Delaunay Diagram Representations For Use in Image Near-Duplicate Detection

Senior Project submitted to
The Division of Science, Mathematics & Computing
Of Bard College

By
Adina Raluca Stoica

Annandale-on-Hudson, New York
May 2011

Table of Contents

ACKNOWLEDGEMENTS

The Senior Project represents one of the most important steps a Bard student has to take. It is the culmination of four years, in which one has experienced learning at its finest. When I first came to Bard from Romania, I didn't know what kind of road I was embarking on, but I am ever so glad I took the chance. The people I have met and interacted with here have changed my life for the better, and the classes I have taken, as well as the out-of-school opportunities I have been able to experience are unmatched in my home country. So, for starters, I would like to thank Bard, for accepting me and offering me a place in its student body, as well as the Division of Science, Mathematics and Computing for making it possible, financially, for me to attend.

From the amazing group of people that I have met here, I would first and foremost like to warmly thank my Senior Project adviser, Robert McGrail, for the wealth of support he has offered me not only through my Senior Project, but throughout the entire four years here—THANK YOU! I would also like to thank the other members of the Computer Science Department – Sven Anderson, Rebecca Thomas, Keith O'Hara – for their support, friendliness and information they have given me, during these 4 years, as well as thank other professors that have been involved in my Senior Project, especially Mary Krembs, for the suggestions given after my midway presentation.

I am now close to the end of my period here, and I know that I will miss Bard. My future guides me to Saint Louis, where I will be pursuing my Ph.D. in Computer Science at Washington University. Bard has helped me grow up, and has taught me to be an independent, free-willed adult: for this I will forever be thankful! And, to the class of 2011: Congratulations! We made it!

ABSTRACT

Given an image or an image Delaunay Triangulation diagram representation, the purpose of this project is to identify a near-duplicate in an image database. This is done by computing the corner points of the image under some default settings, using the Voronoi diagram to adjust these settings in order to get better adjusted corner points for the image, and then using the Delaunay Triangulation of the image to identify whether there are possible duplicates in the database. The project investigates Voronoi Diagrams based on a default Harris Corner Detector as an image segmentation technique for adjusting the threshold for the Harris Corner Detector, as well as Delaunay Triangulation Diagrams as a measure of similarity between images. Ultimately, the project finds image near-duplicates in a database with some likelihood.

1. INTRODUCTION

Given an image to be submitted in a database (or uploaded on the Internet) we would like to know whether that specific image (under any transformations permitted by current image-editing software) is already in the database. Moreover, we would like to be able to withhold the actual image from the duplicate detector when performing the search, and instead use a diagram representation of the image, achieved using the Delaunay Triangulation.

Every image can be characterized to different degrees by various measures, such as contrast difference, preponderancy of colors, wavelets, and entropy. There are some measures that are conserved under various image transformations such as cropping, blurring, desaturation,

rotation, scaling, linear and non-linear changes in brightness, perspective distortion, noise, changes in resolution, and filtering.

Mainly, the aim of this project is to investigate image measures, specifically Delaunay Triangulation Diagrams based on auto-adjusted Harris Corner Points, that are better preserved under some considered transformations and to develop an effective and efficient algorithm to detect whether any near-duplicates for the image already exist in the database using these measures. There are three stages in which the project is going to achieve its aim. First of all, the Harris Corner Points Detection algorithm is used, under some default parameter settings, to detect key points in the image that have an extreme likelihood to maintain their relative position within the image under various transformations (the so-called Harris corner points, see section 4 and (Harris & M., 1988)), and the Voronoi Diagram is computed using the detected points. Secondly, the Voronoi Cells thus computed are considered as an image segmentation method to allow the computer to automatically adjust the threshold used in the Harris Corner Detector – the new Harris Corner Points are computed using this new, auto-adjusted threshold. Lastly, the efficiency of Delaunay Triangulation Diagrams (which are actually the dual graphs of the Voronoi Diagrams) based on these new corner points is investigated in encoding important features of the image, so that the diagram can eventually be used in finding near duplicates – the images are probabilistically matched by comparing the features and structures of these diagrams, while the parts of the image that are outside the Delaunay Diagram are discarded.

Image matching can be done on pairs of images, loaded from a database, as well as on a diagram representation of one image, to be compared with an image in the database; both reduce to the same algorithm. Eventually the program would allow the user to load one image/Delaunay representation and automatically search a database of images. The goal is to detect the likelihood

that the loaded image/Delaunay representation is the near-duplicate of any other image in the database.

The stages the algorithm has to go through consist of first detecting the Harris Corner points for the image using a set value for the parameters (specifically the threshold) and computing the Voronoi diagram based on the thus detected corner points, then using the Voronoi cells to auto-adjust the threshold parameter for a better Harris Corner Points detection, and then computing the Delaunay Triangulation diagram and using various graph similarity measures to determine whether or not we are dealing with similar graphs.

2. WHAT IS AN IMAGE NEAR-DUPLICATE?

While image duplicates are images that are identical in all respects to one another, image near-duplicates are trickier and can be subjected to different interpretations. According to (Foo, Zobe, & Sinha, 2007), there are three groups in which near-duplicates have been categorized in research literature, based on the nature of the modifications of the original image: changes in scene (where the objects placed inside the scene change from one image to another), changes in camera (where the perspective of the camera changes between images) and changes in image (which are due to image transformations produced by image editing software).

In my project I am only concerned with the last group, which deals with images that have the same digital source, and only differ through changes applied in post-production. The fact that the images have the same source should be obvious for a human in order for the images to still be considered near-duplicates by my definition – once the similarities between the images are blurred out for the human eye, it becomes even harder for a computer to identify them (without accessing photo information).

Moreover, just like the human eye might not be able to distinguish between near-duplicate pictures that fall into the other two groups (changes in scene and changes in camera), one of the biggest problems a working algorithm might encounter is false positives falling into those groups.

3. IMAGE TRANSFORMATIONS CONSIDERED

For testing purposes, image near-duplicates were created and placed in a folder. The user has the option of creating some more near-duplicates by using the *slideshow* program. For the purpose of this project, simple, linear, transformations of the image are considered. These transformations can be grouped in different categories (of course various different grouping criteria can be chosen, such as geometric changes versus photometric changes):

- Changes in image size/ aspect ratio: cropping, scaling (geometric)
- Changes in image aspect: blurring, contrast, luminosity (photometric)
- Changes in image content: rotation, reflection, horizontal shear (geometric)

In Figures 2 to 9, the basic transformations permitted by the *slideshow* program are exemplified on the image in Figure 1. See the Annex for a picture of the interface of the program and an explanation about using it.

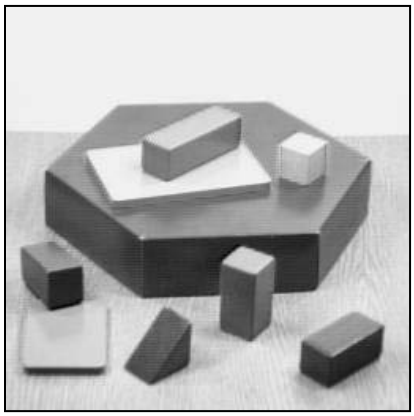


Figure 1: Original Image

Cropping is achieved by using MATLAB's function, *imcrop*, with the same starting point at the origin of the original image and the width and height scaled to 90%.

```
w=size(img,1); % get the width of the original image
h=size(img,2); % get the height of the original image
```

```
img2=imcrop(img,[0,0,h*0.9,w*0.9]); % crop image to 90%
```

Code Snippet 1: Image Cropping

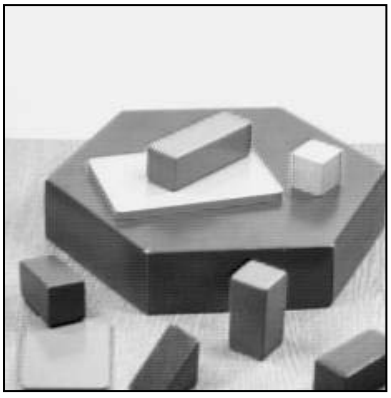


Figure 2: Cropped Image

Scaling the image (up or down) is done using MATLAB's *imresize*— the image is scaled up by 20% and down by 70%.

```
img2=imresize(img,1.2); % scale up
img2=imresize(img,0.3); % scale down
```

Code Snippet 2: Image Scaling

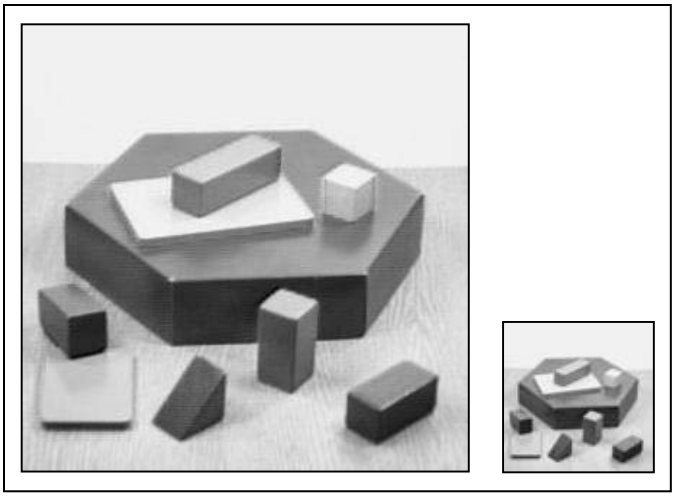


Figure 3: Scaled Images (Up and Down)

