Generating Mosaics: A recreation of the “Novel Artificial Mosaic Generation Technique

By - Aishwarya Pani
and Tabitha Roemish

BE BOUNDLESS

Introduction

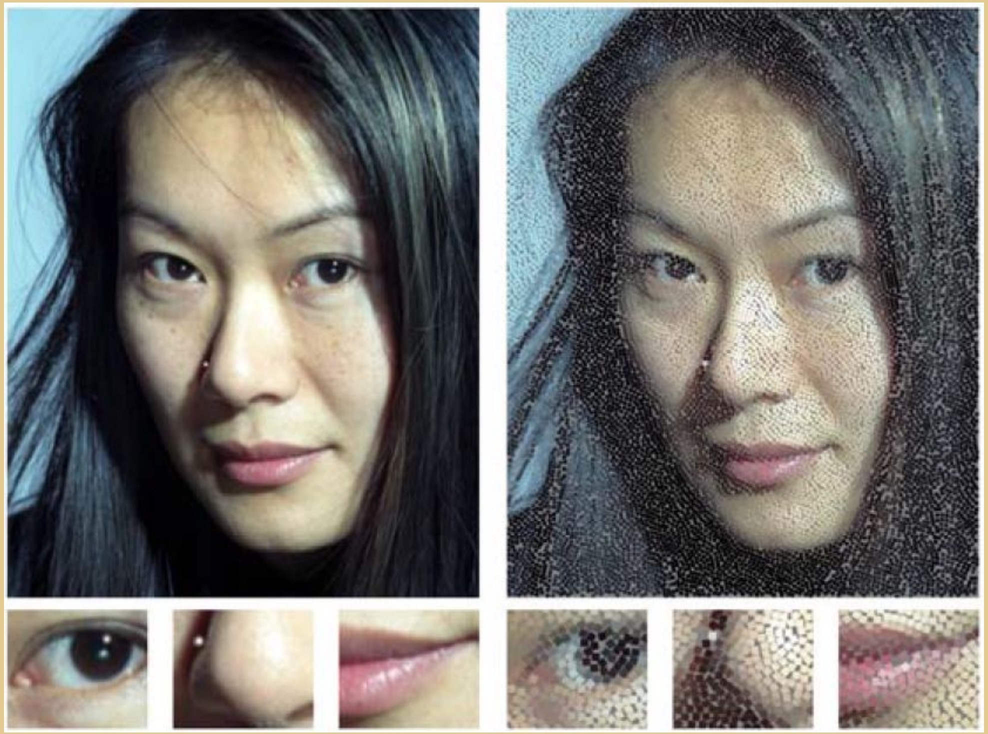
Our project takes in images and turns them into mosaics.

We chose the research paper: “A Novel Artificial Mosaic Generation Technique Driven by Local Gradient Analysis”. The paper provides pseudo code that we will implement in C++ using OpenCV.

The pseudo code uses several different preprocessing methods to arrive at the final mosaic and our code closely follows those proposed methods.

Images pulled from research paper: Battiato, Sebastiano & Blasi, Gianpiero & Gallo, Giovanni & Guarnera, Giuseppe & Puglisi, Giovanni. (2008). A Novel Artificial Mosaic Generation Technique Driven by Local Gradient Analysis. 5102. 76-85. 10.1007/978-3-540-69387-1_9.

