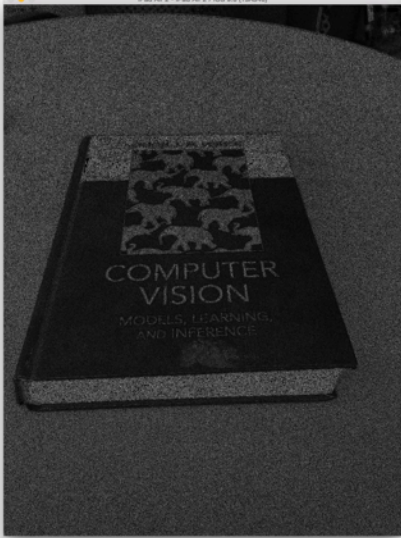
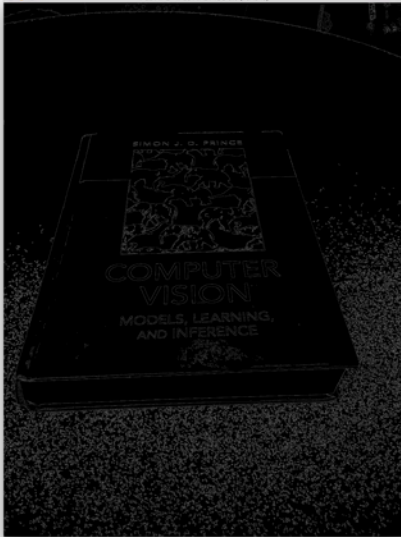
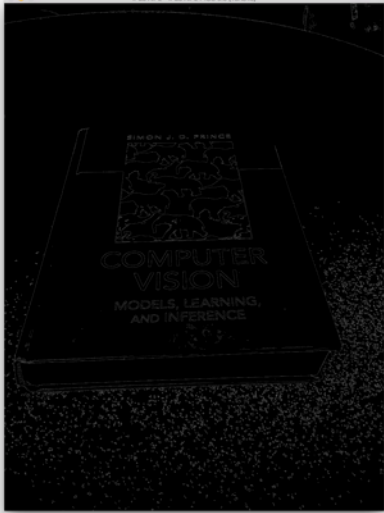
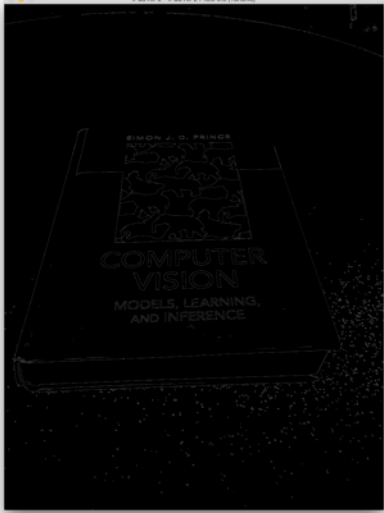
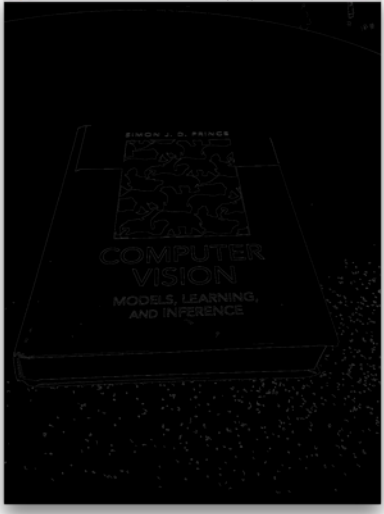


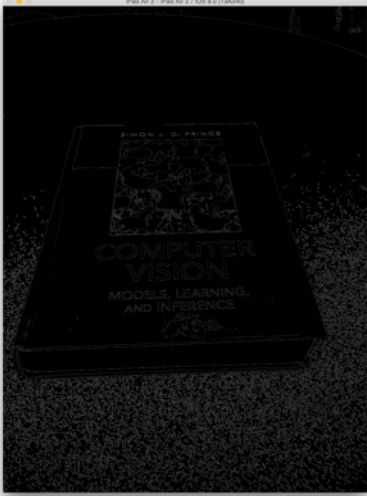
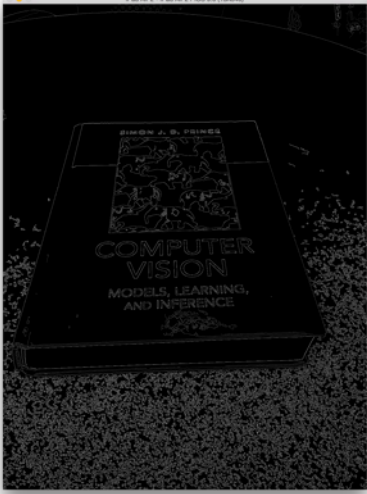

