SciComp with Py

Gradients & Edge Detection

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

- Gradients
- Edge Detection with Gradients
- Edge Detection with OpenCV Canny Algorithm
- Visual Comparison of Gradient and Canny Algorithms and the Effects of No-Blurring and Blurring

Gradients

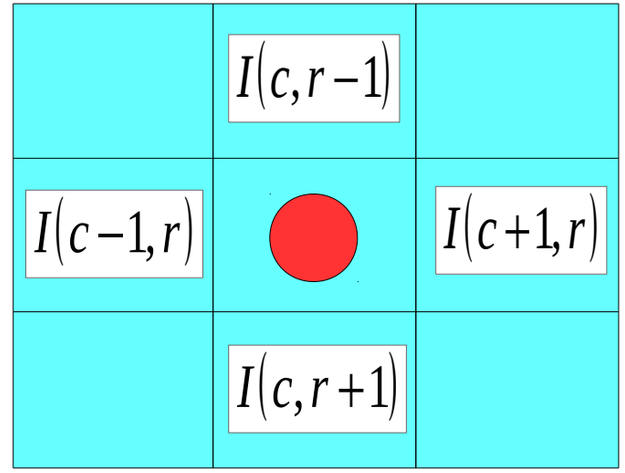
Gradients

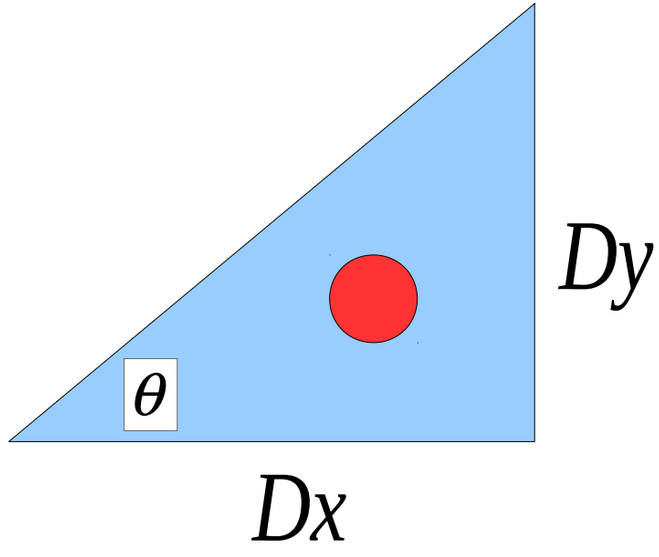
- Gradients are changes in image intensity/color
- Gradients, if viewed as vectors, have directions and magnitudes
- Gradients can be computed for each pixels or for image regions

Vertical & Horizontal Changes: *Dy* & *Dx*

$$Dy = I(c, r-1) - I(c, r+1)$$

$$Dx = I(c+1, r) - I(c-1, r)$$





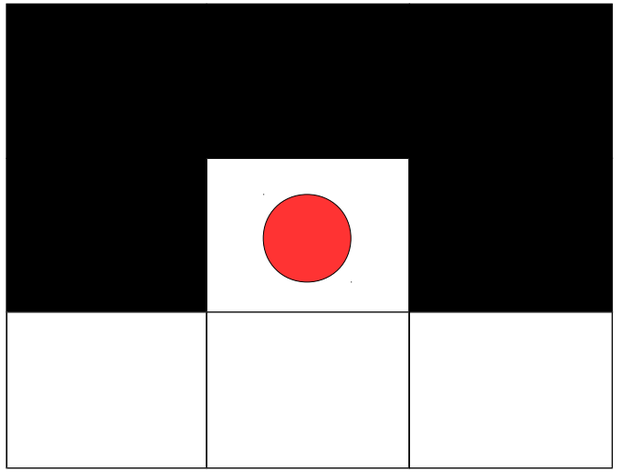
$\| G \| = \sqrt{Dy^2 + Dx^2}$ is the gradient's magnitude at $I(c, r)$

