ColorCrack: Identifying Cracks in Glass

James Max Kanter
Massachusetts Institute of Technology
77 Massachusetts Ave
Cambridge, MA 02139
kanter@mit.edu

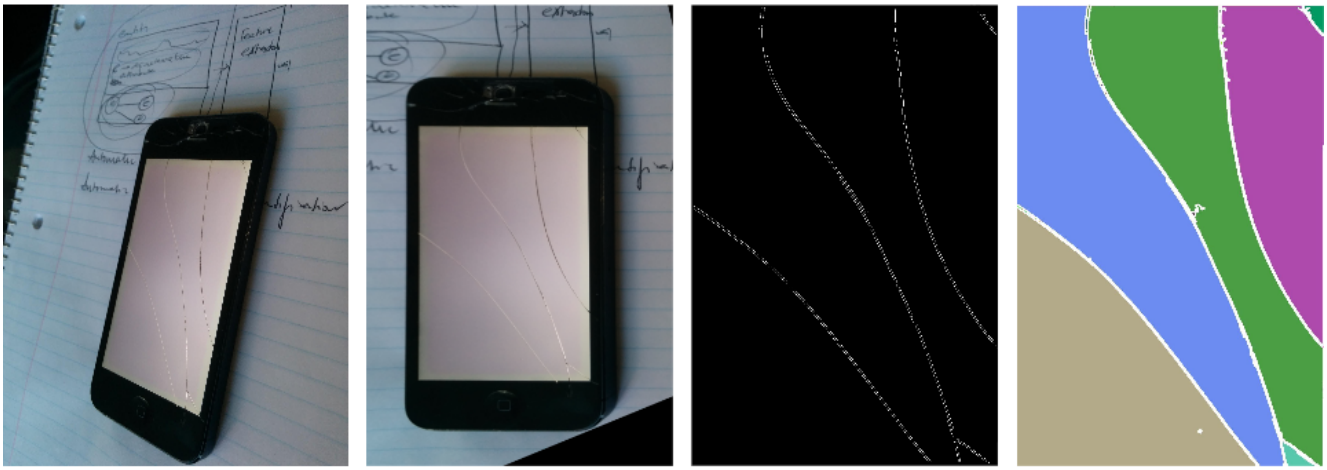


Figure 1: ColorCrack automatically identifies cracks in arbitrary angled photos of cracked screens. It does this in four main steps : (1) detect screen using HOG features (2) Transform screen by applying homophony (3) Find cracks using edge detector (4) Process result to connect cracks and color regions.

Abstract

Automatically identifying cracks in glass is an application of machine vision that has not been studied much before. It poses the unique opportunity to incorporate the physics of glass and how it cracks into a detection algorithm.

ColorCrack is an automated method of finding cracks in glass smartphone screens using insights from previous work in object and edge detection.The key to ColorCrack is the ability to automatically detect cracked screens from nearly any angle. This functionality allows the system to take advantage of the fact that cracks are more visible at certain angles to improve the detection accuracy.

The algorithm was developed and evaluated using a personally collected dataset. Initial results show that ColorCrack can typically find the screen correctly or with little error, as well as identify the main crack structures, efficiently enough for most applications.

In the future, the ColorCrack algorithm could be extended to other glass types for applications and has uses in fields such as material and forensic science.

1. Introduction

Machine vision is a powerful tool to automate a tasks currently performed by humans. However, what is instinctive for humans is not necessarily for a computer. One such task is identifying cracks in glass.

Objects that have glass and have the potential to be cracked are all around us. While it is simple to see cracks, it is harder to quantitatively characterize the cracks that form in glass.

Studying how glass break is especially relevant to scientists and engineers who study, design, and develop glass objects. For instance, in forensic science,researchers seeks to model how cracks form. In studies like ”Star-Shaped Crack Pattern of Broken Windows”, researchers analyse photographs of cracks by hand to develop physical models that infer the point of origin of a projectile[6]. In other fields like material science, scientists could benefit from large digitalized datasets on cracked glass. For example, imagine if the engineering team designing the screen on the next iPhone incorporated real world data on cracks in their design.