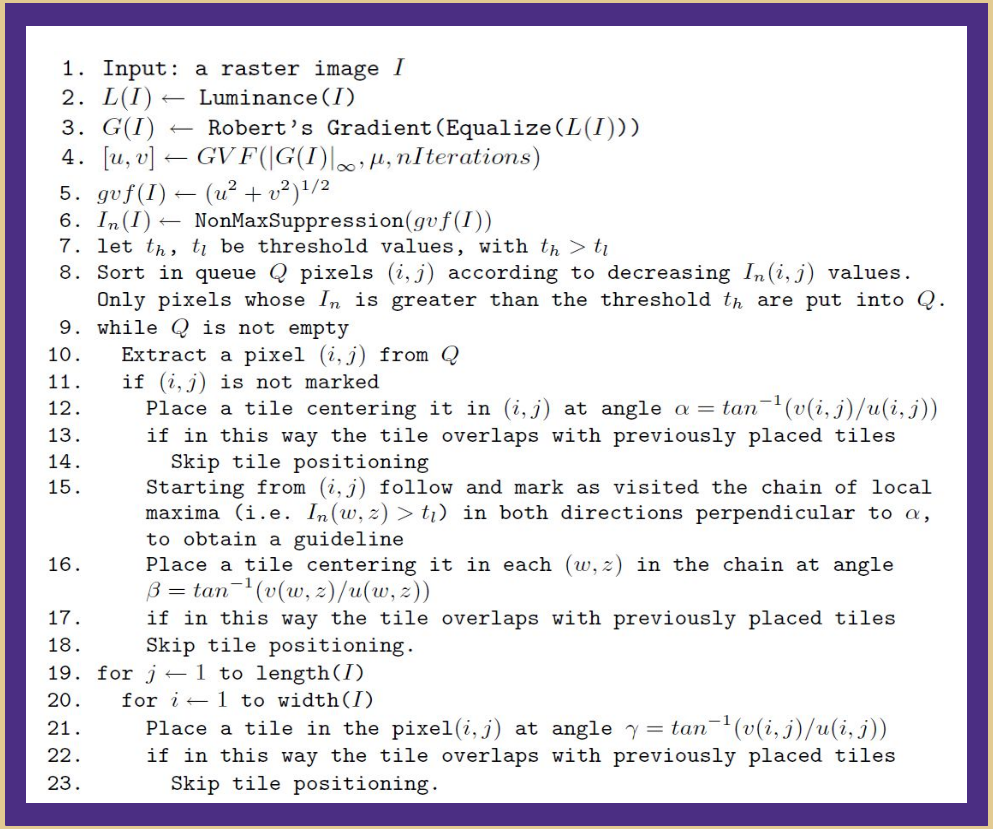
Example



Input Image

Output Image

Pseudocode



```
1.  Input: a raster image $I$
2.  $L(I) \leftarrow$ Luminance($I$)
3.  $G(I) \leftarrow$ Robert's Gradient(Equalize($L(I)$))
4.  $[u, v] \leftarrow GVF(|G(I)|_\infty, \mu, nIterations)$
5.  $gvf(I) \leftarrow (u^2 + v^2)^{1/2}$
6.  $I_n(I) \leftarrow$ NonMaxSuppression($gvf(I)$)
7.  let $t_h$, $t_l$ be threshold values, with $t_h > t_l$
8.  Sort in queue $Q$ pixels $(i, j)$ according to decreasing $I_n(i, j)$ values.
    Only pixels whose $I_n$ is greater than the threshold $t_h$ are put into $Q$.
9.  while $Q$ is not empty
10.   Extract a pixel $(i, j)$ from $Q$
11.   if $(i, j)$ is not marked
12.     Place a tile centering it in $(i, j)$ at angle $\alpha = tan^{-1}(v(i, j)/u(i, j))$
13.     if in this way the tile overlaps with previously placed tiles
14.       Skip tile positioning
15.     Starting from $(i, j)$ follow and mark as visited the chain of local
        maxima (i.e. $I_n(w, z) > t_l$) in both directions perpendicular to $\alpha$,
        to obtain a guideline
16.     Place a tile centering it in each $(w, z)$ in the chain at angle
        $\beta = tan^{-1}(v(w, z)/u(w, z))$
17.     if in this way the tile overlaps with previously placed tiles
18.     Skip tile positioning.
19. for $j \leftarrow 1$ to length($I$)
20.   for $i \leftarrow 1$ to width($I$)
21.     Place a tile in the pixel$(i, j)$ at angle $\gamma = tan^{-1}(v(i, j)/u(i, j))$
22.     if in this way the tile overlaps with previously placed tiles
23.       Skip tile positioning.
```

Goal: Turn each major step in pseudo code into a method

Research Design

Methods -

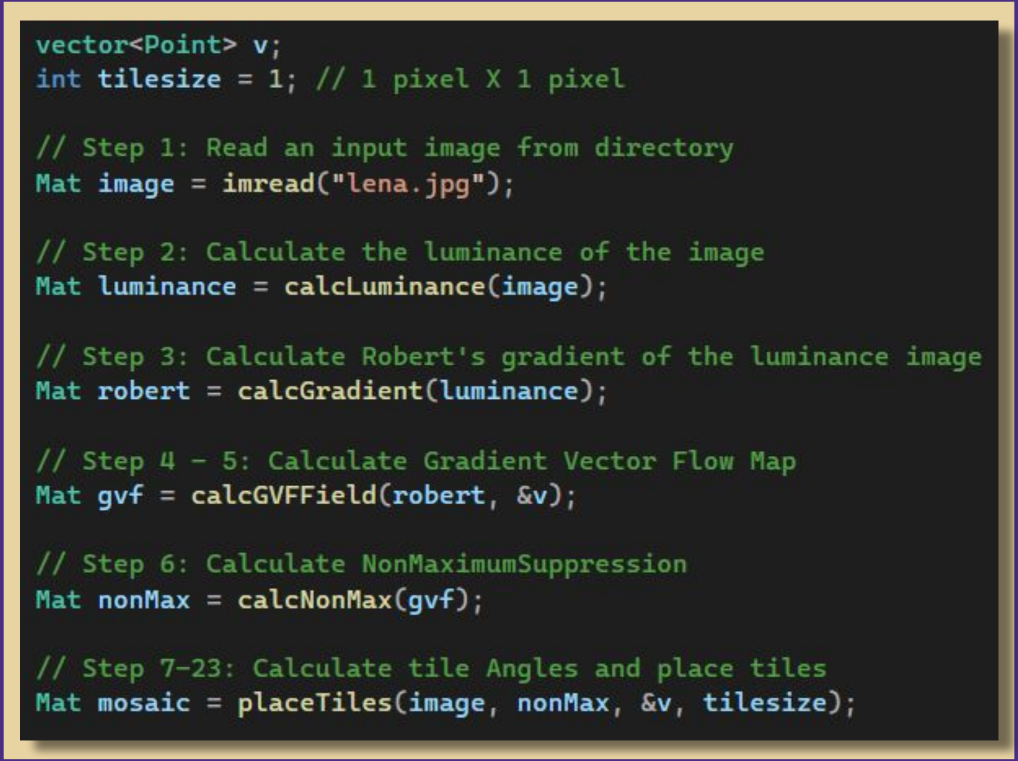
At this point we have 5 methods that feed into each other and will output the final mosaic-ized image.

OpenCV Data Structures Used:

- Mat
- Point
- RotatedRect
- Rect

Libraries used:

- CV
- STD
- imgproc



```
vector<Point> v;
int tilesize = 1; // 1 pixel X 1 pixel

// Step 1: Read an input image from directory
Mat image = imread("lena.jpg");

// Step 2: Calculate the luminance of the image
Mat luminance = calcLuminance(image);

// Step 3: Calculate Robert's gradient of the luminance image
Mat robert = calcGradient(luminance);

// Step 4 - 5: Calculate Gradient Vector Flow Map
Mat gvf = calcGVFField(robert, &v);

// Step 6: Calculate NonMaximumSuppression
Mat nonMax = calcNonMax(gvf);

// Step 7-23: Calculate tile Angles and place tiles
Mat mosaic = placeTiles(image, nonMax, &v, tilesize);
```

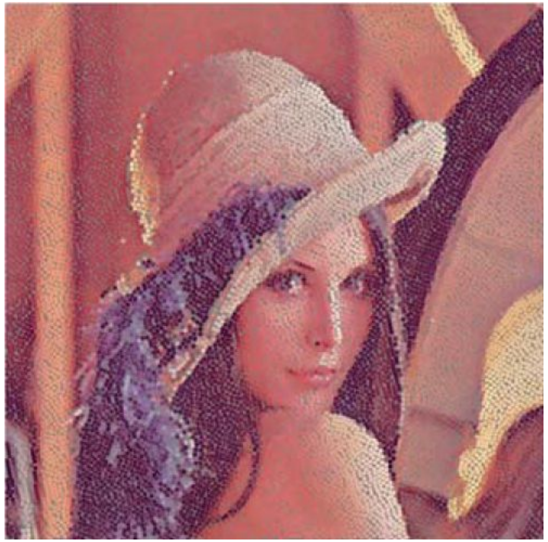
UNIVERSITY *of* WASHINGTON

Images pulled from research paper: Battiato, Sebastiano & Blasi, Gianpiero & Gallo, Giovanni & Guarnera, Giuseppe & Puglisi, Giovanni. (2008). A Novel Artificial Mosaic Generation Technique Driven by Local Gradient Analysis. 5102. 76-85. 10.1007/978-3-540-69387-1_9.