$\theta = \tan^{-1}\left(\frac{Dy}{Dx}\right)$ is the gradient's orientation

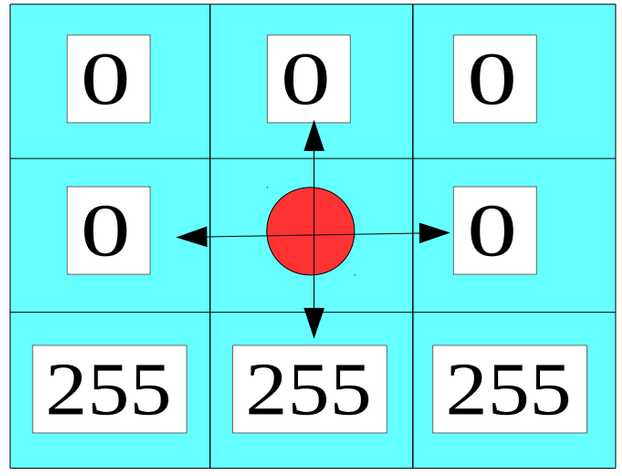
What if $Dx = 0$? In this case, we can set Dx to some small default value, e.g., 1.

Example

Image



Pixel Values



$$dy = 0 - 255 = -255; dx = 0 - 0 \approx 1$$

$$\| G \| = \sqrt{(-255)^2 + 1^2} = 255.00196078 \approx 255$$

$$\theta = \left(\tan^{-1}\left(\frac{-255}{1} \right) \right) \frac{180}{\pi} = -89.775311^\circ \approx -90^\circ$$

Edge Detection with Gradients

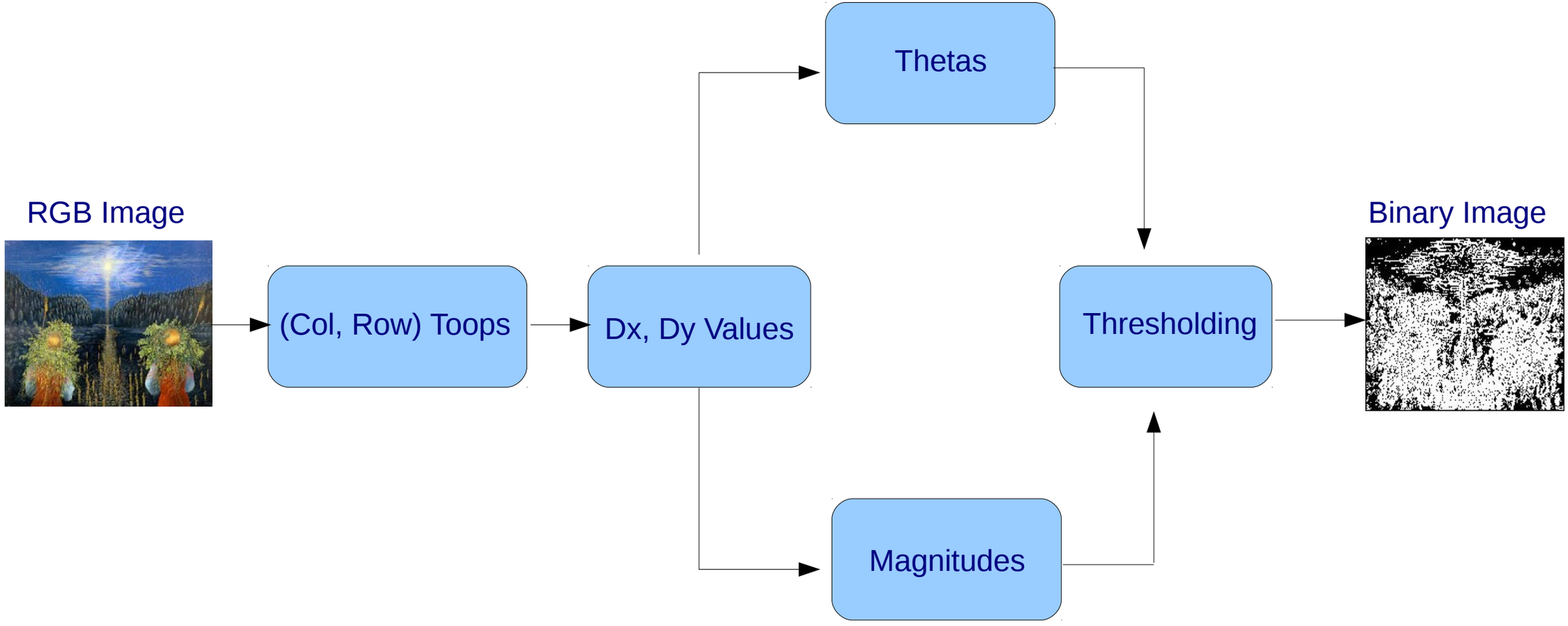
Implementation Steps

- Grayscale RGB pixels with relative luminosity
- Compute ***Dy*** and ***Dx*** at each pixel
- Compute gradient's magnitude and orientation for each pixel
- Implement an edge detection generator pipeline