Rotating, to the right, by 90 degrees, is achieved using MATLAB's *imrotate* function, with the second argument -90, for the angle.

```
img2=imrotate(img,-90); % rotate image 90 degrees, to the right
```

Code Snippet 3: Image Rotation, 90° Right

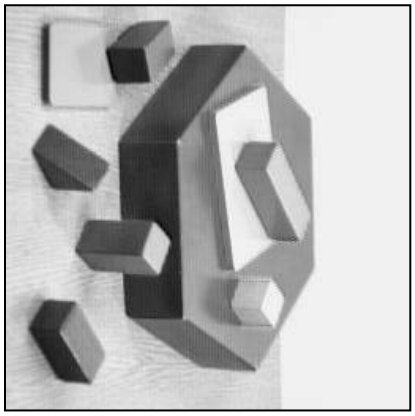


Figure 4: Image Rotated 90° Right

Blurring is done by creating a rotationally symmetric Gaussian low-pass filter of size 10 with standard deviation 10, using MATLAB's *fspecial* function, and applying it to the image.

```
h = fspecial('gaussian',10,10); % define a Gaussian filter
img2=imfilter(img,h); % apply the filter
```

Code Snippet 4: Image Blurring

Figure 5: Blurred Image

Horizontal reflection (or mirroring) is done by using on every color MATLAB's *fliplr* function. Since the function only works on grayscale images, it is applied on every color channel component of the image; then the result is unified.

```
img2 = []; %fliplr works with 2d matrices, so we have to
img2(:,:,1)=fliplr(img(:,:,1)); % flip the image
img2(:,:,2)=fliplr(img(:,:,2)); % around a horizontal axis
img2(:,:,3)=fliplr(img(:,:,3)); % component-by-component
img2=uint8(img2); % convert to uint8, just in case
```

Code Snippet 5: Image Reflection

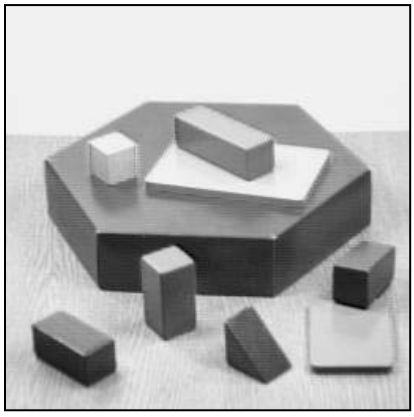


Figure 6: Horizontally Reflected Image

Auto-adjusting the **contrast** is done by equalizing the image histogram by using MATLAB's *histeq* function. Since histeq only works with grayscale, 2D, images, the function is applied on every color component of the image; then the image is unified. Without a second parameter, histeq uses the flat histogram as the default hgram, and enhances the contrast of the image by transforming the values so that the histogram of the output image approximately matches this flat histogram. The algorithm chooses the grayscale transformation T to minimize $\left|c_1\left(T(k)\right) - c_0(k)\right|$, where c_0 is the cumulative histogram of the image, and c_1 is the cumulative sum of histograms for all intensities k.

```
img2 = []; % histeq works with 2d matrices, so we have to do it
img2(:,:,1)=histeq(img(:,:,1)); % on every color component
img2(:,:,2)=histeq(img(:,:,2)); % of the RGB image
img2(:,:,3)=histeq(img(:,:,3)); % represented by a 3D matrix
img2=uint8(img2); % cast to uint8, for certainty
```

Code Snippet 6: Image Auto-Contrast

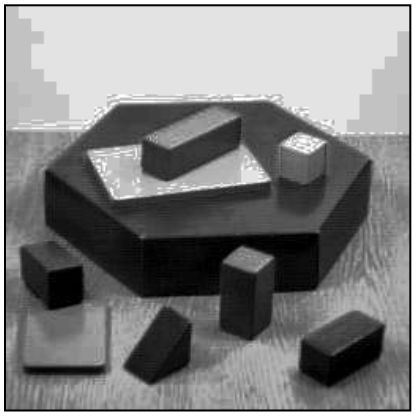


Figure 7: Auto-Contrasted Image

Adjusting the **luminosity** is done using MATLAB's *imadjust*, which takes two vectors of RGB values – an input (In) and an output vector (Out), which both have two components: a high component and a low component. Each vector has form [lowR lowG lowB; highR highG highB], and the values between the two per-color (low and high) components of the first (In) vector are mapped to the values between the same components of the second (Out) vector, while the values that are outside of the interval of the first vector get clipped – the ones lower than lowColor in In get mapped to lowColor in Out, the ones higher than highColor in In get mapped to highColor in Out. The empty vector, [], defaults to [0 0 0; 1 1 1].

```
img2=imadjust(img,[0 0 0; 0.8 0.8 0.8],[]); % brighten image
% all values between 0 0 0 and .8 .8 .8 are mapped to values between
% 0 0 0 and 1 1 1; values above .8 .8 .8 map to 1 1 1.
%....
img2=imadjust(img,[0.2 0.2 0.2; 1 1 1],[]); % darken image
```

Code Snippet 7: Image Brightness

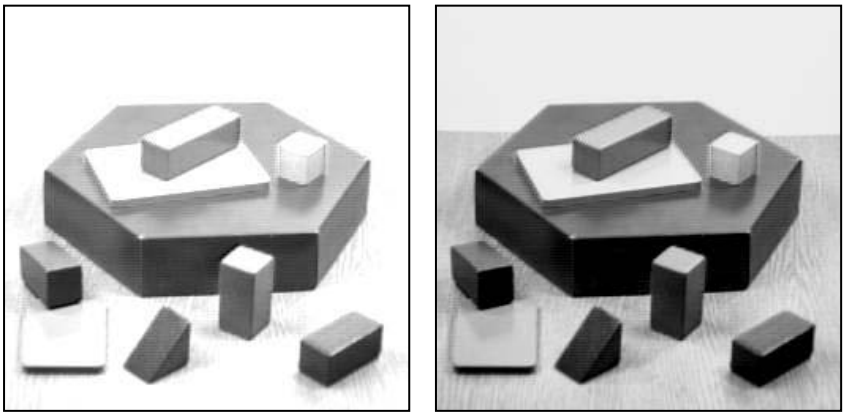


Figure 8: Lighter and Darker Images

Horizontal shear transforms the image by shearing it horizontally to the left, according to the affine transformation matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0.2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Shearing by this matrix slides the bottom horizontal side of the image by 0.2 to the right, creating a parallelogram. It is interesting how this transform can affect both the corner detection and the Delaunay similarity measure.

```
% define a horizontal shear transform
tform = maketform('affine',[1 0 0; 0.2 1 0; 0 0 1]);
img2=imtransform(img,tform); % apply the transformation
```

Code Snippet 8: Image Horizontal Shear

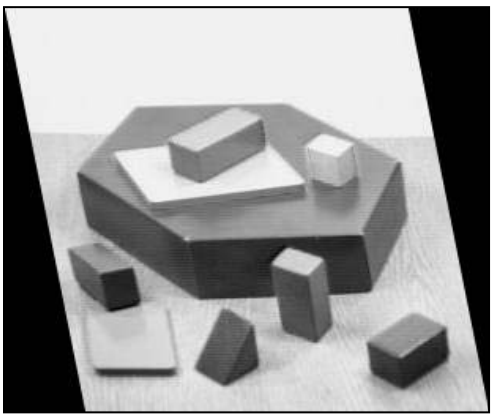


Figure 9: Horizontally Sheared Image

4. THE HARRIS CORNER POINTS DETECTOR

4.1. Introduction to the Harris Corner Detector

Moravec (Moravec H. , 1979) (Moravec, 1980) was the first who measured the average change of intensity by shifting a local window by a small amount in different directions. The Harris Corner Detector method was an improvement on Moravec's Corner Detector and was first described in a paper published in 1988. The motivation for a corner detector, as well as the algorithm and the mathematics behind it are described below. The project intends to improve the algorithm described in the paragraphs below by allowing the automatic adjustment of the parameters (especially the threshold) in order to achieve better-fitted corner points.

4.2. Motivation for the Harris Corner Detector

When using computer vision to understand the unconstrained 3D world, the viewed scenes might contain too many objects for top-down recognition techniques to work. In order to obtain an understanding of natural scenes (buildings, roads, trees, bushes, etc.), we need to track discrete image features (which neither form a continuum, like a texture, nor are edge pixels – called edgels).

In order to successfully track objects in 3D, the image features we are interested in should be both corners (feature points) – which are discrete, reliable and meaningful, but are disconnected and are therefore limited in providing higher level descriptions, such as for surfaces and objects – and edges, which provide richer information, through their connectivity. Edge connectivity and 3D locations of corners and junctions describe approximate 3D surfaces through a wire-frame structural representation and delimited image regions.

4.3. Mathematics behind the Harris Corner Detector

The Harris Corner Detector is a method of determining the nature of a point by computing the average change of intensity in the image when shifting a small local window in the image by a small amount in any direction. By shifting the window in any direction by one pixel, we can determine the nature of the point: if the region is flat, all shifts will result in only a small change; if we are on an edge, a shift along the edge (in any direction) will result in a small change and a shift perpendicular to the edge will result in a large change; if the point is a corner, all shifts (in at least one of the opposite directions) would result in a significant change, as seen in Figure 10.