Research Design

Metrics - Initial metric will be a visual comparison

Original Lena Image


Goal Image

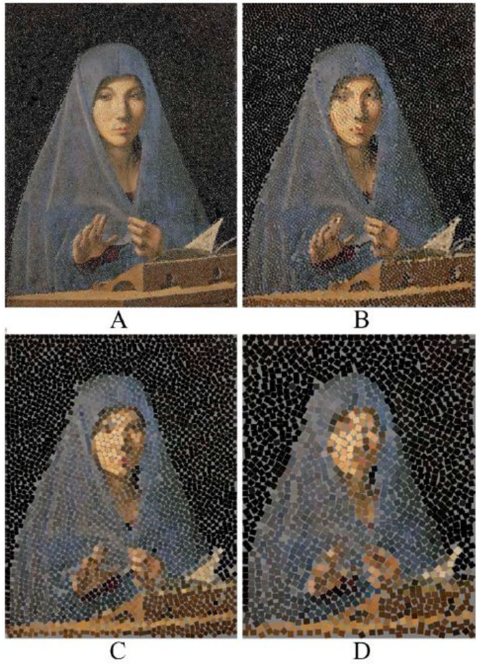


Fig. 4. Mosaics generated with increasing tiles size A (3x3), B (6x6), C (10x10), D (14x14).

Another example for comparison

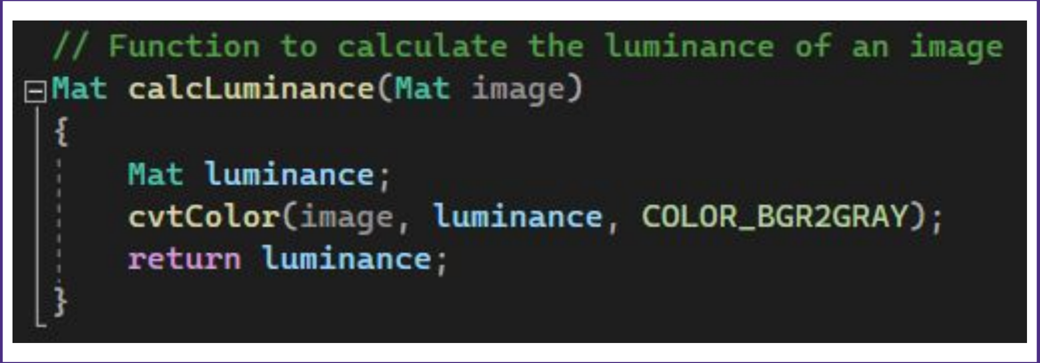
UNIVERSITY *of* WASHINGTON

Methods and Results

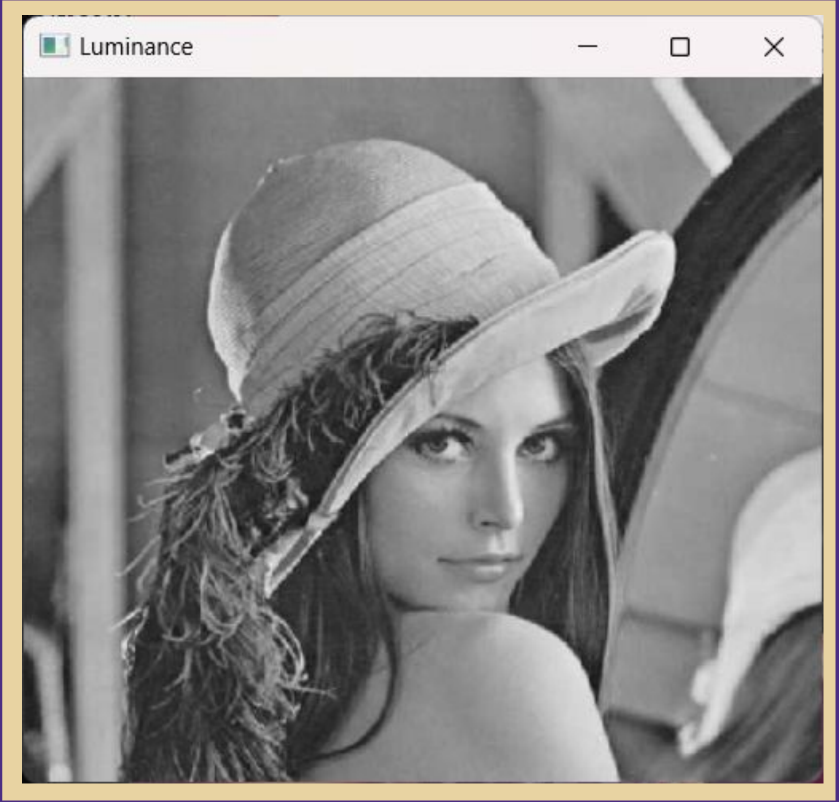
- Luminance
- Robert’s Cross
- GVF (Gradient Vector Flow)
- Non-Max Suppression
- Placing the Tiles

Luminance

Getting the luminance of an image returns the level of brightness for each pixel without the color component so the result is a greyscale image.



```
// Function to calculate the luminance of an image
Mat calcLuminance(Mat image)
{
    Mat luminance;
    cvtColor(image, luminance, COLOR_BGR2GRAY);
    return luminance;
}
```



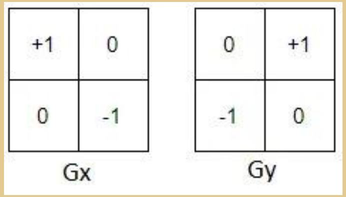
UNIVERSITY *of* WASHINGTON

Images pulled for reference:
https://a-gorkemunuvar.medium.com/robets-edge-detection-and-applyig-in-matlab-479981144a12

Robert Cross Operator

> Used for edge detection.

> We perform convolution of the input image with a 2*2 kernels.