PIL vs. OpenCV

- We will use PIL (Python Image Library) in this implementation
- PIL is for Python 2
- PILLOW is PIL for Python 3
- PIL/PILLOW is a much simpler version of OpenCV
- It is great for rapid prototyping and much easier to install than OpenCV

Edge Detection Generator Pipeline



Generating (Column, Row) Toops

Take the number of columns and rows in the image and generate each legitimate (c, r) toop:

```
def gen_cr(num_cols, num_rows):
    r, c = 0, 0
    while r != num_rows:
        c = c % num_cols
        yield (c, r)
        if c == num_cols - 1: r += 1
        c += 1
```

Grayscaling RGB Pixels

In PIL, pixels are RGB. This function takes a 3-toop rgb = (r, g, b) and converts it to grayscale:

```
def luminosity(rgb, rcoeff=0.2126, gcoeff=0.7152, bcoeff=0.0722):
    return rcoeff*rgb[0]+gcoeff*rgb[1]+bcoeff*rgb[2]
```

Computing Dy & Dx

```
## im is a PIL Image object.
def is_in_range(im, cr):
    return cr[0] > 0 and cr[0] < im.size[0]-1 and cr[1] > 0 and cr[1] < im.size[1]-1

## In PIL, c = x, r = y
def rgb_pix_dy(rgb_img, cr, flumin, default_delta):
    if not is_in_range(rgb_img, cr): return default_delta
    c, r = cr
    dy = flumin(rgb_img.getpixel((c, r-1))) - flumin(rgb_img.getpixel((c, r+1)))
    if dy == 0:
        return default_delta
    else:
        return float(dy)

def rgb_pix_dx(rgb_img, cr, flumin, default_delta):
    if not is_in_range(rgb_img, cr): return default_delta
    c, r = cr
    dx = flumin(rgb_img.getpixel((c+1, r))) - flumin(rgb_img.getpixel((c-1, r)))
    if dx == 0:
        return default_delta
    else:
        return float(dx)
```

Computing Gradient's Magnitude & Orientation

```
def grad_magn(pdx, pdy):
    return math.sqrt(math.pow(pdx, 2.0) + math.pow(pdy, 2.0))

## if pdy == pdx, we return a default_theta value outside of [-pi, pi]
## gradient orientation
def grad_theta(pdx, pdy, default_delta, default_theta):
    if pdy == pdx == default_delta: return default_theta
    th = math.atan2(pdy,pdx)*(180/math.pi)
    if th < 0:
        return math.floor(th)
    elif th > 0:
        return math.ceil(th)
    else:
        return th
```

Constructing Edge Generator Pipeline

```
def detect_edges(rgb_img, ftheta=grad_theta, fmagn=grad_magn,
                 fpixdx=rgb_pix_dx, fpixdy=rgb_pix_dy, flumin=luminosity,
                 default_delta=1.0, default_theta=-200,
                 theta_thresh=360, magn_thresh=20):
    output_img = Image.new('L', rgb_img.size)
    num_cols, num_rows = rgb_img.size
    gdxdy = ((fpixdx(rgb_img, cr, flumin, default_delta),
              fpixdy(rgb_img, cr, flumin, default_delta))
             for cr in gen_cr(num_cols, num_rows))
    gthetas_magns = ((int(ftheta(dxdy[0], dxdy[1], default_delta, default_theta)),
                      int(fmagn(dxdy[0], dxdy[1])))
                     for dxdy in gdxdy)
    for cr_thmagn in itertools.izip(gen_cr(num_cols, num_rows), gthetas_magns):
        cr, thetamagn = cr_thmagn
        theta, magn = thetamagn
        if abs(theta) <= theta_thresh and magn >= magn_thresh:
            output_img.putpixel(cr, 255)
        else:
            output_img.putpixel(cr, 0)
    return output_img
```