Interest Points, Edges, and the GPUImage Library

1a). Canny Edge Detection

Table 1: Canny Edge results using OpenCV and tweaking the different variables.

Variables	Resulting Canny Edge Image	Comments
Threshold = 1 GausBlur = (1.5,1.5) No resize		Basic first try. A lot of the edges, including the table surface is picked up.
Threshold = 50 GausBlur = (1.5,1.5) No resize		By increasing the lower threshold of the Canny detector, details farther away in distance were removed. This was good in terms of the tabletop false edges but was bad because it removed the top edge of the book.

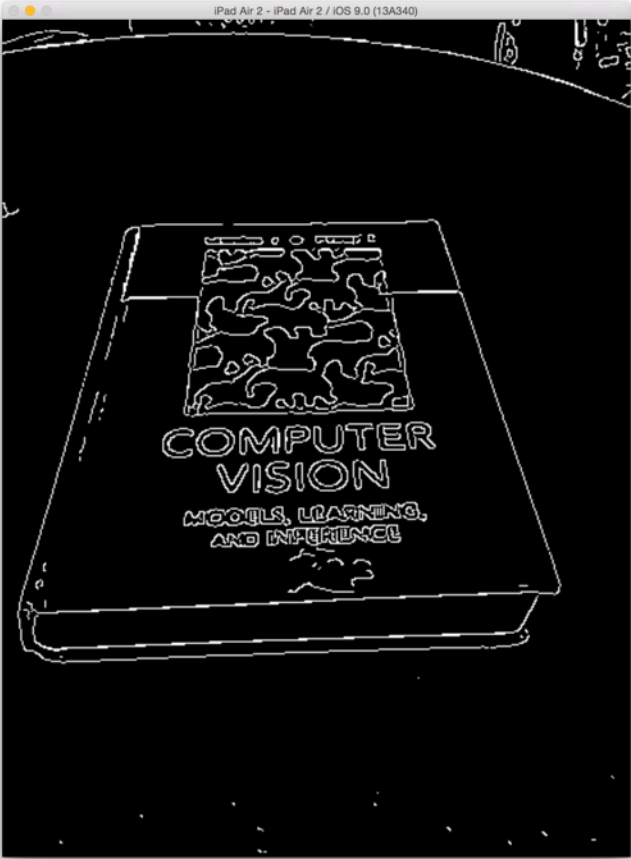
<p>Threshold = 75</p> <p>GausBlur = (1.5,1.5)</p> <p>No resize</p>		<p>More evidence of the case above when we increase the lower threshold of the Canny detector.</p>
<p>Threshold = 100</p> <p>GausBlur = (1.5,1.5)</p> <p>No resize</p>		<p>More evidence of the case above when we increase the lower threshold of the Canny detector.</p>
<p>Threshold = 50</p> <p>GausBlur = (5.9,5.9)</p> <p>No resize</p>		<p>When a higher Gaussian Blur kernel size is used, more details are removed.</p>

<p>Threshold = 25</p> <p>GausBlur = (5.9,5.9)</p> <p>No resize</p>		<p>With just threshold and GaussianBlur, we can't seem to remove enough of the false tabletop edges before removing the top edge of the book.</p>
<p>Threshold = 25</p> <p>GausBlur = (5.9,5.9)</p> <p>Resize factor = 0.5</p>		<p>When we add resize, we seem to be able to keep the top edge of the book while getting rid of more false tabletop edges.</p>
<p>Threshold = 25</p> <p>GausBlur = (5.9,5.9)</p> <p>Resize factor = 0.1</p>		<p>If we resize to just 10% of original size, then we can keep the edges of the book and the edge of the table without any false tabletop edges.</p>

Increasing the Gaussian blur removes some false matching of tabletop edges because the scale of the edge is required to be larger before it is flagged as an edge. Using the blur alone did not work because, as we saw in some of the examples listed above, it would remove the smaller scale items that are farther away from the camera (including the top edge of the book) while keeping the closer tabletop edges. Resizing performs a similar task in that it blows up the details so the smaller edges are discarded. The combination of these allows us to keep the large, solid lines such as the outline of the book.

