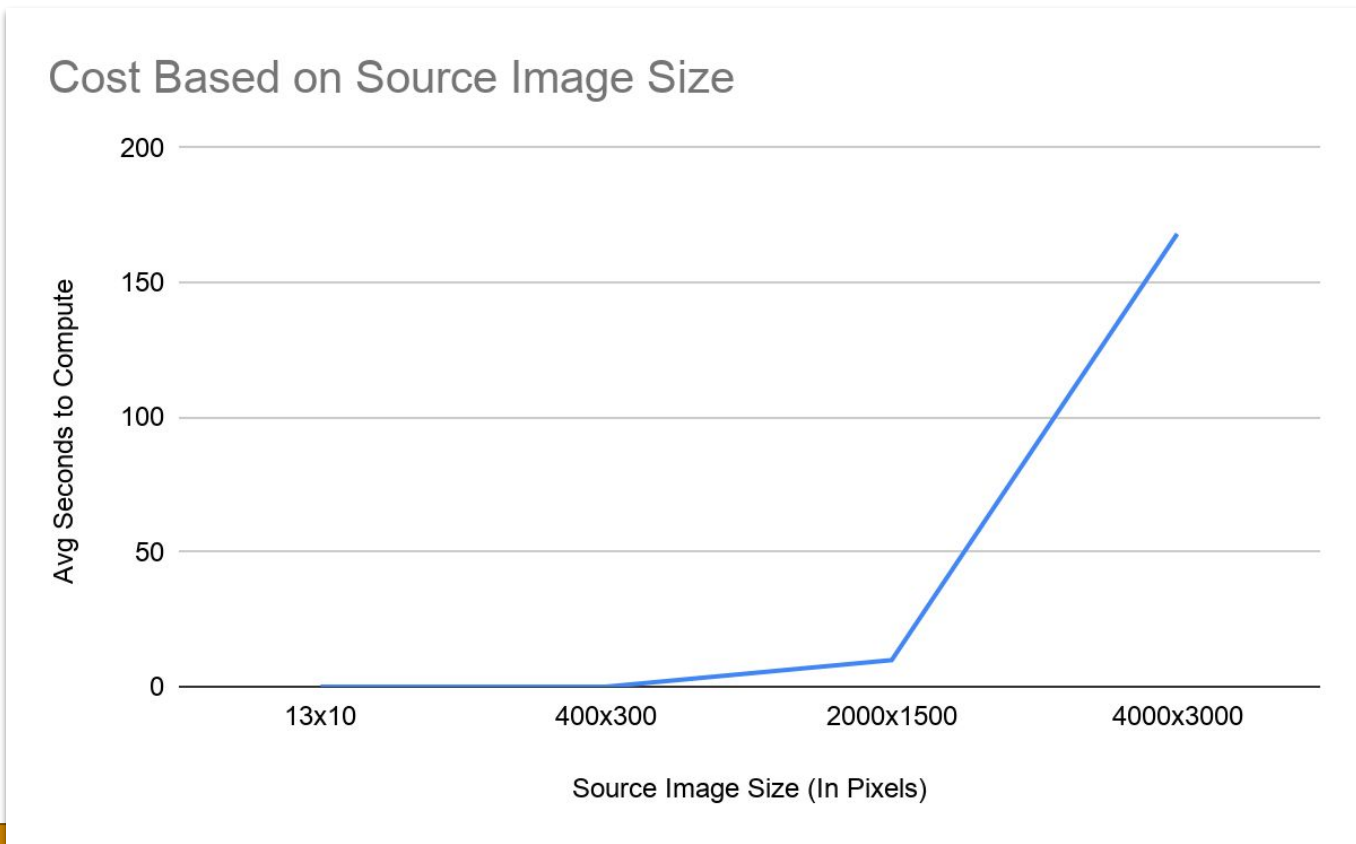
Edge Detection - Channel Impact



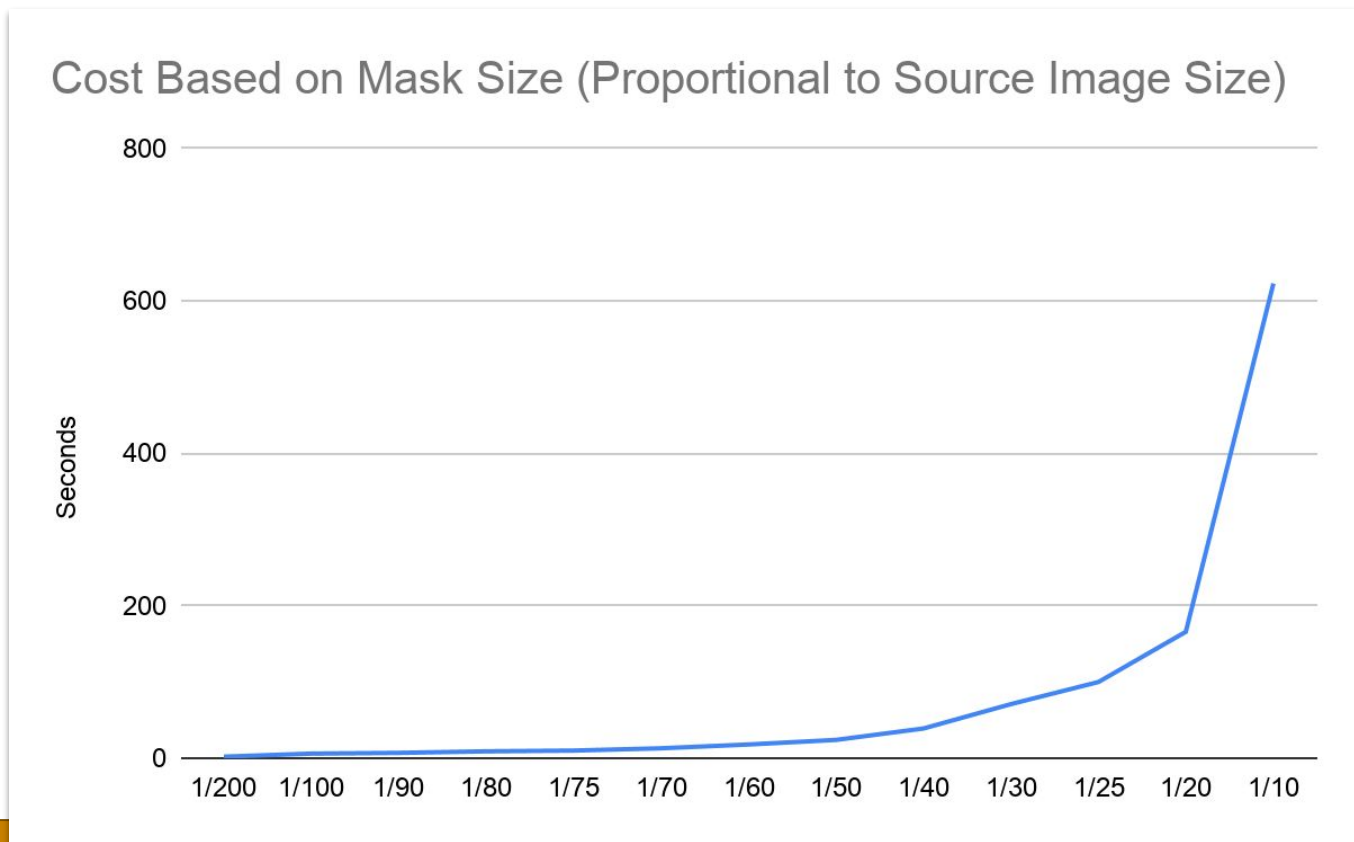


Optimizations

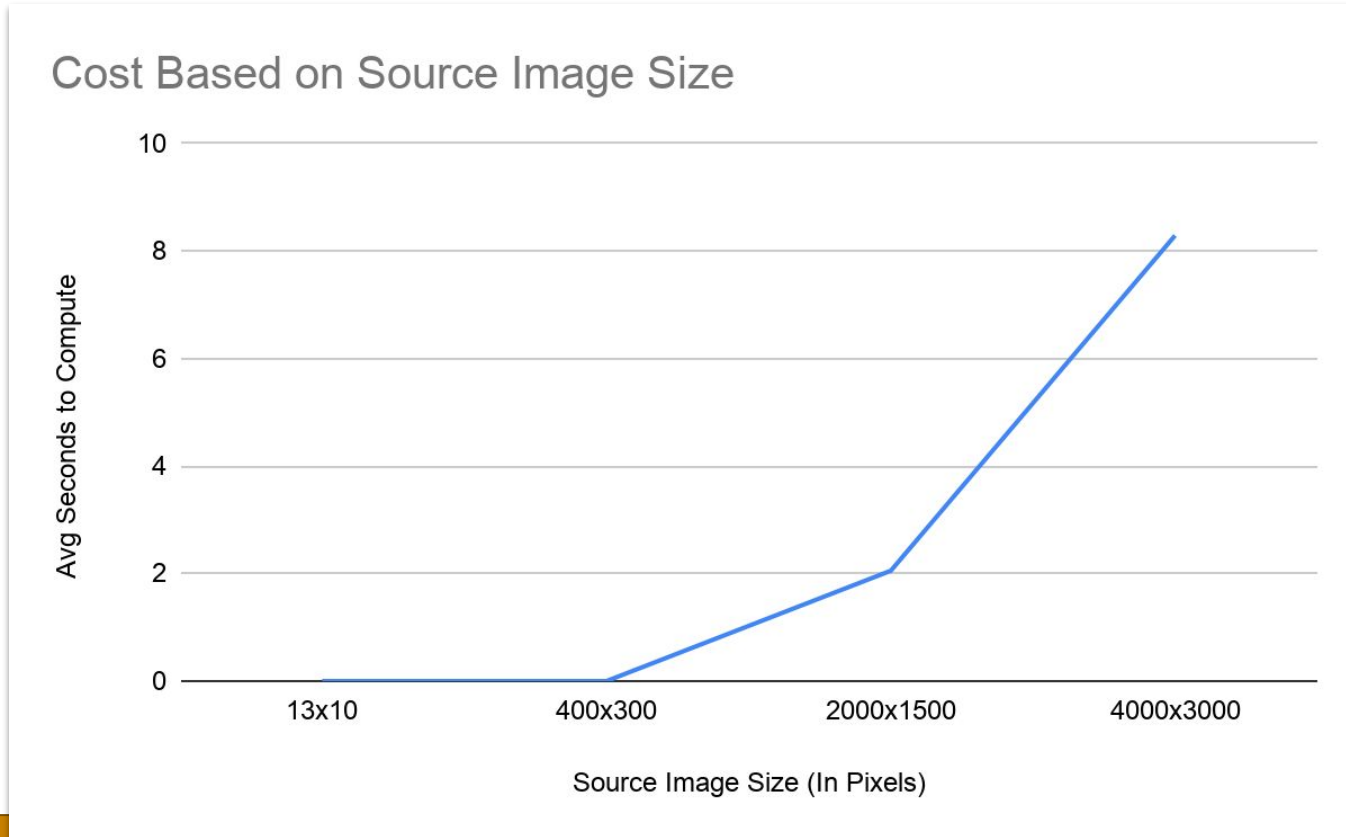
Gaussian Blur Times



Gaussian Blur Times



Edge Detection Times



Optimizations

- Could potentially:
 - Optimize memory usage, such as with tiling.
 - Reduce number of total computations, such as with Fourier Transform.
 - For Gaussian, implement smarter use of masking.
 - We can approximate the end result using a series of smaller masks.



Fast Fourier Transform

- Effectively, **Fourier Transform** converts data from the standard domain (time) to a new domain (frequency).
- Our original image and our mask each have a separate representation in the frequency domain.
- We can combine these two as we convert back to our original domain, which has the same as applying our mask.



Fast Fourier Transform

- Due to the speed of Fast Fourier Transform, this is very fast.
 - Ends up being faster than our original convolution algorithm, while achieving the same effect.
- For more info:
 - <https://www.youtube.com/watch?v=spUNpyF58BY>
 - <https://www.youtube.com/watch?v=gwaYwRwY6PU>