+1	0
0	-1

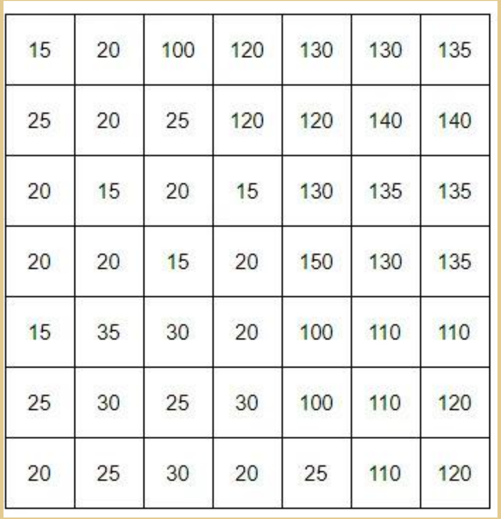
Gx

0	+1
-1	0

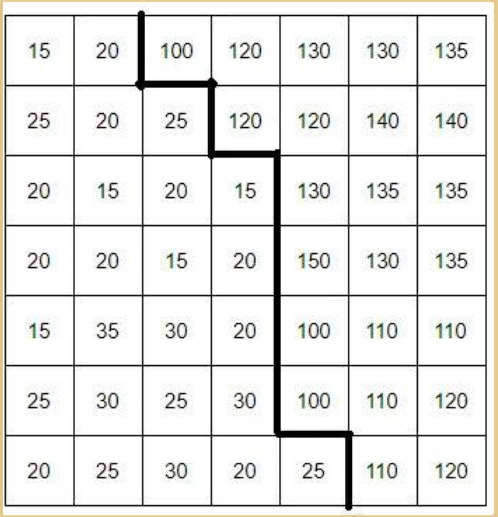
Gy

> After the image is convolved, gradient magnitude is calculated.

$$G(x, y) = \sqrt{G_x^2 + G_y^2}.$$

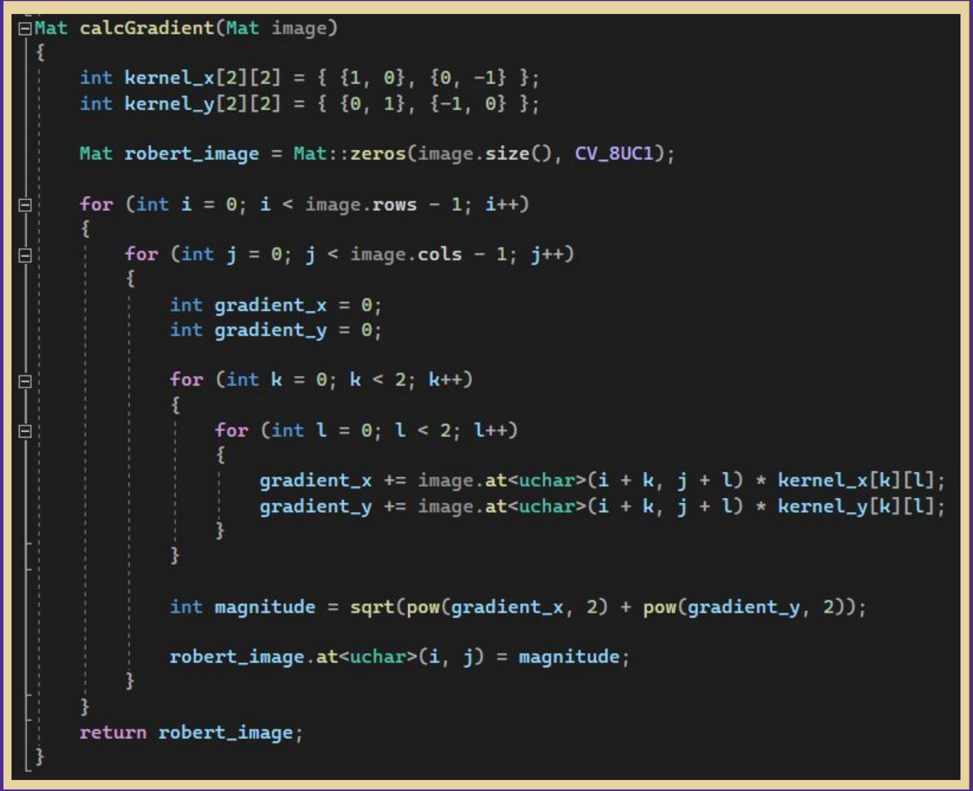


15	20	100	120	130	130	135
25	20	25	120	120	140	140
20	15	20	15	130	135	135
20	20	15	20	150	130	135
15	35	30	20	100	110	110
25	30	25	30	100	110	120
20	25	30	20	25	110	120



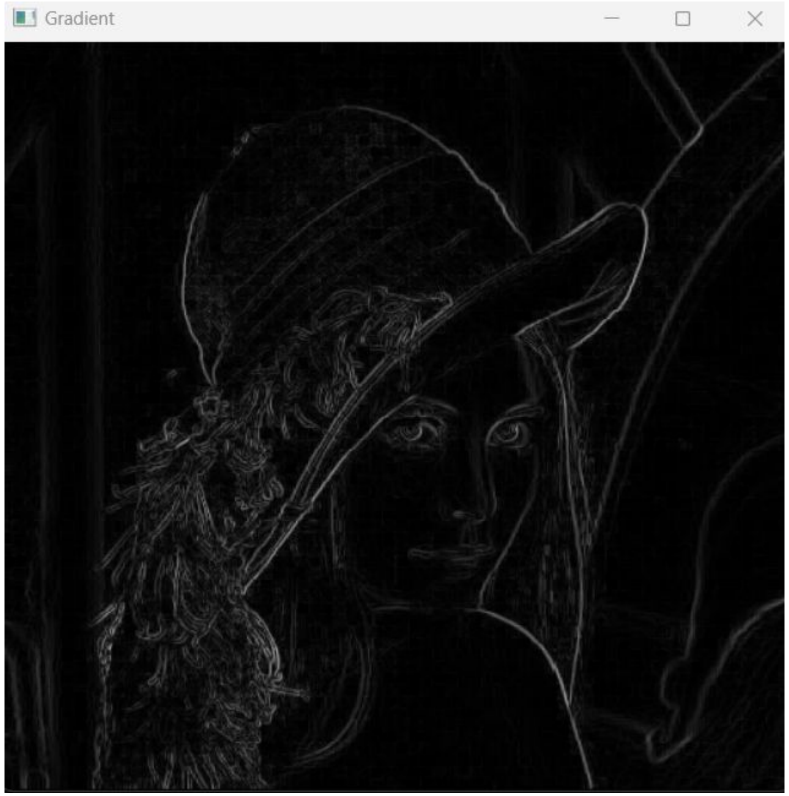
15	20	100	120	130	130	135
25	20	25	120	120	140	140
20	15	20	15	130	135	135
20	20	15	20	150	130	135
15	35	30	20	100	110	110
25	30	25	30	100	110	120
20	25	30	20	25	110	120

