In [1]:
```
%reset -f
```

In [2]:
```
import numpy as np
from PIL import Image
```

In [3]:
```
def read_image(image_name,method):
    image = Image.open(image_name)
    image = image.convert(method)
    image_data = np.array(image)
    image_data_row = len(image_data)
    image_data_col = len(image_data[0])
    return (image_data,image_data_row,image_data_col)
```

Honglu Xu
EECS531
HW 1
Due 02/21/2018

2. Edge detector

In this problem, I will use a function in OpenCV. It will detect the edges aautomaticly and return the detected edges as an image.

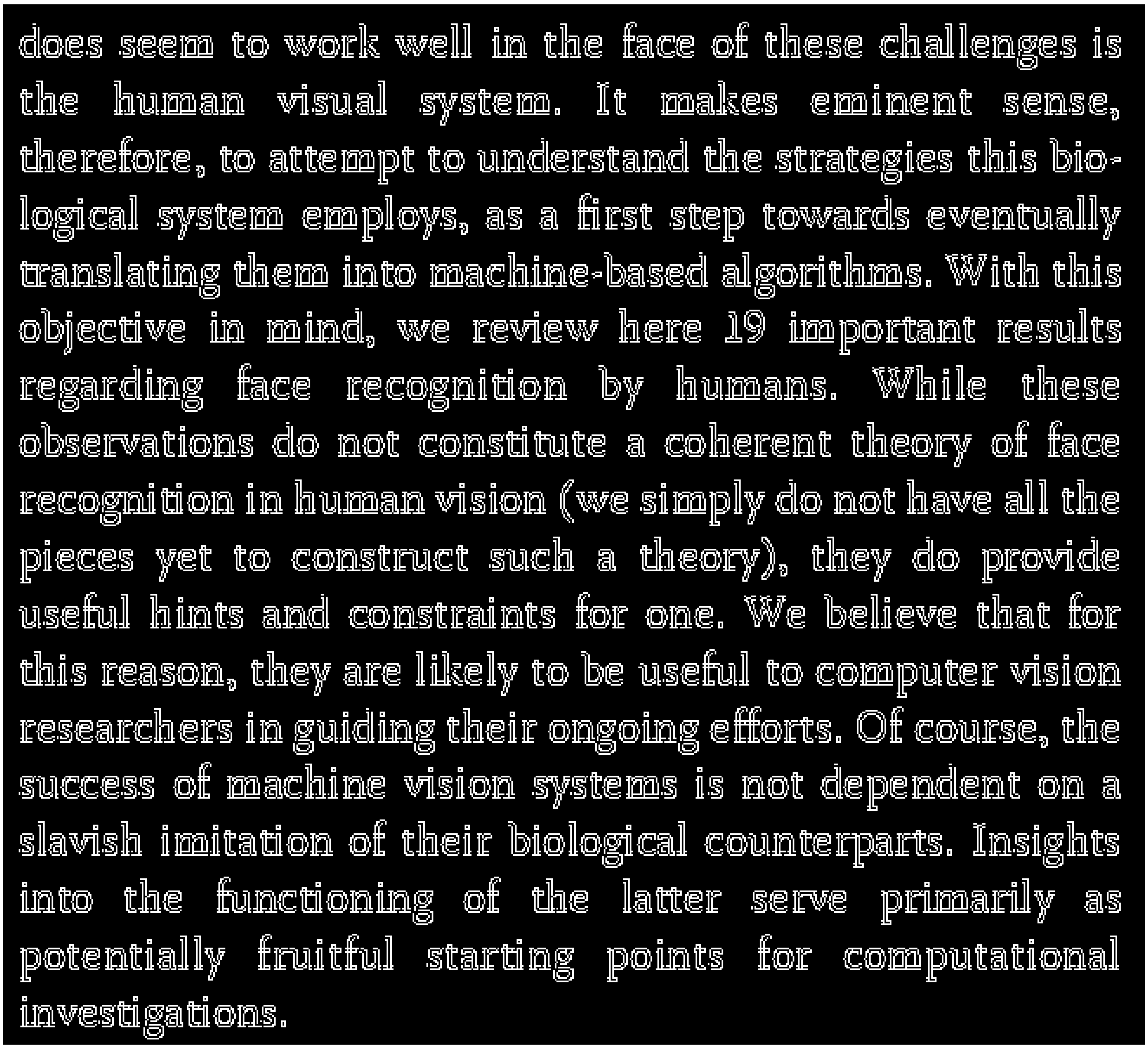
In [4]:
```
import cv2
# import matplotlib.pyplot as plt
```