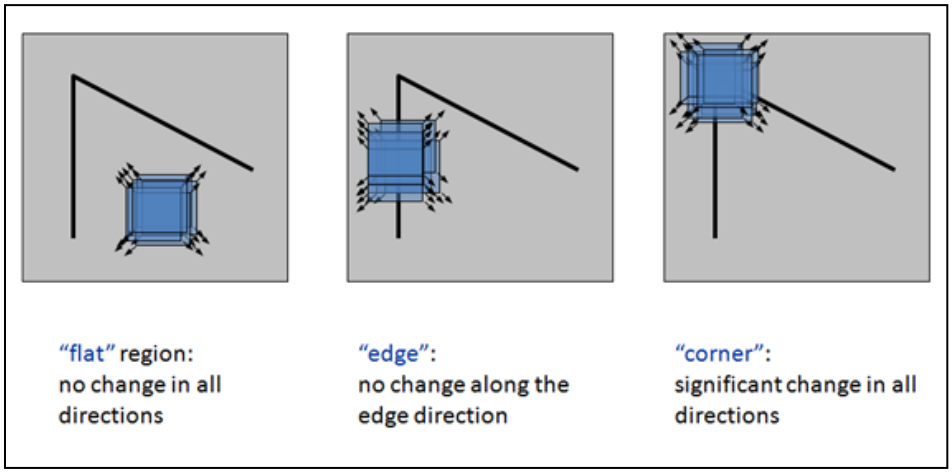


Figure 10: Harris Corner Detector [1]

The change produced by a shift of a window *w* of size (*u*, *v*) by (*x*, *y*) in an image with intensities I, can be expressed by:

$$E_{x,y} = \sum_{u,v} w_{u,v} \left| I_{x+u,y+v} - I_{u,v} \right|^2.$$

[1] Figure from [5]: (http://www.wisdom.weizmann.ac.il/~deniss/2004-03_invariant_features/InvariantFeatures.pdf)

In the equation above, E is the change produced by a shift, w is the window function, I_i are image intensities and the difference between intensities is nearly 0 for a constant patch. In the Harris corner detector, we are using a Gaussian window ($e^{-\frac{u^2+v^2}{2\sigma^2}}$), covering all possible small shifts by performing an analytic expansion about the shift at the origin and making use of the variation of E with the direction of shift.

By performing a Taylor series expansion of E about the origin, the local autocorrelation matrix is derived using first order derivatives:

$$E(x,y) = \sum_{u,v} w(u,v)\,[I(x+u,y+v) - I(u,v)]^2$$

$$= \sum_{u,v} w(u,v)\left[\cancel{I(u,v)} + I_x(u,v)x + I_y(u,v)y - \cancel{I(u,v)}\right]^2$$

$$= \sum_{u,v} w(u,v)\left[I_x(u,v)x + I_y(u,v)y\right]^2$$

$$= \sum_{u,v} w(u,v)\left\{[I_x(u,v)x]^2 + \left[I_y(u,v)y\right]^2 + 2I_x(u,v)I_y(u,v)xy\right\}$$

Here, $I_x = \partial I/\partial x$ and $I_y = \partial I/\partial y$, the gradient vectors are a set of (dx, dy) points with a center of mass defined as being at (0,0), and E is a close approximation of the local autocorrelation function, with M describing its shape at the origin (the quadratic terms in the Taylor expansion). The local autocorrelation function is therefore:

$$E(x,y) = [x\ y]\ \mathbf{M}\ \begin{bmatrix}x\\y\end{bmatrix}, \text{ where } \mathbf{M} = \sum_{u,v} w(u,v)\begin{bmatrix}{I_x}^2 & I_xI_y\\ I_xI_y & {I_y}^2\end{bmatrix},$$

$$\text{or } E(x,y) = [x\ y]\left(\sum_{u,v} w(u,v)\begin{bmatrix}{I_x}^2 & I_xI_y\\ I_xI_y & {I_y}^2\end{bmatrix}\right)\begin{bmatrix}x\\y\end{bmatrix}$$

Here, E(x, y) = t is an ellipse, if t is a constant (Figure 11). Let λ_1, λ_2 be the eigenvalues of *M*. The eigenvalues are proportional to the principal curvatures of the autocorrelation function

E, and form a rotationally invariant description of M: the direction of fastest change is the eigenvector parallel to the minor axis of the ellipse (minimum eigenvalue) and the direction of slowest change is the eigenvector parallel to the major axis of the ellipse (maximum eigenvalue).

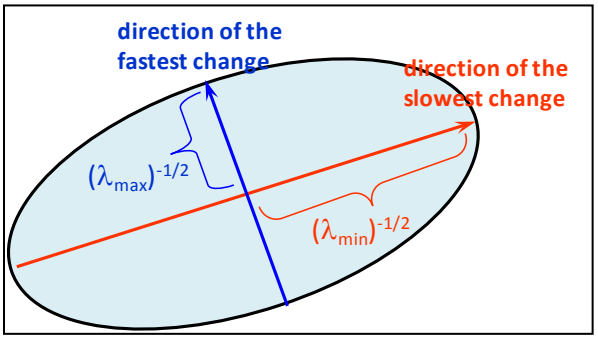


Figure 11: Autocorrelation Function Graphical Representation [2]

Instead of computing eigenvalues, we would like to find a measure of corner and edge quality or response size of the response used to select isolated corer pixels and to thin the edge pixels.

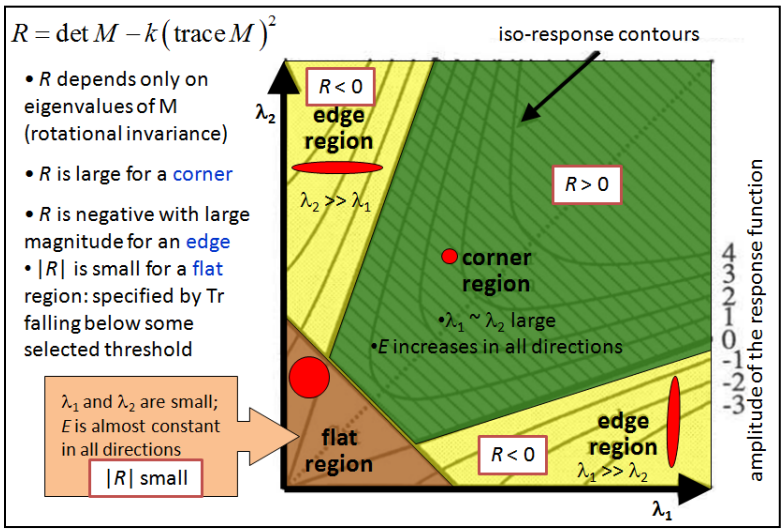


Figure 12: Classification of Image Points Based on Response Size Using the Eigenvalues of M [3]

---

[2] Figure from [5]: (http://www.wisdom.weizmann.ac.il/~deniss/2004-03_invariant_features/InvariantFeatures.pdf)
[3] The figure is an adaptation of Figure 5 from [1]

The paper (Harris & M., 1988) finds an acceptable response function to be $R = \det M - k(\text{trace}M)^2$, where $\det M = \lambda_1\lambda_2$, $\text{trace}M = \lambda_1 + \lambda_2$ and k is an empirical constant between 0.04 and 0.06.

For the Harris detector, a corner region pixel (i.e. one with a positive response) is selected as a nominated corner pixel only if its response is an 8-way local maximum (Figure 12). Edge region pixels are deemed to be edgels if their responses are both negative and local minima in either the x or y directions (in whichever direction the magnitude of the first gradient is larger).

In (Noble, 1989), Noble finds another adequate response function, $R_2 = \frac{\text{determinant(A)}}{\text{trace(A)}} = \frac{I_x{}^2 I_y{}^2 - (I_x I_y)^2}{I_x{}^2 + I_y{}^2}$, where $A = \begin{bmatrix} I_x{}^2 & I_x I_y \\ I_x I_y & I_y{}^2 \end{bmatrix}$. This function has the advantage that it does not rely on a (somewhat arbitrary) value for *k*.