Robert Cross Code



```
Mat calcGradient(Mat image)
{
    int kernel_x[2][2] = { {1, 0}, {0, -1} };
    int kernel_y[2][2] = { {0, 1}, {-1, 0} };

    Mat robert_image = Mat::zeros(image.size(), CV_8UC1);

    for (int i = 0; i < image.rows - 1; i++)
    {
        for (int j = 0; j < image.cols - 1; j++)
        {
            int gradient_x = 0;
            int gradient_y = 0;

            for (int k = 0; k < 2; k++)
            {
                for (int l = 0; l < 2; l++)
                {
                    gradient_x += image.at<uchar>(i + k, j + l) * kernel_x[k][l];
                    gradient_y += image.at<uchar>(i + k, j + l) * kernel_y[k][l];
                }
            }

            int magnitude = sqrt(pow(gradient_x, 2) + pow(gradient_y, 2));

            robert_image.at<uchar>(i, j) = magnitude;
        }
    }
    return robert_image;
}
```

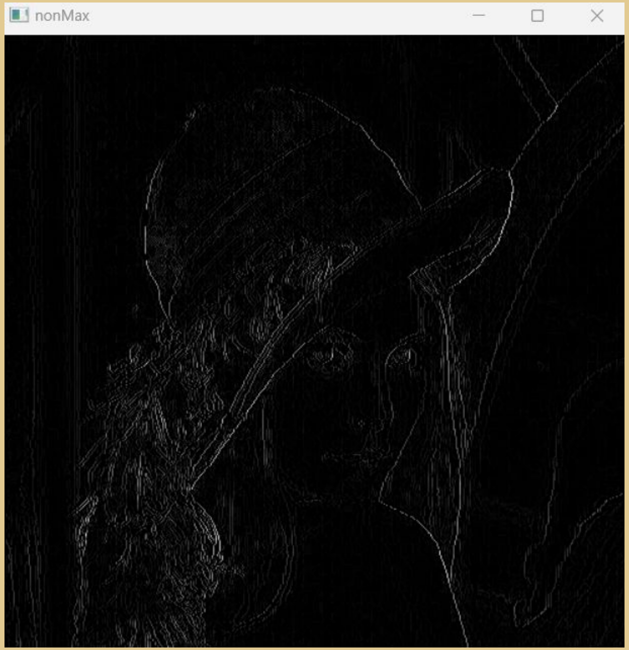
Robert Cross Results

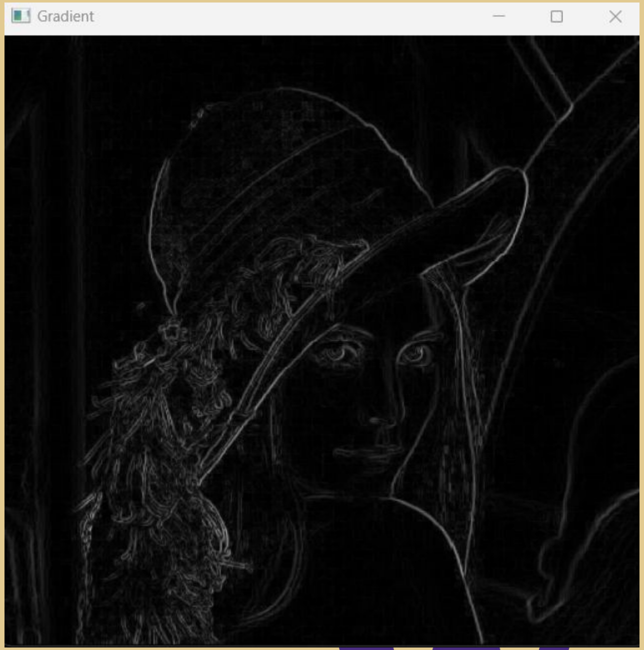


UNIVERSITY *of* WASHINGTON

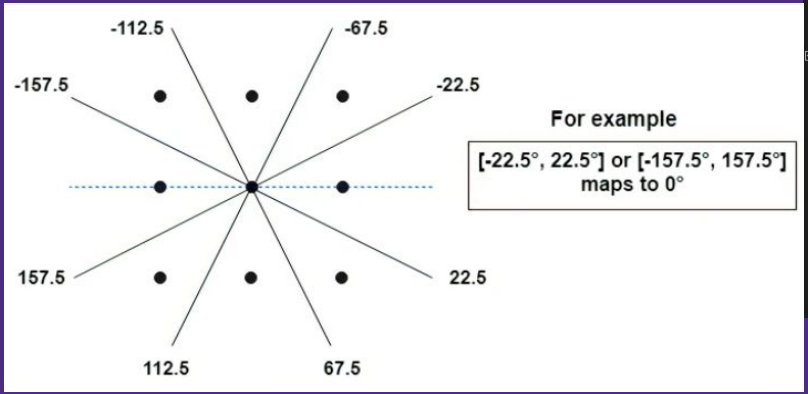
Non-Max Suppression

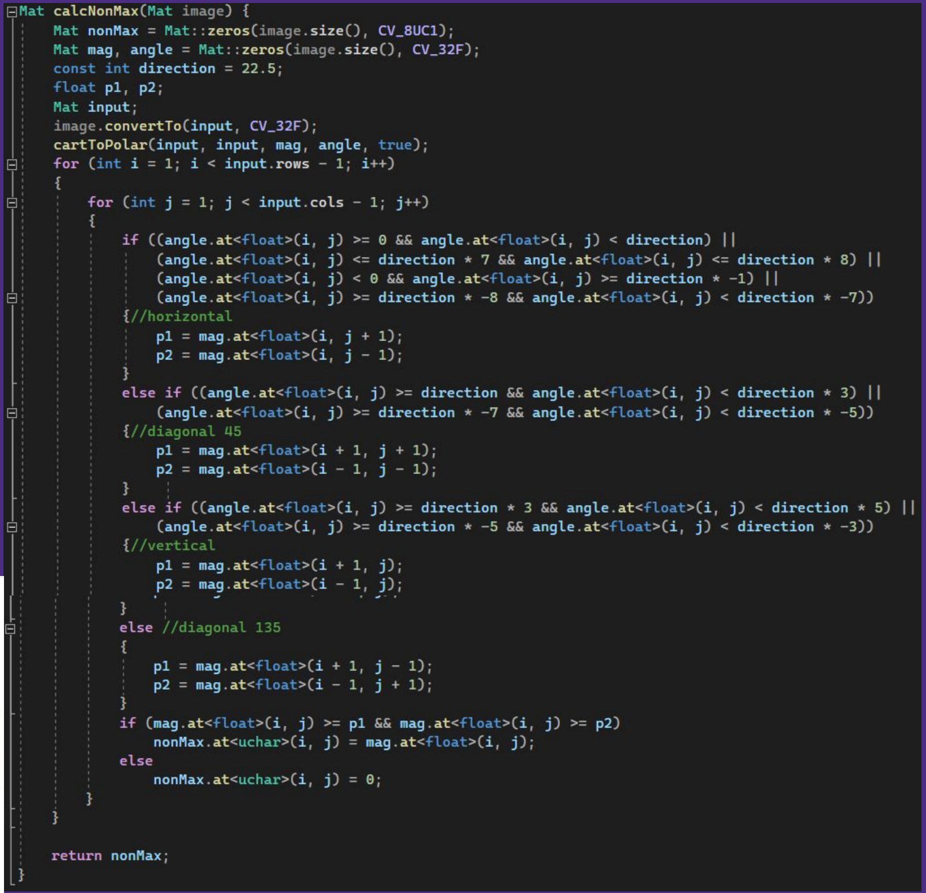
- Selects a single point out of many overlapping points by choosing the one with the highest gradient magnitude.





Non-Max Suppression Code





```
Mat calcNonMax(Mat image) {
    Mat nonMax = Mat::zeros(image.size(), CV_8UC1);
    Mat mag, angle = Mat::zeros(image.size(), CV_32F);
    const int direction = 22.5;
    float p1, p2;
    Mat input;
    image.convertTo(input, CV_32F);
    cartToPolar(input, input, mag, angle, true);
    for (int i = 1; i < input.rows - 1; i++)
    {
        for (int j = 1; j < input.cols - 1; j++)