Generator of Dx and Dy values

Generator of magnitudes

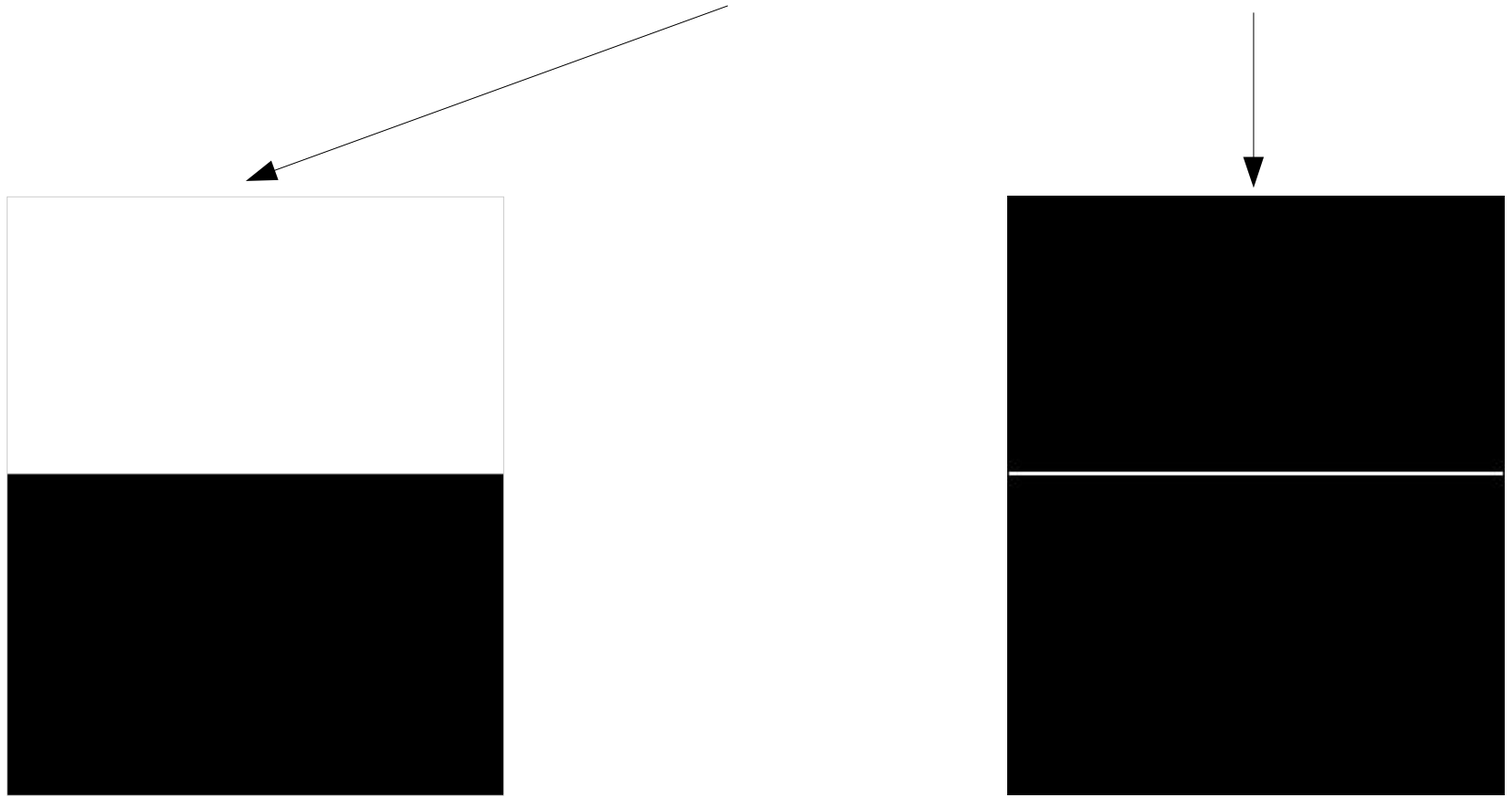
Command Line

```
if __name__ == '__main__':
    input_image = Image.open(args['input_path'])
    output_image = detect_edges(input_image)
    output_image.save(args['output_path'])
    del input_image
    del output_image
```