1b). Canny Edge Detector using GPU Image

Table 2: prince_book.jpg using GPU Image's Lanczos Resample, Grayscale Filter, Gaussian Blur, and Canny Edge Detector function calls.

	<p>Blur radius (pixels) = 1</p> <p>Down scale (resize) = 0.17</p> <p>Lower Threshold = 0.11</p> <p>Upper Threshold = 0.33</p>
--	---

The advantages of using the GPU Image framework are two fold: (1) it makes use of parallel computation, and (2) it is easier to write. GPUs are very good at parallel calculations and, since we are applying oriented filters across height and width gradients, it speeds things up when we can do the computations in parallel. The code is also slightly easier to write because we can cascade the filters.

1c). Hough Lines



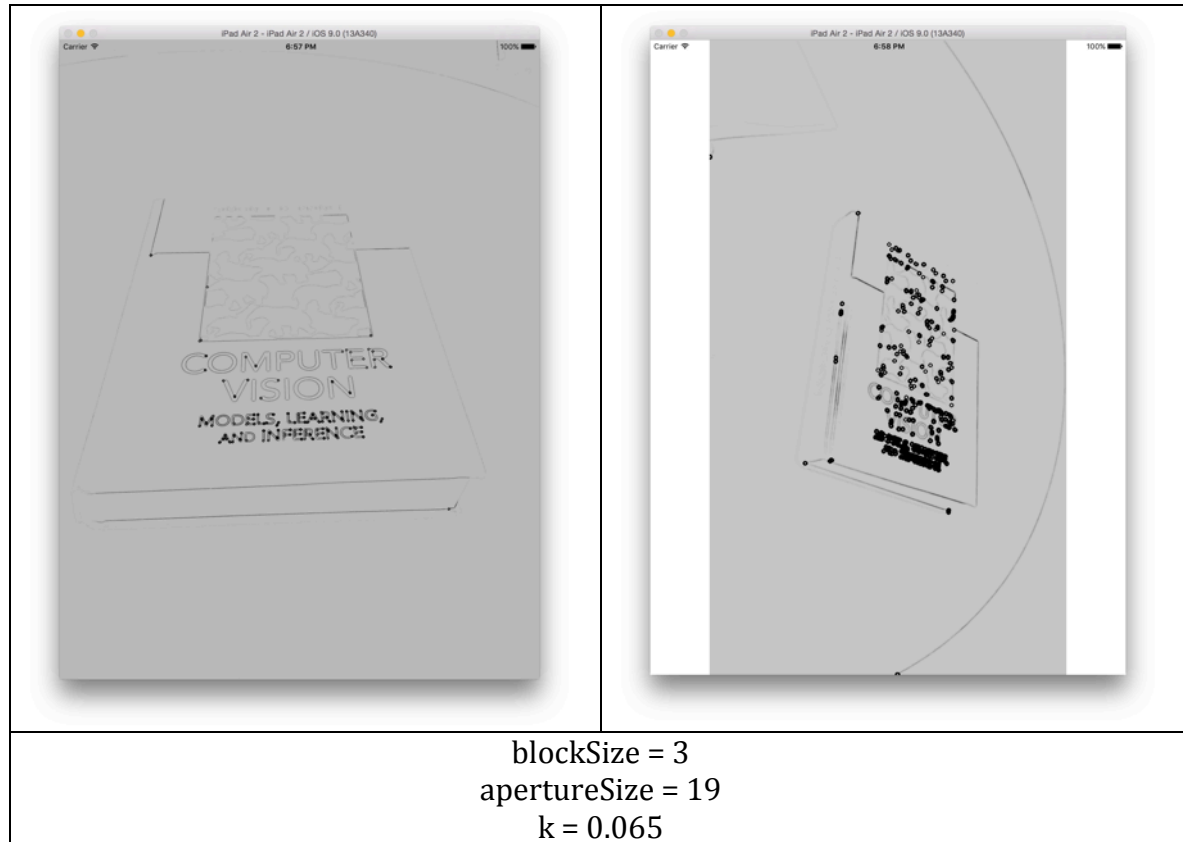
With only Canny Edge Detector.