        {
            if ((angle.at<float>(i, j) >= 0 && angle.at<float>(i, j) < direction) ||
                (angle.at<float>(i, j) <= direction * 7 && angle.at<float>(i, j) <= direction * 8) ||
                (angle.at<float>(i, j) < 0 && angle.at<float>(i, j) >= direction * -1) ||
                (angle.at<float>(i, j) >= direction * -8 && angle.at<float>(i, j) < direction * -7))
            {//horizontal
                p1 = mag.at<float>(i, j + 1);
                p2 = mag.at<float>(i, j - 1);
            }
            else if ((angle.at<float>(i, j) >= direction && angle.at<float>(i, j) < direction * 3) ||
                (angle.at<float>(i, j) >= direction * -7 && angle.at<float>(i, j) < direction * -5))
            {//diagonal 45
                p1 = mag.at<float>(i + 1, j + 1);
                p2 = mag.at<float>(i - 1, j - 1);
            }
            else if ((angle.at<float>(i, j) >= direction * 3 && angle.at<float>(i, j) < direction * 5) ||
                (angle.at<float>(i, j) >= direction * -5 && angle.at<float>(i, j) < direction * -3))
            {//vertical
                p1 = mag.at<float>(i + 1, j);
                p2 = mag.at<float>(i - 1, j);
            }
            else //diagonal 135
            {
                p1 = mag.at<float>(i + 1, j - 1);
                p2 = mag.at<float>(i - 1, j + 1);
            }
            if (mag.at<float>(i, j) >= p1 && mag.at<float>(i, j) >= p2)
                nonMax.at<uchar>(i, j) = mag.at<float>(i, j);
            else
                nonMax.at<uchar>(i, j) = 0;
        }
    }

    return nonMax;
}
```