Py source in gd_detect_edges.py

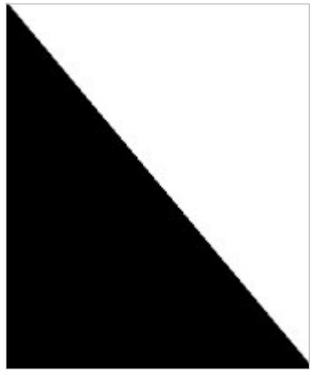
Test Run 1

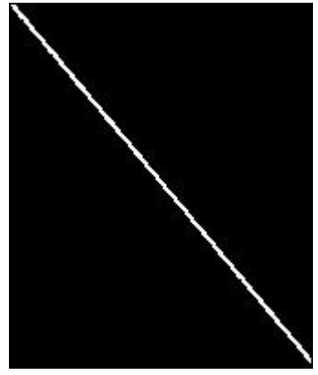
$python gd_detect_edges.py -ip EdgeImage_01.jpg -op EdgeImage_01_gd_ed.jpg



Test Run 2

$python gd_detect_edges.py -ip EdgeImage_02.jpg -op EdgeImage_02_gd_ed.jpg





Test Run 3

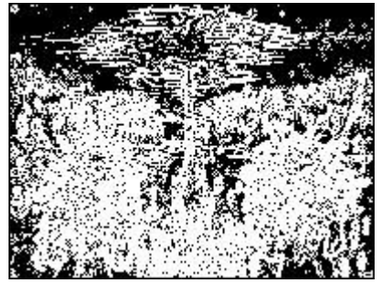
$python gd_detect_edges.py -ip EdgeImage_02.jpg -op EdgeImage_02_gd_ed.jpg

Test Run 4

$ python gd_detect_edges.py -ip BirdOrnament.jpg -op BirdOrnament_gd_ed.jpg

Test Run 5

$ python gd_detect_edges.py -ip june.jpg -op june_gd_ed.jpg



Edge Detection with OpenCV

Canny Edge Detection

- OpenCV has a number of edge detection algorithms; one of them is Canny
- Canny edge detection is an edge detection algorithm developed by John Canny in the 1980's
- Canny edge detection: 1) noise reduction through blur; 2) use a mask (aperture) to compute gradients; 3) non-maximum suppression; 4) thresholding

Non-Maximum Pixel Suppression

- Canny edge detection takes two grayscale value parameters: *minVal* and *maxVal*
- If the gradient at a pixel is above *maxVal*: this pixel is called *strong* pixel
- If the gradient at a pixel is below *minVa*l, this pixel is called a non-edge pixel
- If the gradient at a pixel is a between *minVal* and *maxVal*, then it is retained if it is connected (is in close proximity) to a strong pixel
- If a strong pixel is a local maximum but its gradient is not in the direction of the edge, it is removed
- Images are typically blurred before edge detection

Detecting Edges w/o Blurring in OpenCV

```
def detectEdgesWithoutBlur(input_path, output_path):
    image = cv2.imread(input_path)
    gray_image  = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image_edges = cv2.Canny(gray_image, 100, 200, apertureSize=3, L2gradient=True)
    cv2.imshow('Input', image)
    cv2.imshow('Gray Image', gray_image)
    cv2.imshow('Edges', image_edges)
    cv2.waitKey(0)
    cv2.imwrite(args['output_path'], image_edges)
    del gray_image
    del image_edges
```