The problem poses challenges for machine vision in at least a few regards. First, the 3d nature of cracks means that they cannot be viewed from all angles. Second, the re-

flective and transparent nature of glass differentiates it from a lot of other work in vision. And finally, glass is used in varying manners the differ in both shape and scale which present difficulties for general solutions.

In this paper, I investigate two approaches to solving this problem and present the results.

2. Previous work

While I found no previous academic work specifically on automatic crack detection in glass, the problem relates to several efforts in the fields of object detection and edge detection.

2.1. Object Detection and HOG Features

A Histogram of Oriented Gradients (HOG) is a robust method for generating image features that has been shown to work well for object detection [2]. It is based on the idea that an image can be characterized well by the distribution of local intensity gradients or edge directions.

HOG is calculated for a given image by dividing it into small regions called cells that can be either rectangular or radial. Each of these cells is represented as a histogram of gradient directions of the pixels in that cell. In their paper, Dalal and Triggs show that overlapping these cells can improve performance[2].

To account for illumination and shadowing in a cell, the local regions are contrast normalized before using the HOG features. A common way to do this is to assemble nearby cells into "blocks" and then use the total energy of the block to normalize each cell.

HOG features are then used to train a machine learning model. A common approach is to train a support vector machine (SVM), which is a binary classifier that tries to optimizes a hyperplane between training points. Dalal and Triggs used this approach to get near perfect separation on the MIT pedestrian database[2].

2.2. Edge detection

Cracks in glass can be thought of edges in an image. Edge detection is at core of many machine vision tasks and has a lot of research behind. Early work by John Canny has been shown to sufficient for most edge detection tasks[4]. The Canny detector is based on the gradient of image pixel intensities calculated using the derivative of a Gaussian filter[1]. First, the algorithm smooths the image using a Gaussian filter to reduce noise. In the nonmaxium suppression step, the algorithms takes the gradient in multiple directions and scans the image to only keep pixels that are local maximum in some direction. Finally, the Canny detector uses a high threshold, T_1, and low threshold T_2, to classify pixels. Pixels above T_1 are considered strong edges, while pixels below T_2 are labelled as non-edges. Any pixel that is in between T_1 and T_2 is classified as an weak edge if it connects to a pixel above the high threshold.

While many methods since the Canny detector have used a gradient approach, there are other techniques to solve the problem using machine learning. Dollár and Zitnick use the fact that edges contain structures in the form of straight lines and T-junctions to train a random forest [3]. A random forest is an ensemble of decisions trees where a decision $f_t(x)$ classifies an image patch $x \in X$ using a binary split function

$$h(x, \theta_j) \in 0, 1 \tag{1}$$

These trees are combined using an ensemble method such as majority voting to classify the sample. Dollár and Zitnick apply this to images by predicting a 16 x 16 segmentation mask from a larger 32 x 32 input patch. To improve results they run this prediction at 3 different resolutions of the image and average the result. Finally, they sharpen the image by iteratively resigning pixels in the segmentation mask and averaging the results. The technique computes results an order magnitude faster than other methods while also achieving state of the art accuracy. This results in frame rates that are high enough for of real-time edge detection.

3. Crack Detection Problem Set-up

For this project, I focus on cracked smart screens because that is a dataset I could easily generate myself. However, through the process, ColorCrack was designed with thoughts towards the general case.

The problem of identifying cracks in glass in photographs can be broken down into two parts.

1. Finding the glass in a photograph
2. Finding the clacks in the glass

In the first step, the input is a photo of a non-occluded screen that can be at any angle. The algorithm used in this step needs to identify the location of the screen and transform the photo so the plane of the screen is perpendicular to the camera.

In the second step, the input is the corrected screen and it must output a binary image with the locations of cracks. Therefore, ColorCrack considers a pixel of glass to either be cracked or not cracked and nothing in between.