4.4. Algorithm for the Harris Corner Detector

The algorithm behind the Harris Corner Detector is as follows:

1. Compute the x and y derivatives of the image, I_x and I_y
2. Compute the products of derivatives at every pixel: $I_x{}^2$, $I_y{}^2$, I_xI_y
3. Compute the sums of the products of derivatives at each pixel, so as to define the matrix M at each pixel: $M = \sum_{u,v} w(u,v) \begin{bmatrix} I_x{}^2 & I_x I_y \\ I_x I_y & I_y{}^2 \end{bmatrix}$
4. Compute the response detector R at each pixel: $R = \det M - k(\text{Trace}M)^2$
5. Threshold on value of R:
    - Find points with large corner response function R (R > threshold)

- Compute nonmax suppression (take the points of local maxima of R; suppress all image information that is not part of local maxima)
  - Scan image along the image gradient direction
  - If pixels are not part of the local maxima they are set to zero

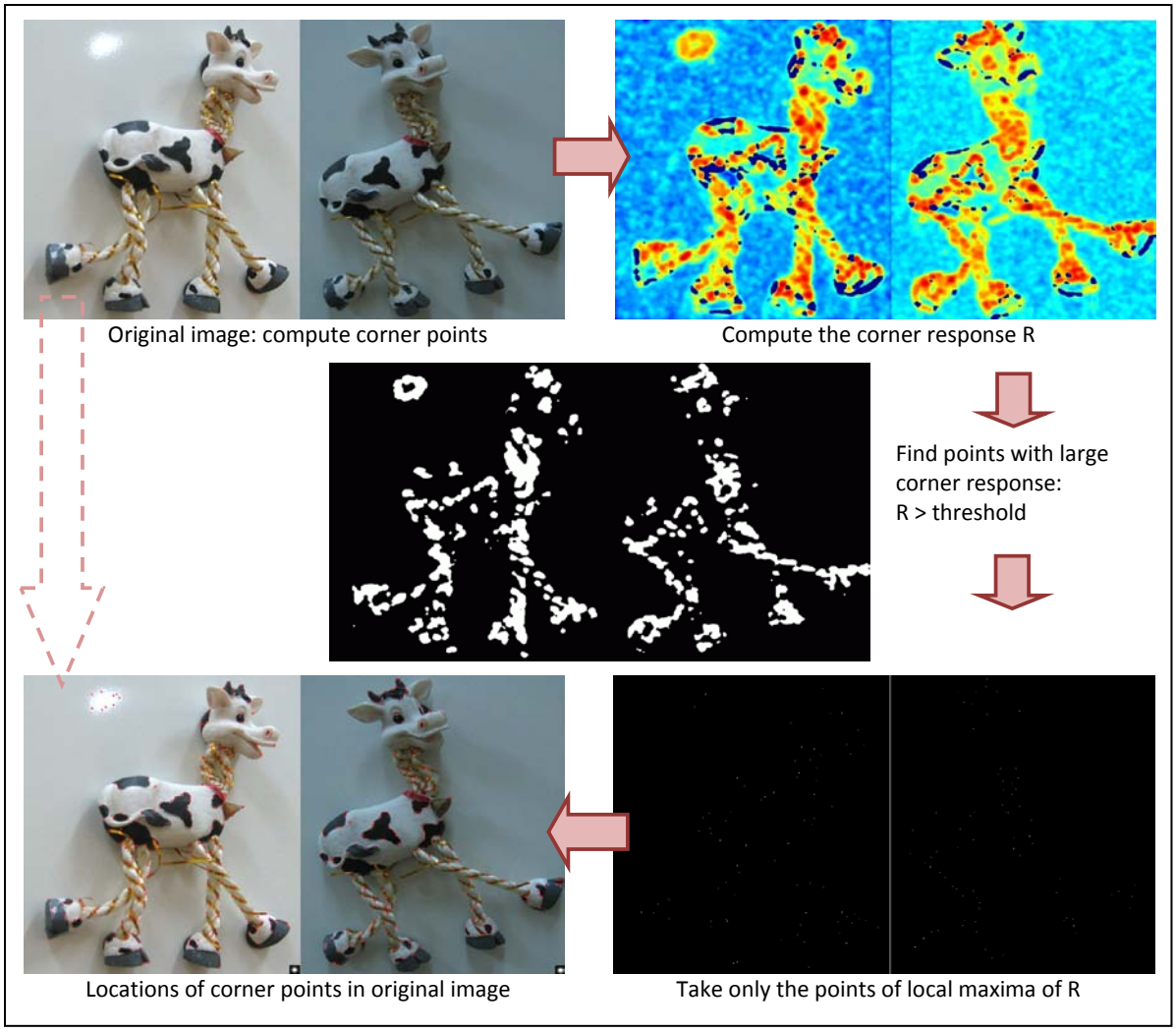


Figure 13: Harris Corner Point Detection Workflow [4]

Figure 13, above, presents a sample workflow for the algorithm.

[4] Figure from [5] (http://www.wisdom.weizmann.ac.il/~deniss/2004-03_invariant_features/InvariantFeatures.pdf)

4.5. Why Use the Harris Corner Detector. Problems

One of the most important reasons for using the Harris Corner Point Detector as opposed to other means is the fact that the resulting corner points are invariant in position to many image transformations, such as to rotations and partially to affine intensity changes.

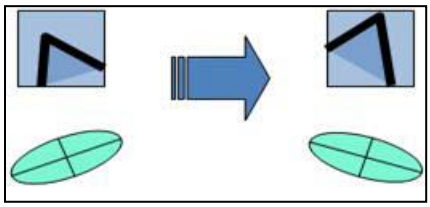


Figure 14: Rotation Invariance [5]

Although the detected points are not invariant to image scale, there are ways to overcome those problems.

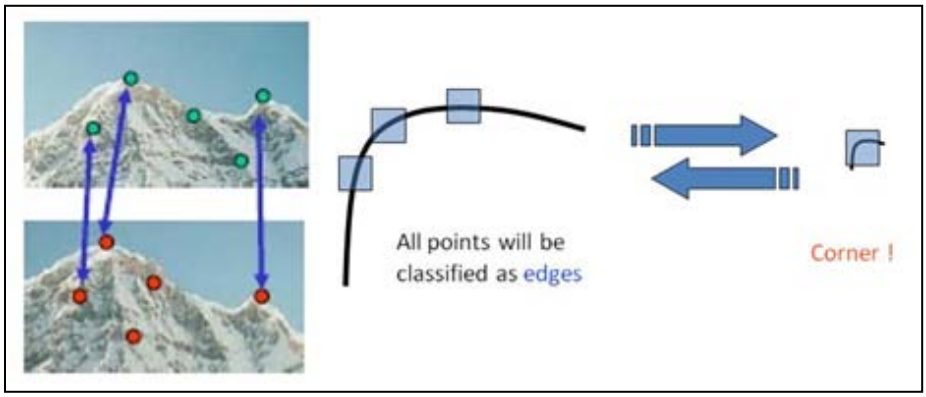


Figure 15: The Harris Corner Detector is Not Invariant to Image Scaling [6]

The rotation invariance is given by the fact that rotating an ellipse preserves its shape, and therefore its eigenvalues, and the autocorrelation function (Figure 14). The Harris Corner Point Detector is also partially invariant to affine intensity changes (i.e. linear scaling of intensity) plus a constant offset, $I_2 = aI + b$ (Figure 16). Since only derivatives are used, the

[5] Figure from [5] (http://www.wisdom.weizmann.ac.il/~deniss/2004-03_invariant_features/InvariantFeatures.pdf)
[6] Figures from [5](http://www.wisdom.weizmann.ac.il/~deniss/2004-03_invariant_features/InvariantFeatures.pdf)

Harris Corner Point Detector is invariant to intensity shifts: $I_2 = I + b$, and also partially insensitive to intensity scaling: $I' = aI$.

It is only partially invariant because sometimes intensity changes place some of the original corner points above or below the selected threshold, so as long as the threshold doesn't get readjusted accordingly, some points will be included in one of the images and some will not.

Unfortunately, the Harris Corner detector is not invariant to image scaling (Figure 15). When an image is scaled, corners may transform to edges or edges may transform to corners.

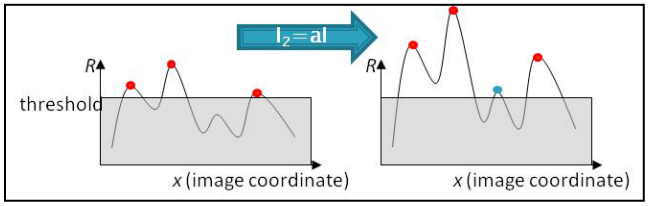


Figure 16: Partial Invariance to Affine Intensity Change [7]

4.5. Invariant Corner Detector: Window Function

Since the purpose of using feature points is to match images against each other, it makes sense that we are interested in detecting the same corner points regardless of image changes. In order for the same points to be computed, we need to adjust the threshold appropriately and independently within each of the images, in such a way that the window function would identify the same corner points across both images. The solution is to design a window function such that it is scale invariant, i.e. the function changes as the scale changes. One example of such a function, that would be appropriate if cropping was not considered as one of the image transformations, is the average intensity of the image.

---

[7] Figure from [5]:(http://www.wisdom.weizmann.ac.il/~deniss/2004-03_invariant_features/InvariantFeatures.pdf)

4.6. MATLAB Code for Harris Corner Detector

4.6.1. Discrete Convolution Matrices. 2-D Convolutions

A convolution (Ahn, 2005) is a mathematical operation on two functions, producing a third function that is a modified version of one of the original functions. If *f* and *g* are functions of an integer variable *n*, then the formula for the discrete convolution of *f* and *g*, where *m* is a random integer (time-shift), is:

$$(f * g)(n) = \sum_{n=-\infty}^{\infty} f(m)g(n-m) = \sum_{-\infty}^{\infty} f(n-m)g(m)$$

2D convolution is just extension of previous 1D convolution by convolving both horizontal and vertical directions in 2 dimensional spatial domains. If *f* and *g* are functions of two discrete variables, n_1 and n_2, then the formula for the two-dimensional convolution of *f* and *g*, each shifted by k_1 and k_2 respectively, is:

$$c(n_1, n_2) = (f * g)(n_1, n_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f(n_1, n_2)g(n_1 - k_1, n_2 - k_2)$$

The MATLAB function *conv2* uses a straightforward formal implementation of the two-dimensional convolution equation (above) in spatial form. In practice conv2 computes the convolution for finite intervals.