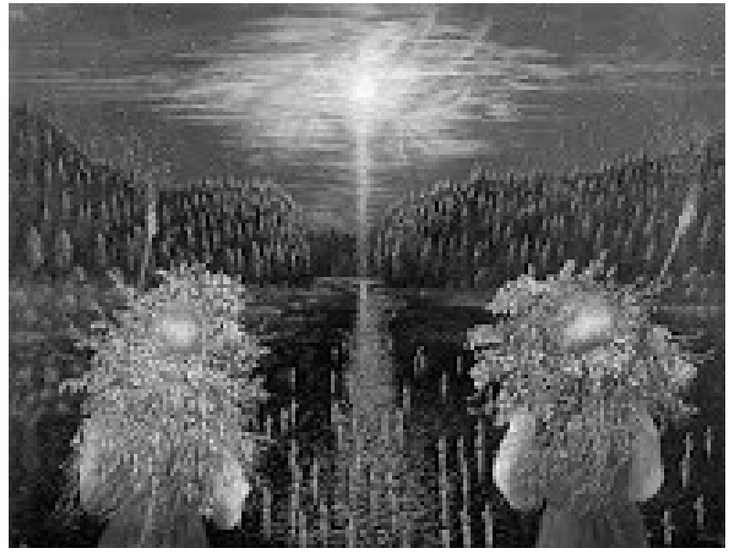
Py source in cv_detect_edges.py

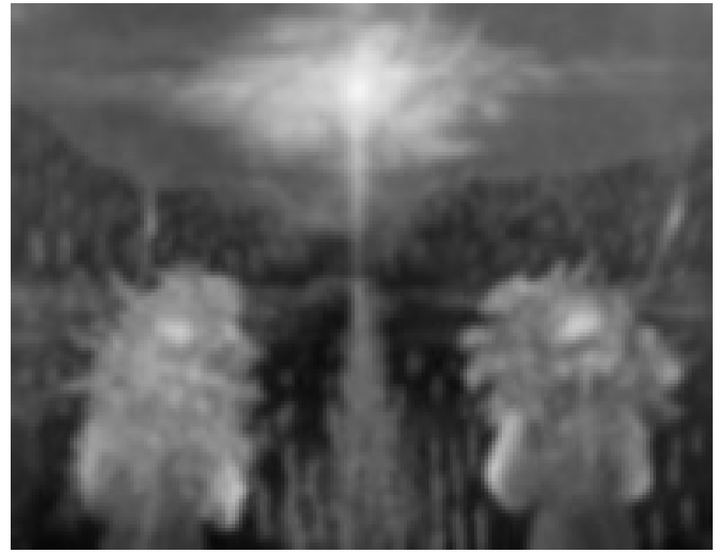
Detecting Edges w/ Blurring in OpenCV

```
def detectEdgesWithBlur(input_path, output_path, gauss_blur_mask):
    image = cv2.imread(input_path)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred_image = cv2.GaussianBlur(gray_image, gauss_blur_mask, 0)
    image_edges = cv2.Canny(blurred_image, 100, 200, apertureSize=3, L2gradient=True)
   ## rest of code to show, save, and delete images is not shown
```

Py source in cv_detect_edges.py

Grayscaling & Blurring





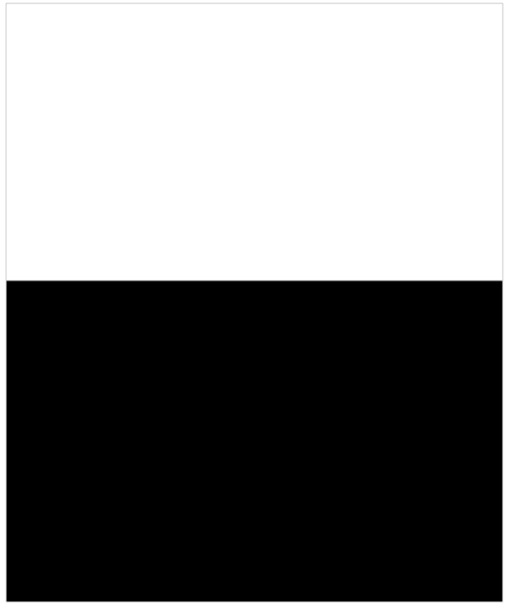
Picture is grayscaled and blurred

Detected Edges

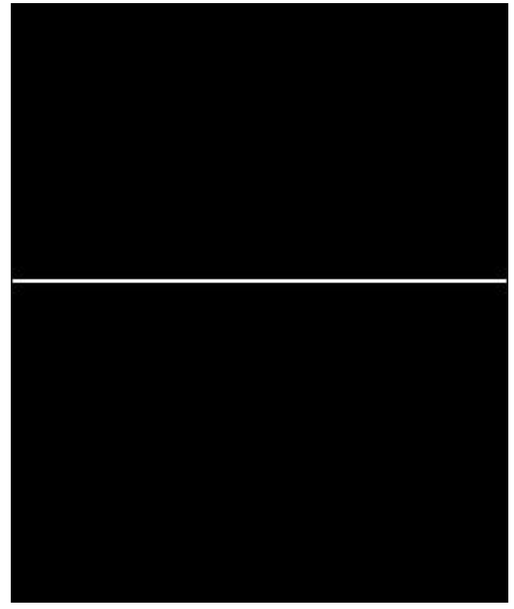
Visual Comparison of Edge Detection Results