4. Step 1: Screen Extractions from images

The first stage of the problem is extracting the screen from an angled photograph of a phone. ColorCrack does not limit the input images to being taken in a certain way to keep the system flexible. Additionally, most cracks photograph better when taken at an angle. This means that allowing angled photos often produces better results.

To extract the screen from an angled image, I explored ways to identify the 4 corners in the image. Once the four corners are identified, then ColorCrack calculates the homography that transforms the input image to one where the plane of screen is perpendicular to the camera.

I experimented with two different approaches for screen extraction.

Detecting checkerboard pattern on screen

One way to detect the screen is to display a known image that is easy to detect. Therefore, the first approach I took was to display a checkerboard pattern on the screen before taking the photo.

The disadvantage of this approach is that it requires downloading a specific image to the phone before photographing it. Even more, this method may be impractical when extended to other glass scenarios where printing large checkerboards won't work. Even though this approach ultimately did not work, it was important for providing evidence that it is possible to capture the crack information even after applying an homography.

Matlab provides the function `dectectCheckerboard` that can find a checkerboard in a photo. This function does not return the full checkerboard, but from the information it does return ColorCrack computes the average square size. Using the average square size, the system extrapolates the location of the corners of the actual screen. Once these corners are identified ColorCrack calculates the homography and uses `imtransform` to get the warped image. The results of extracting the checkerboard pattern can be seen in Figure 2.

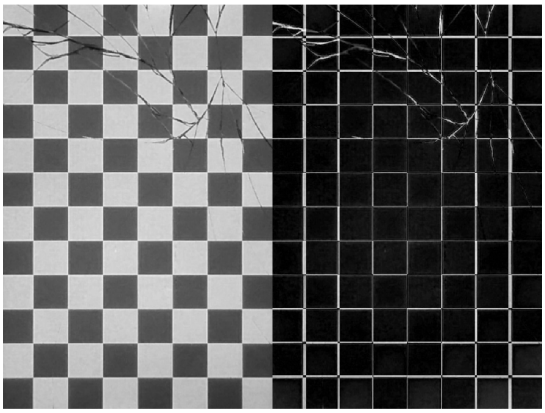


Figure 2: Left: The result of extracting screen using a checkboard pattern displayed an crack phone. Right: The result of trying to remove the checkboard pattern from the image. This approach did not work as well as hoped, so in the end ColorCrack used a different method.

The final step to prepare the image for crack detection is removing the checkerboard pattern. While the system knew the source checkerboard pattern that was displayed on the screen, the colors are slightly differently in the actual photo. To account for this, ColorCrack measures the the color of the squares in the captured photo and adjusts the source checkerboard photo. With the modified source checkerboard, ColorCrack subtracts that photo from the actual photo and takes the absolute value of the result.

The result of this can be seen in Figure 2. As you can see, this preserved the cracks, but left artifacts where the cells change color or the checkerboard was misaligned. While it may be possible to remove these artifacts, I perceived it as too difficult and took another approach to identifying the screen without having to use a specific image.

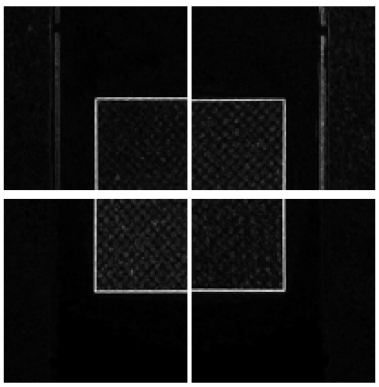
Screen detection using HOG features

Next, I redefined identifying the screen on a phone as the task of identifying each of the four corners. ColorCrack implements this by training a multi-class SVM on the HOG features of a training set of images.

Making training set

I assembled the training set manually using a custom tool. The tool displayed a photo and created 100px by 100px patches around the points I clicked. Using this tool I assembled a training set of 12 examples of each type of corner and 200 examples of non-screen corner objects. Figure 3 shows what these patches look like for each of the 4 corner types. In the non-screen corner objects I included examples of pictures of screens that did not contain corners as well as examples of corners that were not screens. These two changes improved performance on at least a few examples in my data set.