Canny Edge Detector with Hough lines.

Hough transform uses polar coordinates ρ and θ to parameterize lines instead of m and b because, for any given point (x, y) , the family of lines that run through the point can be plotted on ρ and θ to create a curve. Then for multiple points and multiple curves plotted, if the curves all intersect at one point, then we know that all those points form a line at that given (ρ, θ) . We can also set a minimum threshold for the number of points that intersect before declaring that it is a line.

2a). Harris Corners with Estimate Planar Warp



Changing any of the variables from these values either resulted in very little corners being detected in prince_book.jpg or screens hanging/not rendering.

$$dst(x, y) = \det(H_{x,y}) - k \cdot tr(H_{x,y})$$

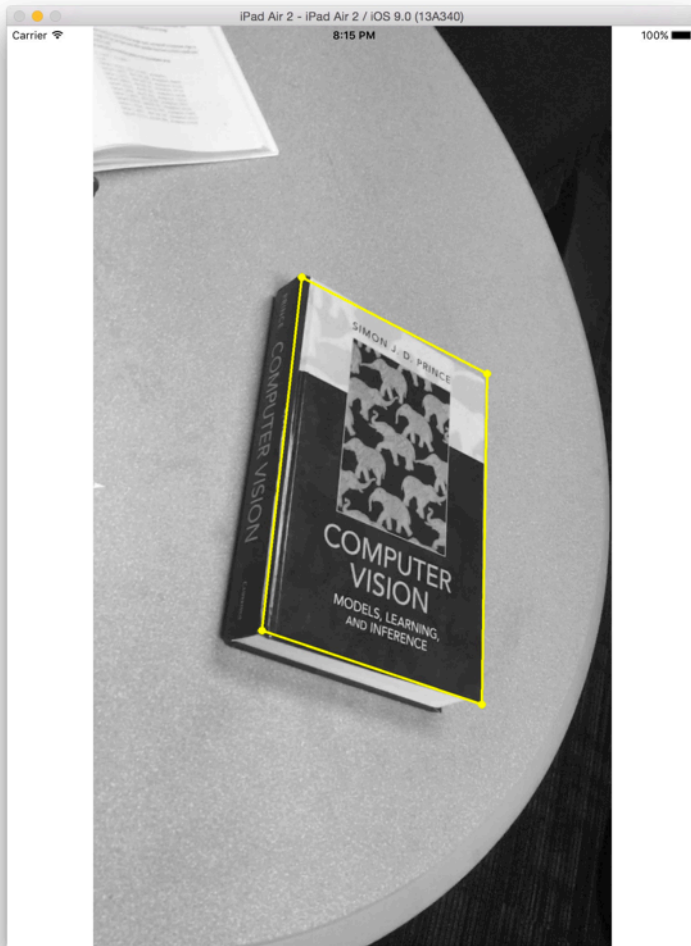
OpenCV uses the determinant and trace because it can get computationally expensive to calculate the eigenvalues of each potential corner.

2b). SIFT Feature Detector / Extractor



The SIFT interest point detector is preferable over Harris because it is scale invariant.

2c). RANSAC



```
Detected 1001 keypoints
Detected 1000 keypoints
matches: 511
max_inliers: 109
best_h:
  9.7461e-04  4.1179e-04  2.9939e-01
  6.1926e-04  2.4498e-03 -9.5413e-01
  2.1696e-07  8.5999e-07  1.7423e-03
```

3). The number of iterations of RANSAC we need to run in order to have 99% chance of computing an accurate homography with 50% initial matches is given by:

$$N = \frac{\ln(1 - p)}{\ln(1 - (1 - \epsilon)^r)}$$

where N is the number of iterations, p is the confidence that one sampling is error free, ϵ is the percent of outliers, and r is the number of points per sample.

$$N = \frac{\ln(1 - 0.99)}{\ln(1 - (1 - 0.5)^4)}$$

$$N = \frac{\ln(0.01)}{\ln(0.9375)}$$

$$N = 71.36$$

Thus, it will take about 72 iterations.