I will start from the materials we have, which is the 'charaters.png'.

In [5]:
```
image_data,_,_ = read_image('characters.png','L')

edges = cv2.Canny(image_data,100,200)

img = Image.fromarray(edges, 'L')
# img.show()
img.save('edge_test1.png')# read the image
```


does seem to work well in the face of these challenges is the human visual system. It makes eminent sense, therefore, to attempt to understand the strategies this biological system employs, as a first step towards eventually translating them into machine-based algorithms. With this objective in mind, we review here 19 important results regarding face recognition by humans. While these observations do not constitute a coherent theory of face recognition in human vision (we simply do not have all the pieces yet to construct such a theory), they do provide useful hints and constraints for one. We believe that for this reason, they are likely to be useful to computer vision researchers in guiding their ongoing efforts. Of course, the success of machine vision systems is not dependent on a slavish imitation of their biological counterparts. Insights into the functioning of the latter serve primarily as potentially fruitful starting points for computational investigations.

We can see it worked very well.
The Canny detector is a very good detector, there is a reference about how it works:
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html
(https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html)
In general, it will firstly use the Gaussian filter, which we have already covered in the problem one, to get away the noise.
Then, it will use some edge filters to to detect the edges. The edge filters are similar to the Sobel filter.

In [13]:
```
G1 = np.array([[-1,0,-1],[-2,0,-2],[1,0,1]])
print("G1:\n",G1)
G2 = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
print("G2:\n",G2)
```

```
G1:
 [[-1  0 -1]
 [-2  0 -2]
 [ 1  0  1]]
G2:
 [[-1 -2 -1]
 [ 0  0  0]
 [ 1  2  1]]
```

We can see that G1 is for the vertical edge detector, and G2 is for the horizontal edge detector.
Then, we can use G1 and G2 to calculat the edge direction.