Figure 3: An example of four corner examples extracted from an image for training.



Before training the 5 one vs all SVM's on these patches

I had to decide on the cell size to use to computer my HOG features. I determined this by trying several different cell sizes as show in Figure 4. The visualization shows that a cell size of 32x32 does not seem to encode enough shape information. On the other hand, a cell size of 8x8 encodes more information than is probably necessary. This means that the dimensionality of the training examples would be very high, increasing time to train. The cell size of 16x16 seems to be a compromise that limits the size of the resulting vector while encoding enough information.

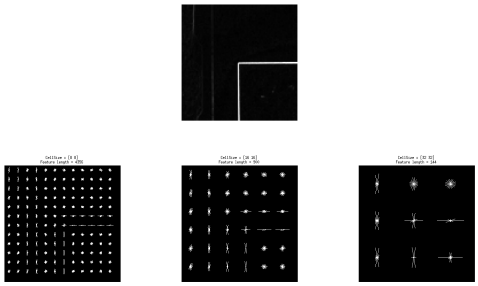


Figure 4: Visualization of the HOG features for for 3 different cell sizes. ColorCrack uses a cell size of 16x16 to balance capturing shape information and limiting feature size. Left to right cell sizes: 8x8, 16x16, 32x32.

Using image gradients for training

In the course of training, I discovered that taking the gradient of the image before extracting HOG features improved performance. This might be because the sharp change in color between glass and the border around the glass is a defining feature of glass that can be exploited for better performance. Figure 5 shows the result of this addition to the the processing pipeline.

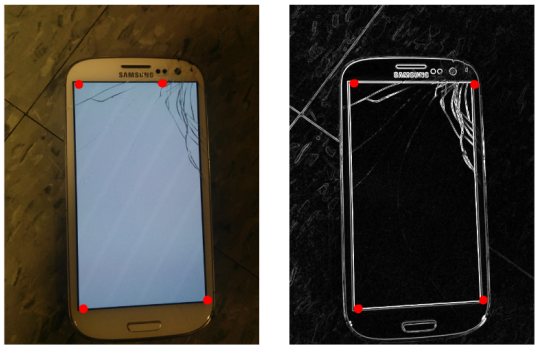


Figure 5: The result of using the image gradient to calculate HOG features. The classification of the upper right corner becomes noticeably more accurate.

Training SVM's and Corner Detection

With the features determined, ColorCrack trains several SVM's to detect each type of corner – upper left, upper right, lower right, and lower left – as well as an SVM for non corner objects. They were training in a one vs all fashion.

To find a corner, ColorCrack uses a sliding window of 100px by 100px that skips 10 pixels at a time to assemble a list of candidate coordinates. The candidate coordinate that has the highest score for each corner type gets labelled as that corner. While the corner detection generally worked well, it did not always perform as well as necessary for step 2. In cases like the one in Figure 6, I manually picked out the corners in order to evaluate the performance of just crack identification.

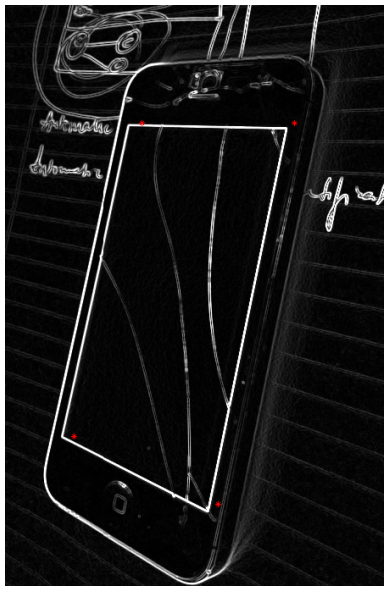


Figure 6: An example of when ColorCrack did not properly identify the location corners. This error may be too much depending on the application.

5. Step 2: Crack identification

Finding the cracks in the transformed image happens in 4 steps: convert to grayscale and apply Gaussian blur, run edge detection, filter out noise, and dilate then erode edges. The results at each step of of this process are shown in figure 7. As mentioned above, to access the performance only of this step, I manually selected corners when step 1 did not not perform well enough.