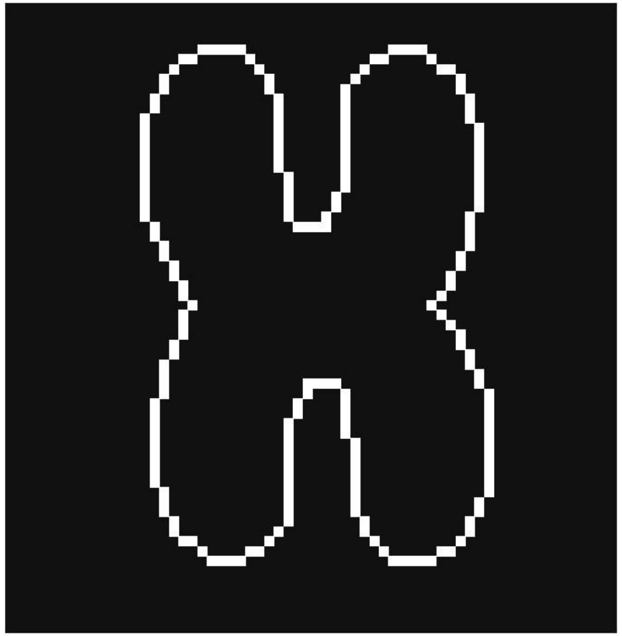
UNIVERSITY *of* WASHINGTON

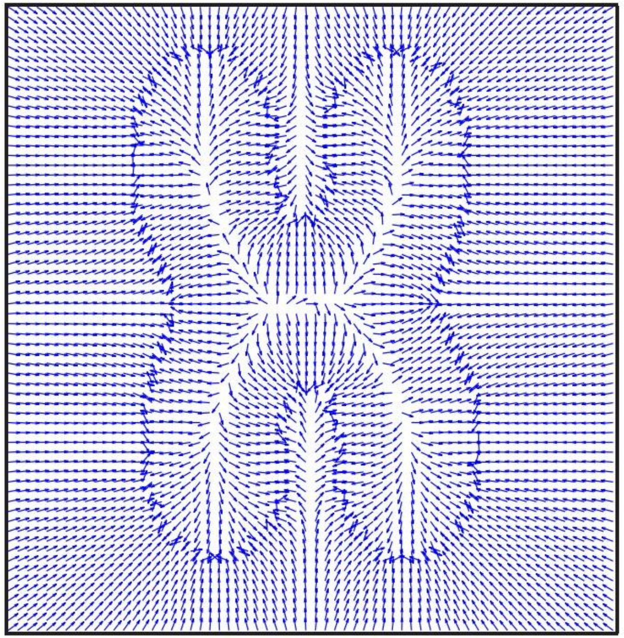
Image from : non max suppression | TheAILearner

Image reference taken from:
https://www.semanticscholar.org/paper/Gradient-Vector-Flow-Snakes-Cartas-Ayala/f90428a7ff2c51dc187dc9d39931b392287d27b1

GVF

Gradient vector flow is a process which is used to yield a new vector field on the images that points to object edges from a distance throughout the image.

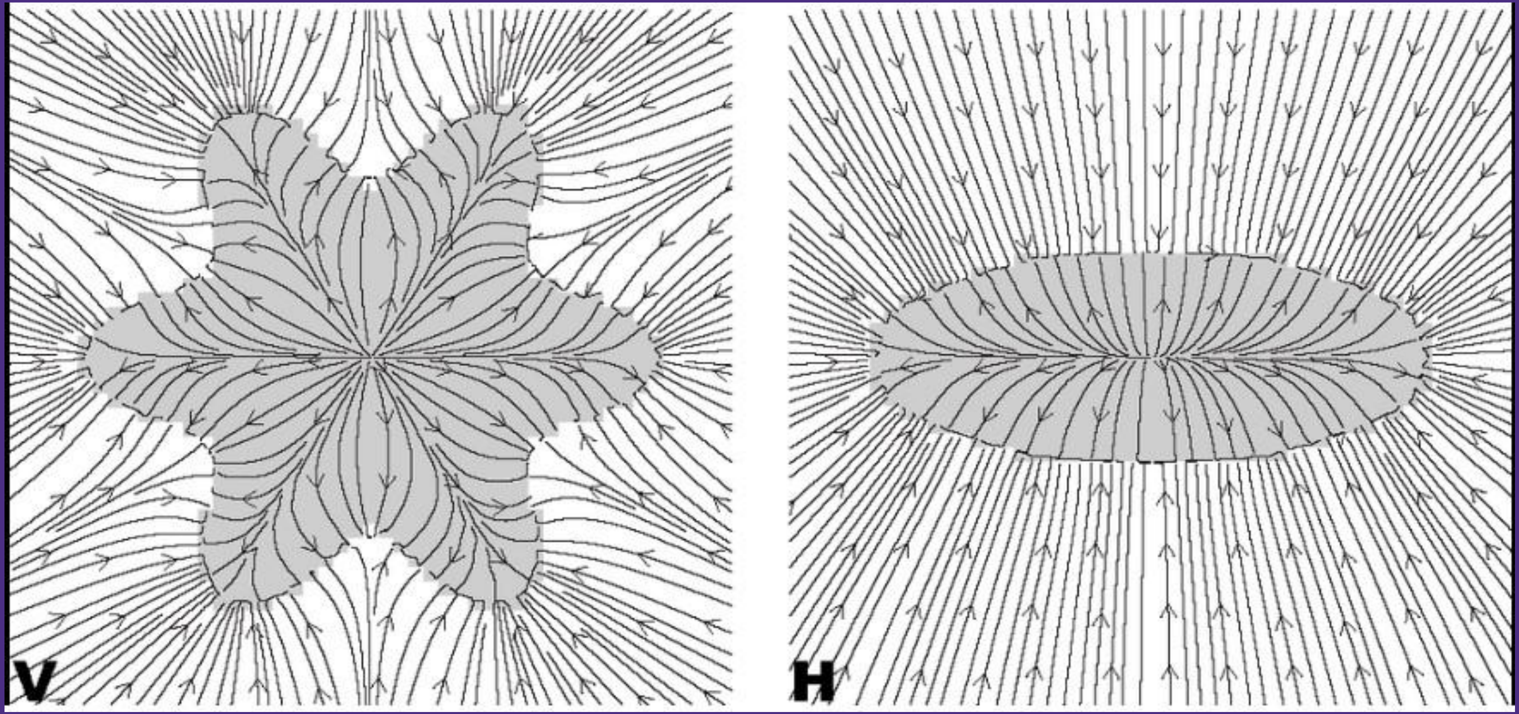

(a) Edge map


(b) Normalized GVF field.

UNIVERSITY *of* WASHINGTON

Image reference taken from: https://en.wikipedia.org/wiki/Gradient_vector_flow

GVF cont.