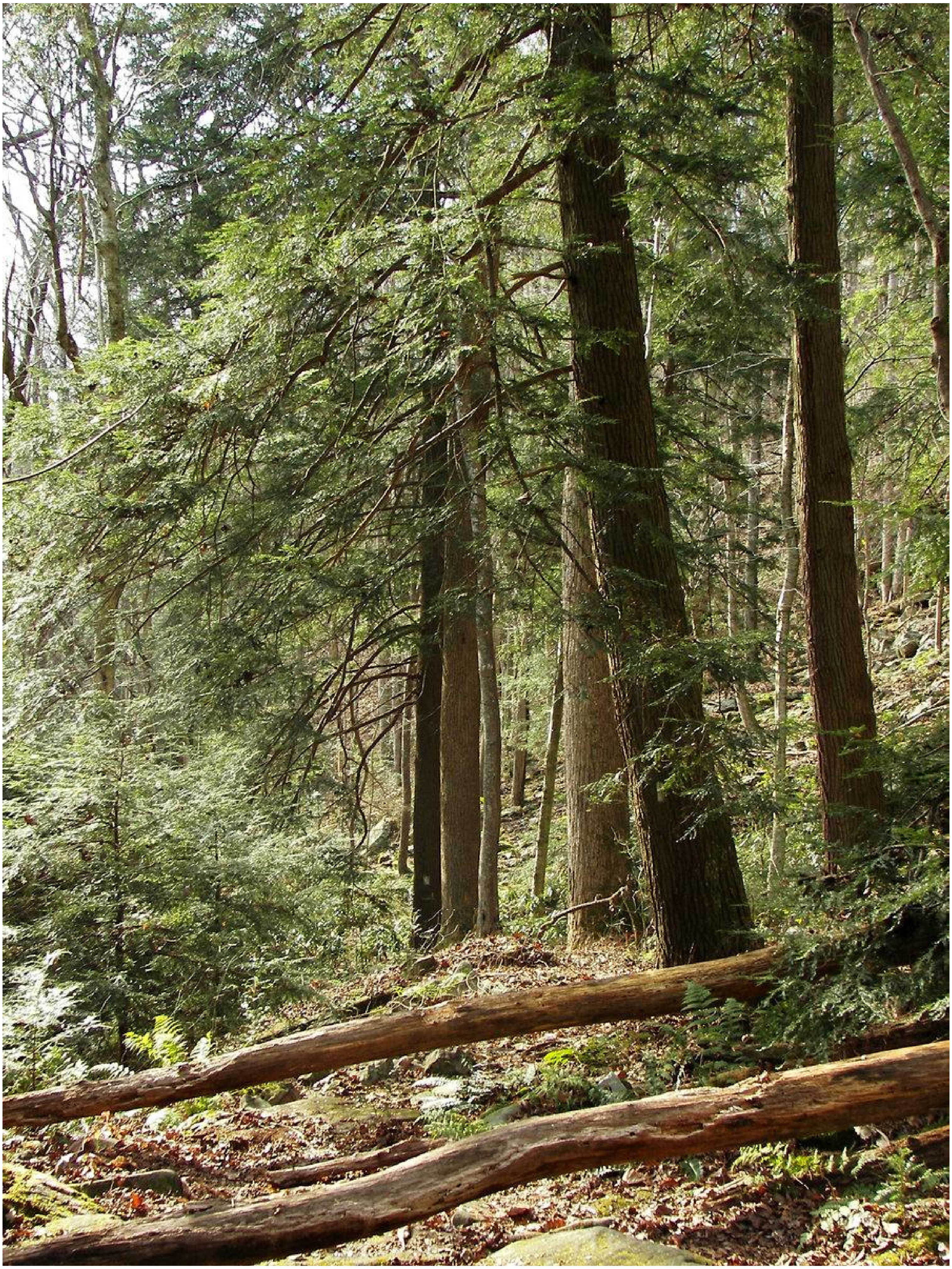
$$\theta = arctan(\frac{G2}{G1})$$

And also the gradient

$$G = \sqrt{G1^2 + G2^2}$$

By the document, the direction will be rounded to 0,45,90 or 135.
After that, it will apply the Non-maximum suppression and use to threasholds to determine whether this edge is the edge that the user wanted.
They upper threasholds and lower threaholds. The upper threashold is to confirm some obviously edges, and the lower threashold is to pick out some none edge pixels. If the gradient G is between the upper and lower threaholds, it will check whether it is connected to the "obviously edges", if it does, then this will be considered to be edges, too.

However, this image only contains the black and white elements, we can try some RGB images that see if that works.
I found an image from google search.



Let's turn it to Grayscale iamges so we can apply it to the filter.

In [6]:
```
image_data,_,_ = read_image('forest.jpg','L')

img = Image.fromarray(image_data, 'L')
img.save('forest_gray.png')
```

Now we can apply the function and see the results.