Convert to grayscale and apply Gaussian blur
This is a preprocessing step. Color data is not used by ColorCrack to finding cracks, so the input image can be

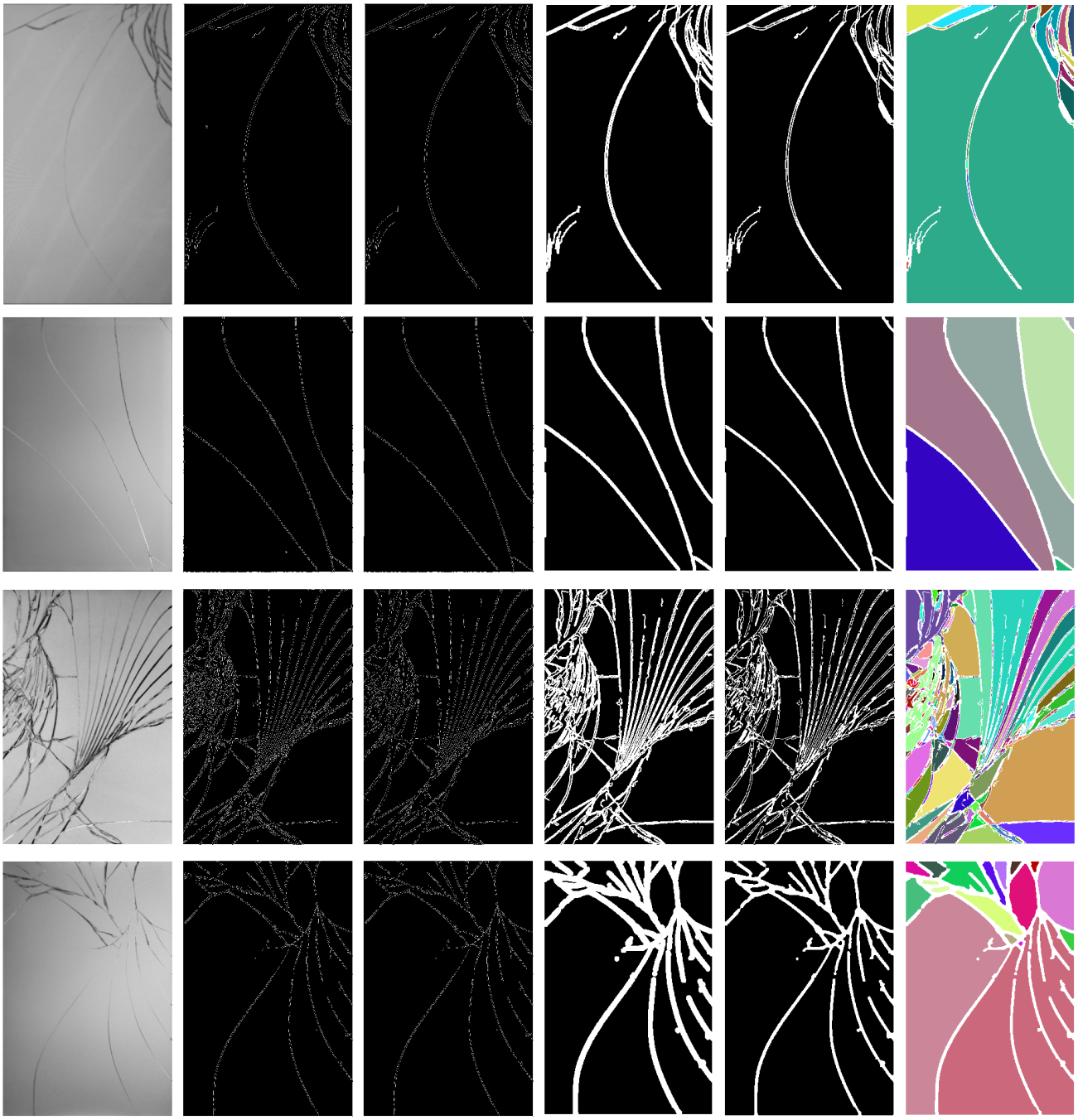


Figure 7: The results of applying ColorCrack's crack detection algorithm to 4 sample photos. From left to right: input image, result of applying Gaussian blur and Canny edge detector, result of removing regions that are too small to be cracks, result of dilation, result of erosion, result of coloring in regions. Small tweaks were made to the parameters of each image to achieve optimal results.