4.6.2. The MATLAB program

The Harris Algorithm uses the response measure given by Nobel in her thesis. The original Harris measure is commented out. The inputs of the function are the image on which we are identifying corner points, *im*, the standard deviation of smoothing Gaussian, *sigma* (with best

values 1 to 3), the threshold *thresh*, and the radius of region considered in nonmax suppression *radius* (with best values 1 to 3).

The function returns the array *cim* (of the same size as *im*) which is the response detector at each pixel according to (Noble, 1989), as well as the 1D arrays *r* and *c*, which represent the row and column coordinates of the corner points, respectively. The algorithm first computes the derivatives that form matrix $A = \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}$ and the response function $cim = \frac{\text{determinant(A)}}{\text{trace(A)}} = \frac{{I_X}^2 {I_y}^2 - (I_x I_y)^2}{{I_X}^2 + {I_y}^2}$. Then, nonmax suppression is performed such that all the information in cim that is not part of the local maxima is discarded.

```
function [cim, r, c] = harris(im, sigma, thresh, radius)
    error(nargchk(2,5,nargin)); % validate the number of input arguments
    dx = [-1 0 1; -1 0 1; -1 0 1]; % special matrix for convolution
    dy = dx'; % the transpose of dx, also used for convolution

    % Compute the x and y derivatives of the image, Ix and Iy
    Ix = conv2(single(im), single(dx), 'same');
    Iy = conv2(single(im), single(dy), 'same');

    g = fspecial('gaussian',max(1,fix(6*sigma)), sigma); % Gaussian filter

    % Compute the products of derivatives at every pixel: Ix^2, Iy^2, IxIy
    Ix2 = conv2(single(Ix.^2), single(g), 'same');
    Iy2 = conv2(single(Iy.^2), single(g), 'same');
    Ixy = conv2(single(Ix.*Iy), single(g), 'same');

    % cim is the determinant/trace
    cim = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps);
    %k = 0.04;
    %cim = (Ix2.*Iy2 - Ixy.^2) - k*(Ix2 + Iy2).^2;
    % Threshold on cim
    if nargin > 2
        sze = 2*radius+1;
        mx = ordfilt2(cim,sze^2,ones(sze));
        % Find local maxima
        cim = (cim==mx)&(cim>thresh);
        [r,c] = find(cim);
    end
```

Code Snippet 9: Harris Corner Detector Function

A few notes on the MATLAB pre-defined functions used in the code:

- *error(MSG)* displays error message MSG and aborts function;
- *nargchk(LOW,HIGH,N)* returns an appropriate error message string if N is not between LOW and HIGH and an empty matrix otherwise;
- *nargin* returns the number of input arguments that were used to call the function;
- *single(X)* converts convert the variable *X* to single precision for less storage space;
- *conv2(A,B, 'same')* computes the 2D convolution of *A* and *B* and returns the central part of the convolution (basically the intensity derivatives I_x, I_y, I_xI_y, I_x^2, I_y^2);
- *fspecial('gaussian',HSIZE,SIGMA)* returns a predefined, rotationally symmetric Gaussian lowpass filter of size HSIZE with standard deviation SIGMA (positive);
- *ordfilt2(A,ORDER,DOMAIN)* computes the 2D order-statistic filtering by replacing each element in A by the ORDER-th element in the sorted set of neighbors specified by the nonzero elements in DOMAIN;
- *eps* returns the distance from 1.0 to the next larger double precision number ($=2^{-52}$);
- *ones(sze)* returns a 1-by-sze matrix of ones;
- *find(X)* returns the linear indices corresponding to the nonzero entries of the array X.

5. GRAPHS. GRAPH SIMILARITY

The definitions and examples in this chapter are taken from (West, 1996, 2001), (MathWorks, 1984-2011), (Bondy & Murty, 1976), (William & Hirschfeld, 2001), sometimes verbatim. They are provided as the theoretical background for the project.

5.1. Graphs. General Definitions

5.1.1 Graphs. Finite Graphs. Simple Graphs. Vertex Degree

Definition: A *graph* G is a triple consisting of a *vertex set* V(G), an *edge set* E(G), and a *relation* that associates with each edge two vertices (not necessarily distinct), called endpoints.

A graph is *finite* if its vertex set and edge set are finite. Every graph computed on images, as done in this project, is finite. *Drawing* a graph means placing the vertices as points in the plane and representing the edges as curves joining the locations of their endpoints.

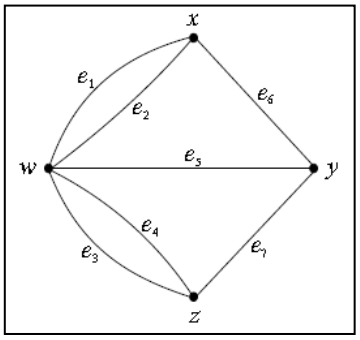


Figure 17: Graph (G) Example

Example:

In the graph (Figure 17) to the left:

- the vertex set is V = {*x*, *y*, *z*, *w*}
- the edge set is E = {$e_1 = wx$, $e_2 = wx$, $e_3 = wz$, $e_4 = wz$, $e_5 = wy$, $e_6 = xy$, $e_7 = yz$}

For the purposes of this project, the graphs we are dealing with are *simple graphs*, which are graphs that have no *loops* (edges whose two endpoints are the same) or *multiple edges* (edges having the same pair of endpoints). In other words, in graph G in Figure 17, edges e_1 and e_2 are not different edges, and neither are edges e_3 and e_4.

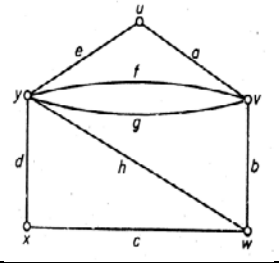
A vertex *v* and an edge *e* are *incident* if *v* is an endpoint of *e*. In this case, we call *e* an *incident edge* of *v*. The *degree* of a vertex is the number of incident edges for that vertex.

5.1.2. Walks. Trails. Paths. Cycles. Connected Graphs

Definition: A *walk* in G is a finite non-null sequence $W = v_0e_1v_1 \ldots e_kv_k$ of vertices and edges such that for $1 \leq i \leq k$, the edge e_i has endpoints v_{i-1} and v_i. We say that W is a walk from v_0 to v_k, or a v_0, v_k–walk. In a simple graph, a walk $v_0e_1v_1 \ldots e_kv_k$ is determined by the sequence $v_0v_1 \ldots v_k$ of its vertices; hence a walk in a simple graph can be specified simply by its vertex sequence.

A *trail* is walk with no repeated edges. A *path* is a trail with no repeated vertices. A *u,v–path* is a path whose vertices of degree 1 (its endpoints) are *u* and *v* and the others are *internal* vertices.

Two vertices *u* and *v* of G are said to be *connected* if there is a *u,v–path* in G. A graph is *connected* if all of its vertices are connected.



A walk: *uavfyfvgyhwbv*
A trail: *wcxdyhwbvgy*
A path: *xcwhyeuav*
Sample cycles: *wcxdyhw* and *yhwbvgy*

Figure 18: Example of Walk, Trail, Path, and Cycles [8]

A walk is *closed* if it has positive length and its origin and terminus are the same. A closed trail whose origin and internal vertices are distinct is called a *cycle.*

In Figure 18, we have examples of a walk, a trail, a path, and two cycles.

---

[8] The figure is adapted from Figure 1.8 of [12]

5.1.3. Directed Graphs

Definition: A *directed graph* or *digraph* G is a triple consisting of a *vertex set* V(G), an *edge set* E(G), and a function assigning each edge an ordered pair of vertices (Figure 19). The first vertex of the ordered pair is the *tail* of the edge, and the second is the *head*; together, they are called the *endpoints*. We say that an edge is an edge *from its tail to its head.*

In this project we are only considering undirected graphs, since there is no specific direction for edges when constructing the Voronoi Diagram.

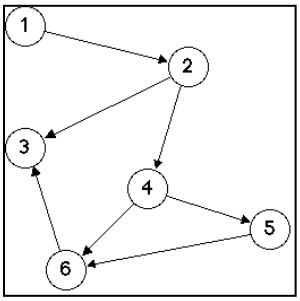


Figure 19: A Directed Graph[9]

5.1.4. Weighted Graphs

A *weighted graph* is a graph with numerical labels on its edges. These labels can represent a variety of costs, such as distance, price, time etc. In the project, weighted graphs, with weights being actual distances between the two endpoints of the edges (i.e. the lengths of the edges) were considered. Ultimately, for this stage of the project, I have nevertheless opted for unweighted graphs, though distances have the potential of eliminating many false positive results.

---

[9] Image taken from http://www.cs.hmc.edu/~keller/courses/cs60/s98/examples/acyclic/

5.1.5. Edge Crossings. Graph Faces. Planar Graphs. Plane Graphs. Dual Graphs

Definition: A *curve* is the image of a continuous map from [0,1] to $\mathbb{R}^2$. A *polygonal curve* is a curve composed of finitely many line segments. It is a *polygonal u,v–curve* when it starts at *u* and ends at *v*.