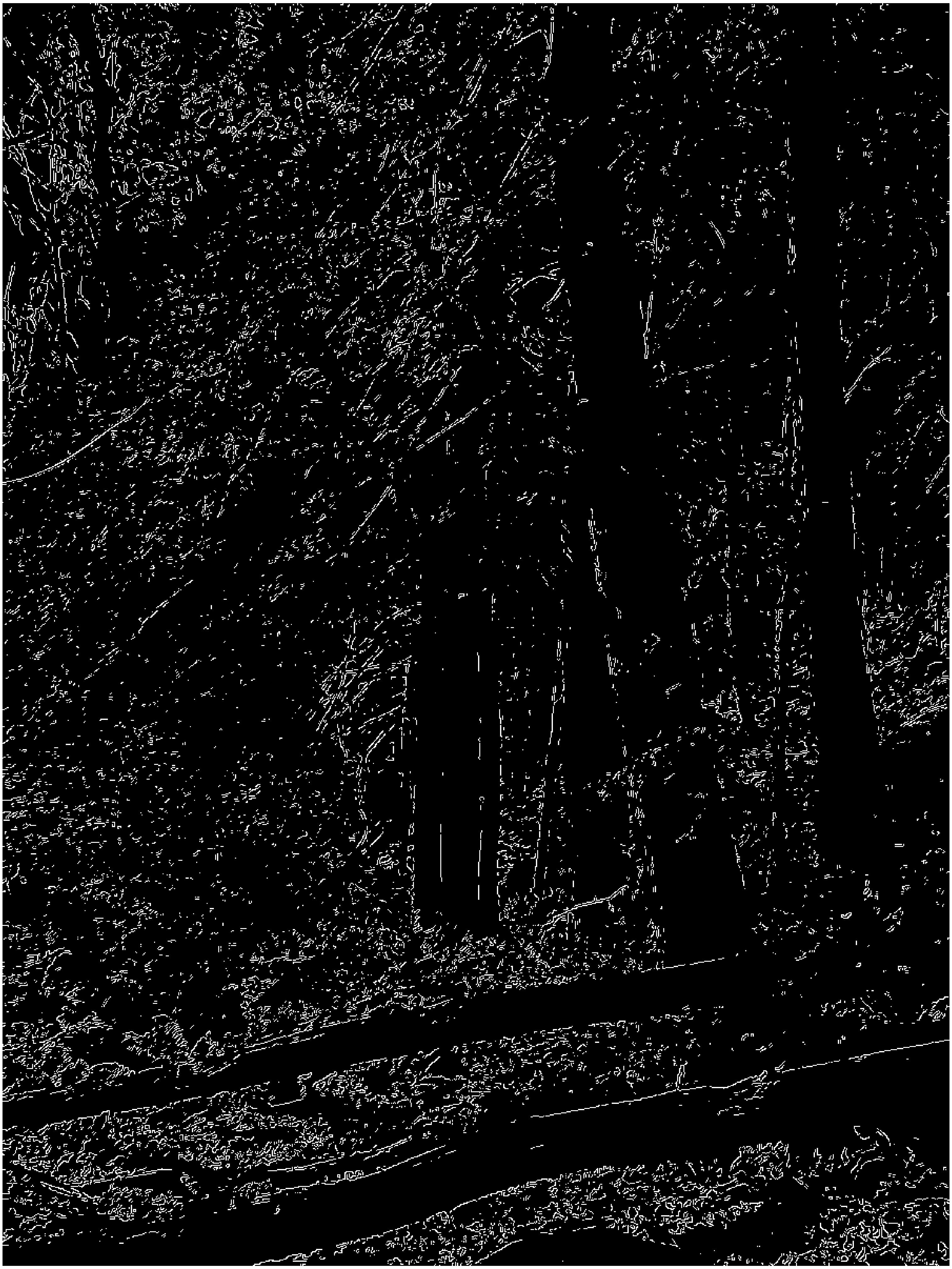
```
In [7]: edges = cv2.Canny(image_data,100,200)

        img = Image.fromarray(edges, 'L')
        # img.show()
        img.save('forest_edge1.png')
```

We can see this time, it surly covered too many edges. We need to adjust the threshold so only the main edges will be detected.

```
In [8]: edges = cv2.Canny(image_data,600,700)

        img = Image.fromarray(edges, 'L')
        # img.show()
        img.save('forest_edge2.png')
```



After saveral adjustments, I found some thresholds that are reasonable. We can see there are still too many edges that are not the main edges, but we can tell shapes of the elemensts in this image now.