converted to grayscale. After that, ColorCracks applies a Gaussian blur to the image. Cracks are very defined parts of the input image, so blurring the photos preserves the crack information while removing noise that might be falsely identified as a crack.

Run edge detection

ColorCrack uses the Canny edge detector to locate cracks.

Cracks continue to form until they hit another crack [5]. This creates the branching pattern that the weak and strong edge threshold the canny detector detects well. Thus, the Canny edge detector is a good choice because the physics of how cracks form is similar to the mechanics of the Canny edge detector.

However, it is important to set the threshold for strong and weak edges in the Canny edge detector. ColorCrack modifies the default values of Matlab's Canny edge detector implementation based the insight that there will be at least on crack in the image that is very easy to recognize. This means that the threshold for strong edges is set higher than the default.

Filter out noise
After running the edge detector, there are sometimes "cracks" that are just noise in the image. To filter this out, ColorCrack uses the observation that cracks are typically large continuous regions. The Matlab function `bwareaopen` is used to remove regions that contain too few pixels to be cracks.

Dilate edges then erode edges
As mentioned above, cracks form until they hit another crack. To fulfil this constraint, ColorCrack tries to connect the cracks the Canny detector finds by dilating the pixels of the crack using `imdilate` and a disk structuring element. After applying the dilation to connect cracks, the Matlab function `imerode` with a smaller disk structuring element is applied to shrink the detected cracks, while leaving the new connections in tact. As an added benefit, the dilation and erosion has the effect of filling in the area between the outsides edges of the crack, which is the desired behavior.

6. Evaluation

ColorCrack was evaluated using 8 different photos of cracked screens. Below are the highlights of what worked and thoughts on how to improve what did not work.

What worked
ColorCrack generally correctly found corners or found them with small errors. This worked on a relatively small dataset of just 248 images, which speaks to the robustness of HOG features and SVM's.

ColorCrack always found the most prominent cracks in a photo and often found the weaker connecting cracks. This is likely a result of the fit between the crack identification problem and the Canny Edge detector. It struggled on the higher resolution cracks and very weak cracks, but requiring accuracy in these regards depends on the exact application.

ColorCrack is performant enough to be used on large sets of data. In a reasonable amount of time (hours), it could be applied to high speed photography or thousands of photos. This is largely because of the efficient implementation of all steps of the process in Matlab.

What did not worked and thoughts on how to improve
When ColorCrack incorrectly identifies the corner locations it heavily affects the homography calculation. A way to improve this would be to have highly ranked candidate points "vote" on where the corner should be. Another solution might be to implement logic that selected the highest ranking four points that fulfilled the constraints of how they could be arrange (e.g an upper corner cannot be below a lower corner).

Another issue is that even when taking a photo at an angle, cracks sometime do not show up clearly. In this case, it is impossible to identify every crack. A possible solution to this could be to compute image correspondences between multiple image a different angles and combine the results.

Finally, the next step is to apply ColorCrack to other uses of glass. In particular, it would be useful to explore how much tweaking is necessary on a dataset of crack windows.

References

[1] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, Nov 1986.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[3] P. Dollár and C. L. Zitnick. Fast edge detection using structured forests. *CoRR*, abs/1406.5549, 2014.

[4] B. Green. Canny edge detection tutorial, 2002.

[5] B. of Criminal Apprehension. Glass, 2014.

[6] N. Vandenberghe, R. Vermorel, and E. Villermaux. Star-shaped crack pattern of broken windows. *Phys. Rev. Lett.*, 110:174302, Apr 2013.