A *drawing* of a graph G is a function *f* defined on V(G) ∪ E(G) that assigns each vertex *v* a point *f(v)* in the plane and assigns each edge with endpoints *u, v* a polygonal *f(u), f(v)*–curve. The images of vertices are distinct. A point in *f(e)* ∩ *f(e')* that is not a common endpoint is a *crossing.*

Definition: A graph is *planar* if it has a drawing without crossings. Such a drawing is called a *planar embedding* of G. A *plane graph* is a particular planar embedding of a planar graph.

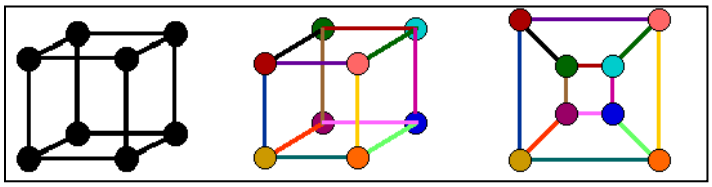


Figure 20: The Cube Graph; Edge Crossing; Planar Representation [10]

The first representation in Figure 20 shows a special type of graph, the cube graph; the second representation shows the edge crossings, while the third representation shows a re-drawing without edge crossings. Therefore, the cube graph is a planar graph.

Definition: An *open set* in the plane is a set U ⊆ $\mathbb{R}^2$ such that for every *p* ∈ U, all points within some small distance from *p* belong to U. A *region* is an open set U that contains a polygonal *u, v*–curve for every pair *u, v* ∈ U. The *faces* of a plane graph are the maximal regions

[10] Image adapted from: http://www.math.lsa.umich.edu/mmss/coursesONLINE/graph/graph5/

of the plane that contain no point used in the embedding. The faces are thus the regions bounded by the edges of the graph.

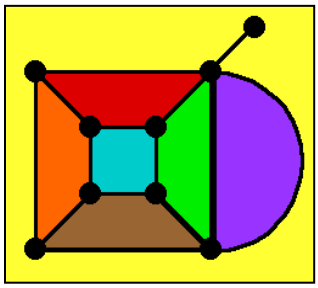


Figure 21: Faces of a Graph [11]

Figure 21 shows the faces of a graph, each colored in a different color. The exterior face, colored in yellow, is called an *infinite face*.

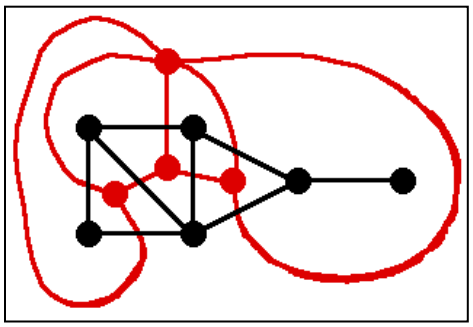


Figure 22: Dual Graphs[12]

Definition: The *dual graph* G^* of a plane graph G is a plane graph whose vertices correspond to the faces of G and whose edges correspond to the edges of G as follows: if *e* is an edge of G with face X on one side and face Y on the other, then the endpoints of the *dual edge* $e^* \in E(G^*)$ are the vertices $x, y \in G^*$ that represent the faces X, Y of G. The order in the plane of the edges incident to $x \in V(G^*)$ is the order of the edges bounding the face X of G in a walk along its boundary.

---

[11] Image taken from http://www.math.lsa.umich.edu/mmss/coursesONLINE/graph/graph5/
[12] Image taken from http://www.math.lsa.umich.edu/mmss/coursesONLINE/graph/graph5/

In Figure 22, the original graph is drawn in black, and its dual is drawn in red. It is important to note that, as defined in Chapters 6 and 7, Voronoi Graphs and Delaunay Graphs are dual graphs.

5.2. Graph Similarity. Possible Measures of Similarity

In this project, we are concerned with what makes two graphs related, and more specifically with measures of such connectedness. In the simplest sense, two Delaunay graphs of the same image would be identical, both in number of vertices, edge connections and in vertex labeling. In the same way, the graphs of scaled and sheared images would be identical to the original graph if distances between vertices do not matter (so the graphs are not weighted) and the graphs of rotated and reflected images would be identical but with different vertex labeling (a graph isomorphic to the original). The graphs of cropped images would be subgraphs of the original image graph, and the similarity between the original image and the images obtained by blurring, contrast auto-adjustment, and changing luminosity will consist of either induced subgraphs of the original graph or of minors of the original graph. Other similarity measures considered are chromatic numbers and the graph diameter.

5.2.1. Subgraphs. Induced subgraphs

Definition: A *subgraph* of a graph G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ and the assignment of endpoints to edges in H is the same as in G.

A subgraph H is a *spanning subgraph* of a graph G if H and G have the same vertex set. We then say that H *spans* G. H is an *induced subgraph* of G if, for any pair of vertices *u* and *v* of H, *uv* is an edge of H if and only if it also is an edge of G, that is if H has exactly the same edges

of G over its vertex set. If the vertex set of H is the subset S of the vertex set of G,V(G), then we can say H is induced by S and write H as G[S].

5.2.2. Isomorphic Graphs

Definition: An *isomorphism* from a simple graph G to a simple graph H is a bijection f: V(G) → V(H) such that $uv \in E(G)$ if and only if $f(u)f(v) \in E(H)$. If there is an isomorphism between G and H, then G is *isomorphic* to H.

For example, the graphs G_1 and G_2 on page 32 are isomorphic to each other, and the isomorphism f: $V(G_1) \rightarrow V(G_2)$ is given by: $f(1_1) = 3_2$, $f(2_1) = 4_2$, $f(3_1) = 1_2$, $f(4_1) = 2_2$.

5.2.3. Edge Contraction. Graph Minors

The *contraction* of an edge *e* of G with endpoints *u* and *v* is the deletion of *e* such that its endpoints *u* and *v* are identified and the edges incident to the new vertex are the edges, other than *e*, that were incident to *u* or to *v*; the resulting graph is denoted by $G \cdot e$ and has one less edge than G. An example of edge contraction is given in Figure 23.

Definition: A graph H is a *minor* of another graph G if H can be obtained from a subgraph of G by edge contraction. An equivalent definition states that H is a minor of G if there is a map that associates with each vertex *v* of H a non-empty, connected subgraph G_v of G such that G_v and G_u are disjoint for $u \neq v$ and if there is an edge between *u* and *v* in H then there is an edge in G between some vertex in G_v and some vertex in G_u.

In Figure 23, the graph on the right is a minor of the graph on the left.

Unfortunately for the relative execution time of the graph-based image matching program, the determination of graph minors is an NP-hard problem for which no good

algorithms are known (Wolfram Research, 2011). Nevertheless, brute-force methods exist, such as those given by Robertson, Sanders, and Thomas in (Barber, Dobkin, & Huhdanpaa, 1996).

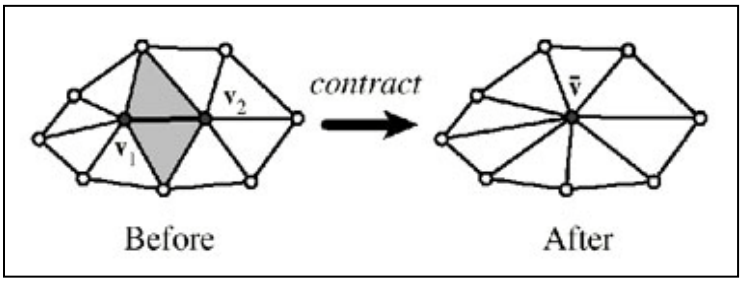


Figure 23: Contracting edge v_1v_2 to vertex $\bar{v}$ [13]

5.2.4. Chromatic Numbers

Definition: A *k-coloring* of a graph G is a labeling f: V(G) → S, where |S| = *k*. The labels are *colors*; the vertices of one color form a color class. A *k*-coloring is *proper* if adjacent vertices have different labels. A graph is *k-colorable* if it has a proper *k*-coloring. The *chromatic number* χ(G) is the least *k* such that G is *k*-colorable.

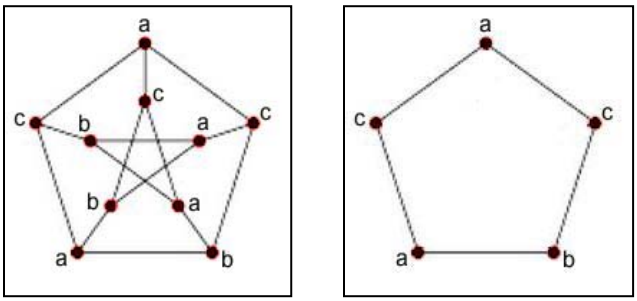


Figure 24: Graph Coloring Examples with χ = 3

[13] The image is taken from http://www.cg.tuwien.ac.at/studentwork/VisFoSe98/msh/

5.2.5. Graph Diameter

Definition: If G has a *u*,*v*–path, then the *distance* from *u* to *v*, written as $d_G(u, v)$ or simply $d(u, v)$ is the least length of a *u*,*v*–path. If G has no such path, then $d(u, v) = \infty$. The *diameter* of a graph G, diam(G), is the maximum such distance, $\max_{u,v \in V(G)} d(u,v)$.

In other words, a graph's diameter is the largest number of vertices which must be traversed in order to travel from one vertex to another when paths which backtrack, detour, or loop are excluded from consideration (Wolfram Research, 2011).