GD vs. OpenCV

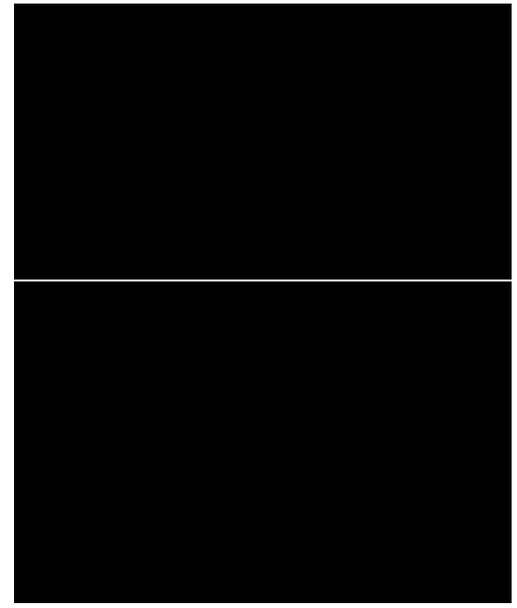
Original



GD

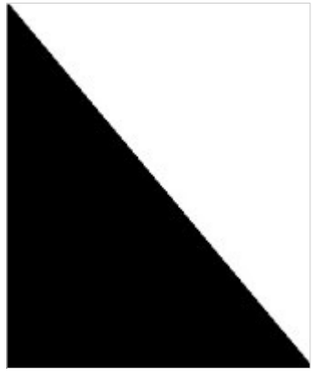


Canny

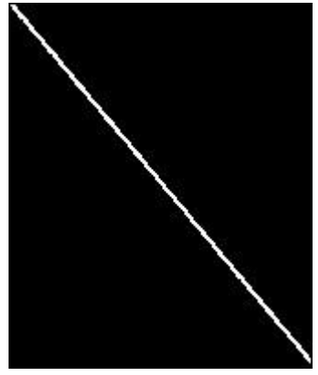


GD vs. OpenCV

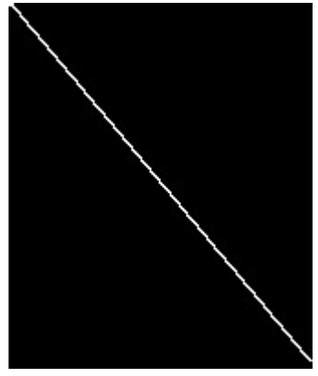
Original



GD

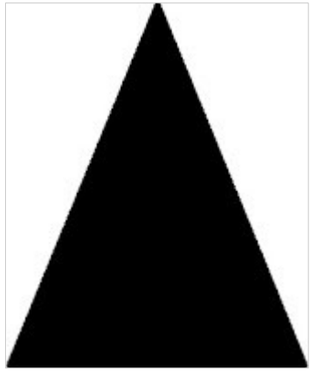


Canny

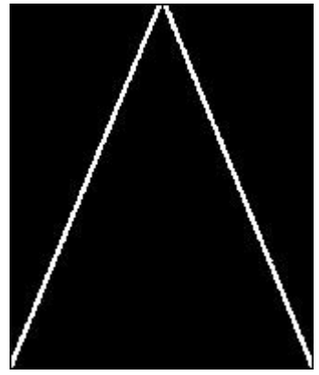


GD vs. OpenCV

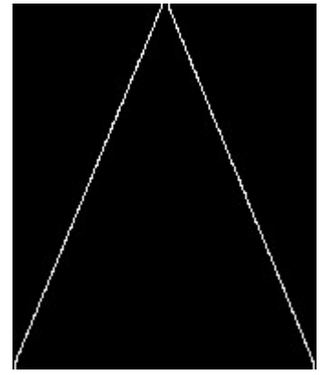
Original



GD



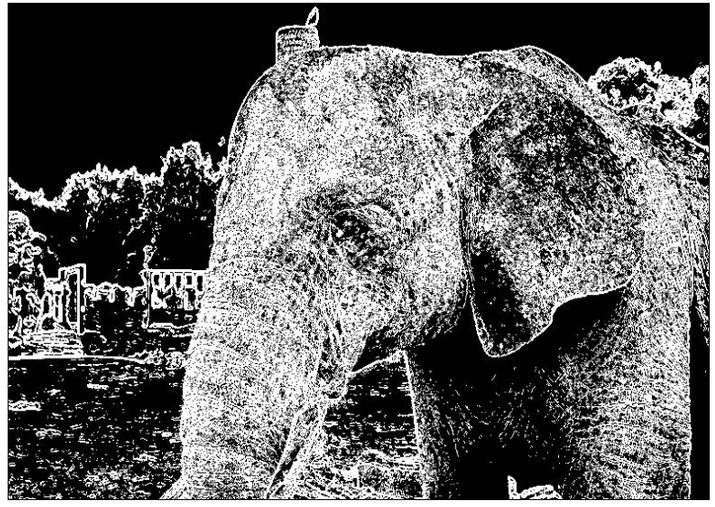
Canny



GD vs. OpenCV

Original

GD

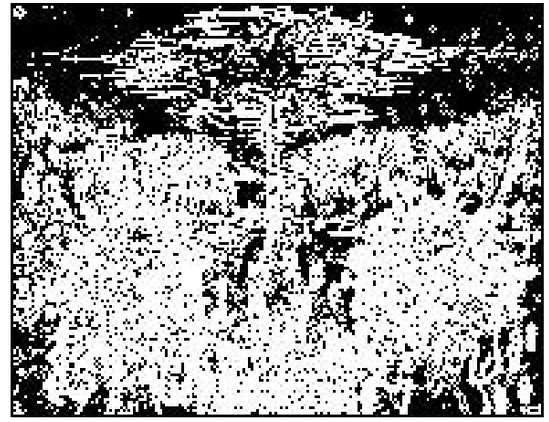


Canny

GD vs. OpenCV

Original

GD

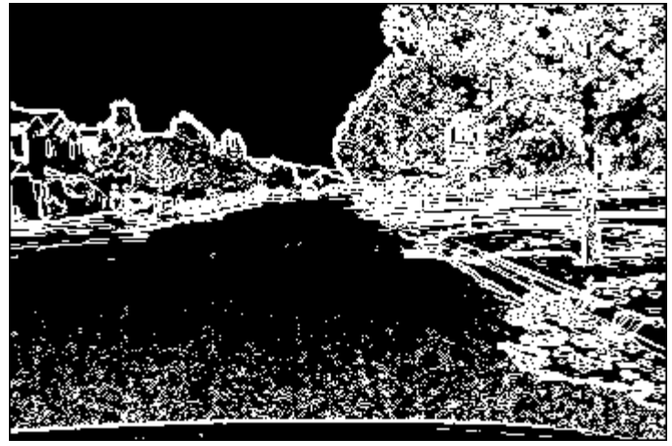


Canny

GD vs. OpenCV

Original

GD



Canny

GD vs. OpenCV

Original

GD

Canny

GD vs. OpenCV

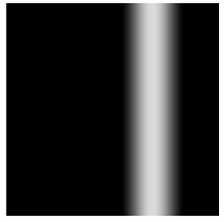
Original

GD

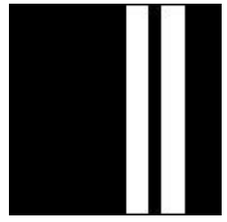
Canny

GD vs. OpenCV

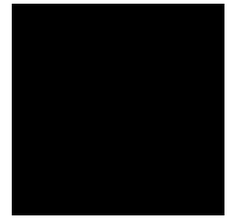
Original



GD



Canny



Some Conclusions

- On simple images, the simple gradient algorithm detects edges as well as the OpenCV Canny algorithm
- On more complex images, the simple gradient algorithm detects edges more crudely than the OpenCV Canny algorithm
- Blurring does make a difference if you want fewer edges detected
- Canny with blurring may have trouble with wide blurry edges

References

- https://en.wikipedia.org/wiki/Edge_detection
- http://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html