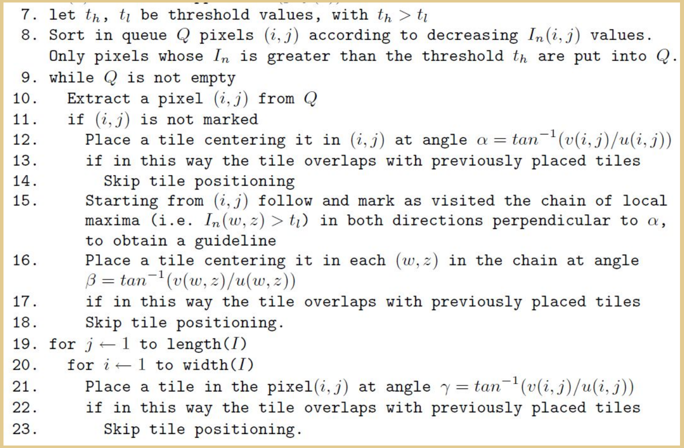


GVF Steps

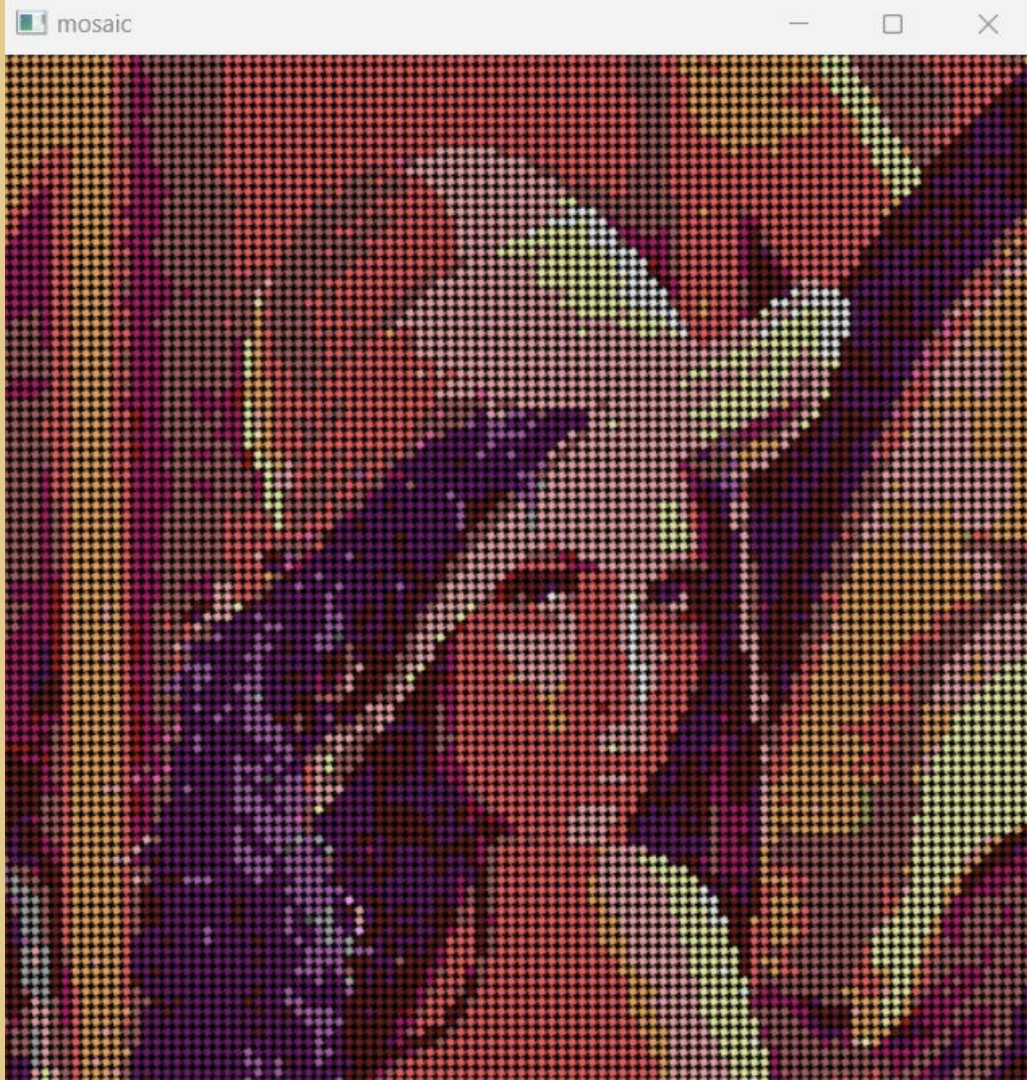
1. Normalized the edge map
2. Computed the edge map gradient
3. Computed parameters and initialized arrays
4. Solved GVF iteratively
5. Normalized the GVF
6. Released the memory

Place Tiles

- Place tiles based on largest gradient magnitude first at angle specified by gradient vector flow.
- Fill in tiles perpendicular to angle of placed tiles
- Fill in remaining tiles



```
 7. let $t_h$, $t_l$ be threshold values, with $t_h > t_l$
 8. Sort in queue $Q$ pixels $(i,j)$ according to decreasing $I_n(i,j)$ values.
    Only pixels whose $I_n$ is greater than the threshold $t_h$ are put into $Q$.
 9. while $Q$ is not empty
10.   Extract a pixel $(i,j)$ from $Q$
11.   if $(i,j)$ is not marked
12.     Place a tile centering it in $(i,j)$ at angle $\alpha = tan^{-1}(v(i,j)/u(i,j))$
13.     if in this way the tile overlaps with previously placed tiles
14.       Skip tile positioning
15.     Starting from $(i,j)$ follow and mark as visited the chain of local
        maxima (i.e. $I_n(w,z) > t_l$) in both directions perpendicular to $\alpha$,
        to obtain a guideline
16.     Place a tile centering it in each $(w,z)$ in the chain at angle
        $\beta = tan^{-1}(v(w,z)/u(w,z))$
17.     if in this way the tile overlaps with previously placed tiles
18.     Skip tile positioning.
19. for $j \leftarrow 1$ to length($I$)
20.   for $i \leftarrow 1$ to width($I$)
21.     Place a tile in the pixel$(i,j)$ at angle $\gamma = tan^{-1}(v(i,j)/u(i,j))$
22.     if in this way the tile overlaps with previously placed tiles
23.       Skip tile positioning.
```



Completed Tasks

1. Luminance function
2. Robert Gradient function
3. Non-Max Suppression function
4. Generated tiles from input image

Tasks To Complete

1. Gradient Vector Flow - Function to perform Gradient Vector Flow on the magnitude of the gradient.
2. Place Tiles - Function to sort pixels based on GVF intensity and threshold. Then iterate through the sorted pixels and place tiles, avoiding overlap with existing tiles.
3. Prepare the final report.
4. Possibly a implement a Trackbar.

UNIVERSITY *of* WASHINGTON

Demo

Time for the Demo!

Questions?

UNIVERSITY *of* WASHINGTON

References

1. Images pulled from research paper: Battiato, Sebastiano & Blasi, Gianpiero & Gallo, Giovanni & Guarnera, Giuseppe & Puglisi, Giovanni. (2008). A Novel Artificial Mosaic Generation Technique Driven by Local Gradient Analysis. 5102. 76-85. 10.1007/978-3-540-69387-1_9.

2. https://en.wikipedia.org/wiki/Gradient_vector_flow