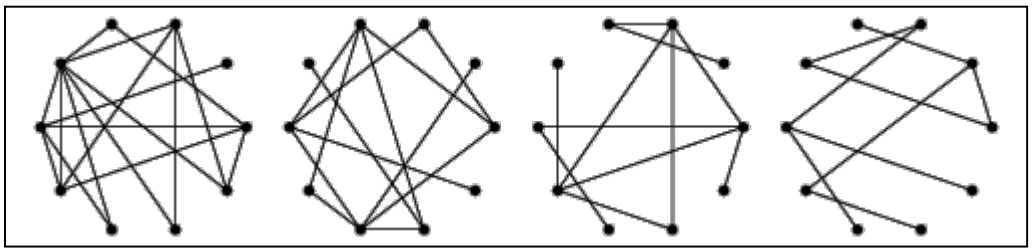


Figure 25: Graph Diameter Example [14]

In Figure 25, all the graphs have 10 vertices and diameter 3, 4, 5, and 7, respectively.

5.3. Graph Representations

Definition: An *adjacency matrix* of a graph G, A(G), is, the n-by-n (square) matrix in which entry $a_{i,j}$ is the number of edges in G with endpoints $\{v_i, v_j\}$. An *incidence matrix* of a graph G, M(G), is the n-by-m matrix in which $m_{i,j}$ is 1 if v_i is an endpoint of e_j, and 0 otherwise.

In computer science, graphs are most often represented using the adjacency matrix. Note that, for simple graphs, the adjacency matrix is the square matrix in which entry $a_{i,j}$ is 1 if there is an edge with endpoints $\{v_i, v_j\}$ and 0 otherwise. Also, since the graphs we are dealing with are

[14] The image was taken from http://mathworld.wolfram.com/GraphDiameter.html

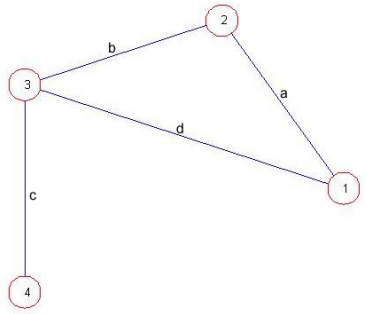
not oriented graphs, the adjacency matrix is symmetric about the principal diagonal, whose entries are all 0.

	1	2	3	4
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1
4	0	0	1	0

Adjacency Matrix $A_1(G_1)$

	a	b	c	d
1	1	0	0	1
2	1	1	0	0
3	0	1	1	1
4	0	0	1	0

Incidence Matrix $M_1(G_1)$



Graph G_1

Figure 26: Graph G_1 Representations

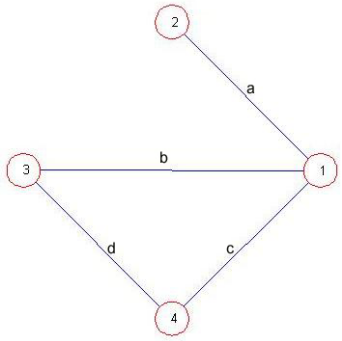
A graph can also be represented by its degree sequence, where a degree sequence is a monotonic non-increasing sequence of the vertex degrees of its vertices. For example, for the graph in Figure 26, the degree sequence is {3, 2, 2, 1}. Since the graph in Figure 27 has the same degree sequence, it becomes apparent that they are in fact the same graph (an isomorphism).

	1	2	3	4
1	0	1	1	1
2	1	0	0	0
3	1	0	0	1
4	1	0	1	0

Adjacency Matrix $A_2(G_2)$

	a	b	c	d
1	1	1	1	0
2	1	0	0	0
3	0	1	0	1
4	0	0	1	1

Incidence Matrix $M_2(G_2)$



Graph G_2

Figure 27: Graph G_2 Representations

Another method to represent graphs that is sometimes used in computer science (and is also used in some of the MATLAB functions employed in this project) implies two arrays: one array representing the vertices of the graph (either in 1D, containing the vertex labels, or in 2D,

containing the coordinates for each vertex), and the other one representing the edge connections. In the simplest implementation, this second array is 2-dimmensional and contains the two indices of the vertices in the first array that constitute the endpoints of each edge. If the first array is 1D, sometimes it can be omitted and the vertex indices in the second array can be replaced with the actual labels of the vertices. In more complicated cases, the second array can be multi-dimensional, and contain all cycles in the graph with every edge represented only once (the MATLAB data structure for arrays within arrays or arrays of matrices is called a cell).

For example, for graph G_1 in Figure 26, we have the vertex matrix V_1 = [1, 2, 3, 4] and the edge matrix E_1 = [[1, 2], [1, 3], [2, 3], [3, 4]].

6. VORONOI DIAGRAMS AND DELAUNAY TRIANGULATIONS

The definitions and other theoretical aspects in this chapter are taken from (Dirichlet., 1850), (Voronoi, 1908), (Fukuda, 2004), (Rosenlicht, 1985), (Aurenhammer F.) and Lectures 16 and 17 of (Pless, 2003), sometimes verbatim. They are provided in order to give a background into the computational geometry constructs used as a basis for the project.

6.1. Theoretical Preliminaries

Please note that, although general definitions are given in what follows, we are dealing with only two dimensions, so $n = 2$.

6.1.1. Metric Spaces. The Euclidean Metric Space

Definition: A *metric space* is a set E, together with a rule which associates with each pair $p, q \in E$ a real number d(p, q) such that:

1. $d(p, q) \geq 0$ for all $p, q \in E$
2. $d(p, q) = 0$ if and only if $p = q$
3. $d(p, q) = d(q, p)$ for all $p, q \in E$
4. $d(p, r) < d(p, q) + d(q, r)$ for all $p, q, r \in E$ (triangle inequality).

Thus a metric space is an ordered pair (E, n), where E is a set and n is a function $n: E^2 \rightarrow R$ satisfying properties (1) to (4).

For any positive integer n we define a metric space E_n, called an *n-dimensional Euclidean space*, by taking the underlying set of E_n to be all n-tuples of real numbers $\{(a_1, \ldots, a_n) \mid a_i \in \mathbb{R}\}$, and defining, for $p = (x_1, \ldots, x_n)$, $q = (y_1, \ldots, y_n)$,

$$d(p,q) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

That is, the *Euclidean metric* is the ordinary distance used in the day-to-day world and that is given by the Pythagorean formula. The metric space assumed in the project, for all the types of diagrams defined, is the Euclidean Space.

6.1.2. Convex Polytopes. Convex Polyhedrons

Definition: A subset P of $\mathbb{R}^n$ is called a *convex polyhedron* if it is the set of solutions to a finite system of linear inequalities, and called a *convex polytope* if it is a convex polyhedron and it is also bounded. When a convex polyhedron (or polytope) has dimension *k*, it is called a k-*polyhedron* (or a *k*-polytope).

6.1.3. Faces of a Convex Polytope/Polyhedron

Definition: Let P be a convex n-polyhedron (or n-polytope) in $\mathbb{R}^n$. For a real n-vector c and a real number n, a linear inequality $c^T x \leq n$ is called *valid* for P if the inequality holds for all $x \in P$. A subset F of a polyhedron P is called a *face* of P if it is represented as $F = P \cap \{x: c^T x = n\}$ for some valid inequality $c^T x \leq n$.

By this definition, both the empty set Ø and the whole set P are faces. These two faces are called *improper faces* while the other faces are called *proper faces*.

We can also define faces geometrically. A hyperplane h of $\mathbb{R}^n$ is *supporting* P if one of the two closed half-spaces (one of the two parts into which a plane divides the three-dimensional space) of h contains P. A subset F of P is called a face of P if it is Ø, P itself or the intersection of P with a supporting hyperplane.

The faces of dimension 0, 1, dim(P)–2 and dim(P)–1 are called the *vertices, edges, ridges* and *facets*, respectively. The vertices coincide with the *extreme points* of P which are

defined as points which cannot be represented as convex combinations of two other points in P. When an edge is not bounded, there are two cases: either it is a line or a half-line starting from a vertex. A half-line edge is called an *extreme ray*.

6.1.4. Simplices. Convex Hulls. Simple / Simplicial Polytopes

Definition: For a subset S of $\mathbb{R}^n$, the *convex hull*, conv(S), is defined as the smallest convex set in $\mathbb{R}^n$ containing S. The convex hull computation means the "determination" of conv(S) for a given finite set of n points S = $\{p^1, p^2, \ldots, p^n\}$ in $\mathbb{R}^n$. If *S* is finite, the convex hull defines a matrix *A* and a vector *b* such that for all *x* in *S*, $Ax+b <= [0,...]$.

Definition: A subset P of $\mathbb{R}^n$ is called a *k*-simplex (*k* = 0, 1, 2….) if it is the convex hull of *k*+1 affinely independent points. It has exactly *k*+1 vertices and *k*+1 facets.

An n-polytope is called *simple* if each vertex is contained in exactly n facets. An n-polytope is called *simplicial* if each facet contains exactly n vertices. By definition, a *dual* of a simple polytope is simplicial and the other way around. Every facet of a simplicial n-polytope is a (n–1)–simplex. Each vertex of a simple n-polytope is contained in exactly n-edges.

6.1.5. Cell Complexes. Triangulations

Definition: A *cell complex* (a complex) in $\mathbb{R}^n$ is a set K of convex polyhedra (called cells) in $\mathbb{R}^n$ satisfying two conditions:

1. Every face of a cell is a cell (i.e. in K), and
2. If P and P' are cells, then their intersection is a common face of both.

A *simplicial complex* is a cell complex whose cells are all simplices.

The *body* |K| of a complex K is the union of all cells. When a subset P of $\mathbb{R}^n$ is the body of a simplicial complex K, then K is said to be a triangulation of P. For a finite set S of points in $\mathbb{R}^n$, a *triangulation* of S is a simplicial complex K with |K| = conv(S).

6.2. Voronoi Diagrams

6.2.1. Motivation

First studied by Voronoi (Voronoi, 1908) and Dirichlet (Dirichlet., 1850), the Voronoi/Dirichlet spatial tessellations have applications in a variety of disciplines, including image compression, epidemiology, climatology, deriving the capacity of wireless networks, computer graphics, robot navigation, cellular biology, statistics, the territorial behavior of animals etc. (Aurenhammer & Klein, 2000), (Okabe, Boots, Sugihara, & Chiu., 2000), (Du, Faber, & Gunzburger, 1999). In this project, Voronoi diagrams are used as an image segmentation method for auto-adjusting the threshold in order to obtain the best possible Harris Corner Points to generate the best Delaunay Triangulation for each of the two images to be compared.

6.2.2. Voronoi Diagrams (Tessellations)

Definition: A *Voronoi diagram* (also called a Voronoi tessellation, Voronoi decomposition, or Dirichlet tessellation) is a special kind of decomposition of a metric space determined by distances to a specified discrete set of objects in the space (usually denoted by a set of points), according to the *nearest-neighbor rule*, such that each point is associated with the region of the plane closest to it.

Given S = $\{p_1, p_2, \ldots, p_n\}$, a set of n distinct points (sites) in $\mathbb{R}^n$, the Voronoi diagram is the partition of $\mathbb{R}^n$ into n polyhedral regions, Vo(p), called the Voronoi cells of p. A *Voronoi cell* is defined as the set of points in $\mathbb{R}^n$ which are closer to p than to any other site, or more precisely, $Vo(p) = \{x \in \mathbb{R}^n | \, \mathrm{d}(x,p) \leq \mathrm{d}(x,q) \; \forall q \in S - p\}$, where d is the Euclidean distance function.

The set of all Voronoi cells and their faces forms a cell complex. The vertices of this complex are called the *Voronoi vertices*, and the extreme rays (i.e. unbounded edges) are the *Voronoi rays*. For each point $v \in \mathbb{R}^n$, the *nearest neighbor set* nb(S, v) of v in S is the set of points, $p \in$ S – v, which are closest to v in Euclidean distance. Alternatively, one can define a point $v \in \mathbb{R}^n$ to be a Voronoi vertex of S if nb(S, v) is maximal over all nearest neighbor sets.

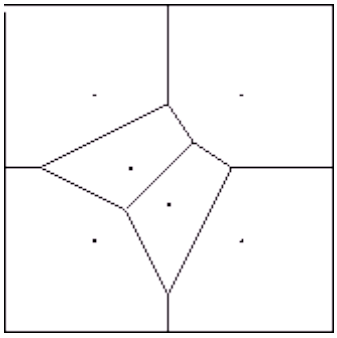


Figure 28: Voronoi Diagram [15]

Another way to define Vo(p_i) is in terms of the intersection of half-planes. Given two sites p_i and p_j, the set of points that are strictly closer to p_i than to p_j is just the open half-plane whose bounding line is the perpendicular bisector between p_i and p_j. Denote this half-plane h(p_i, p_j). Then a point q lies in V(p_i) if and only if q lies within the intersection of h(p_i, p_j) for all $j \neq i$. In other words, $Vo(p_i) = \bigcap_{\forall j} h(p_i, p_j)$.

[15] Image taken from http://www.ifor.math.ethz.ch/~fukuda/polyfaq/node29.html

6.2.3. Properties of Voronoi Diagrams

Voronoi edges: Each point on an edge of the Voronoi diagram is equidistant from its two nearest neighbors p_i and p_j. Thus, there is a circle centered at such a point such that p_i and p_j lie on this circle, and no other site is interior to the circle.

Voronoi vertices: It follows that vertex at which three Voronoi cells $V(p_i)$, $V(p_j)$, and $V(p_k)$ intersect is equidistant from all sites. Thus it is the center of the circle passing through these sites, and this circle contains no other sites in its interior.

Degree: If we assume that no four points are co-circular, then the vertices of the Voronoi diagram all have degree three.

Convex hull: A cell of the Voronoi diagram is unbounded if and only if the corresponding site lies on the convex hull. (Observe that a point is on the convex hull if and only if it is the closest point from some point at infinity.)

Size: If n denotes the number of sites, then the Voronoi diagram is a planar graph (if we imagine all the unbounded edges as going to a common vertex infinity) with exactly n faces. It follows from Euler's formula that the number of Voronoi vertices is at most $2n - 5$ and the number of edges is at most $3n - 6$.

Voronoi diagrams can be represented as graphs, where the Voronoi edges described in Section 6.2.2 are the graph's edges, and the Voronoi vertices are the graph's vertices. Generating Voronoi diagrams based on feature points enables us to ascertain which region in the image is best characterized by the generating feature point embedded.

6.3. Delaunay Triangulations

6.3.1. Motivation

The dual of the Voronoi Diagram is called the Delaunay Triangulation. The Delaunay Triangulation using the best (auto-adjusted) Harris Corner Points is computed on both the original image and on all the images in the database with which the image is to be compared. The Delaunay graphs are then compared two-by-two. In other words, images in the database are represented as Delaunay diagrams.

6.3.2. Delaunay Triangulations

The *vertices for the Delaunay Triangulation* can be taken to be the sites themselves. Since (assuming general position) the vertices of the Voronoi diagram are of degree three, it follows that the faces of the dual graph (excluding the exterior face) will be triangles. The resulting dual graph is a triangulation of the sites, the Delaunay triangulation.

Let S be a set of m points in $\mathbb{R}^n$. The convex hull *conv*(*nb*(S,*v*)) of the nearest neighbor set of a Voronoi vertex *v* is called the *Delaunay cell* of *v*. The Delaunay complex (or triangulation) of S is a partition of the convex hull *conv*(S) into the Delaunay cells of Voronoi vertices together with their faces.

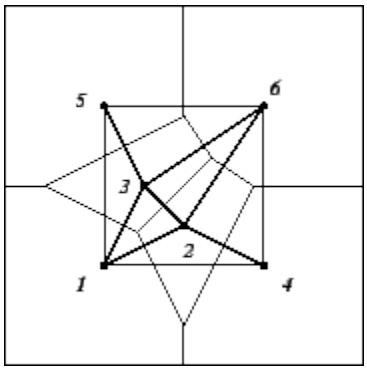


Figure 29: Delaunay Triangulation [16]

The Delaunay complex is not in general a triangulation but becomes a triangulation when the input points are in *general position* (or *non-degenerate*), i.e. no $n+2$ points are co-spherical or equivalently there is no point $c \in \mathbb{R}^n$ whose nearest neighbor set has more than $n+1$ elements.

6.3.3. Properties of Delaunay Triangulations

Delaunay triangulations have a number of interesting properties that are consequences of the structure of the Voronoi diagram.

Convex hull: The exterior face of the Delaunay triangulation is the convex hull of the point set.

Circum-circle property: The circum-circle of any triangle in the Delaunay triangulation is empty (contains no sites of P).

Empty circle property: Two sites p_i and p_j are connected by an edge in the Delaunay triangulation, if and only if there is an empty circle passing through p_i and p_j.

Closest pair property: The closest pair of sites in P are neighbors in the Delaunay triangulation. (The circle having these two sites as its diameter cannot contain any other sites, and so is an empty circle.)

Size: Given a point set P with n sites where there are h sites on the convex hull, in the plane, the Delaunay triangulation has $2n - 2 - h$ triangles, and $3n - 3 - h$ edges.

If the sites are not in general position, in the sense that four or more are co-circular, then the Delaunay triangulation may not be a triangulation at all, but just a planar graph (since the

[16] Image taken from http://www.ifor.math.ethz.ch/~fukuda/polyfaq/node30.html

Voronoi vertex that is incident to four or more Voronoi cells will induce a face whose degree is equal to the number of such cells). In this case the more appropriate term would be Delaunay graph. However, it is common to either assume the sites are in general position (or to enforce it through some sort of symbolic perturbation) or else to simply triangulate the faces of degree four or more in any arbitrary way. MATLAB ensures that the resulting diagram is a triangulation.

Delaunay Triangulations can also be represented as graphs, where the Delaunay edges are the graph's edges, and the Delaunay vertices are the graph's vertices.

7. MATLAB FUNCTIONS: VORONOI, VORONOIN, DELAUNAYTRI, QHULL

7.1. Voronoi

Unfortunately for my choice of programming language, the most evident MATLAB function to use in order to obtain a Voronoi diagram only plots the diagram, without returning any relevant information about its topology.

The MATLAB function voronoi(x,y) plots the bounded cells of the Voronoi diagram for the points *x* and *y*. Lines-to-infinity are approximated with an arbitrarily distant endpoint (MathWorks, 1984-2011) [17]. The function's drawback is that it does not return the locations of the Voronoi vertices or any other information regarding the topology.

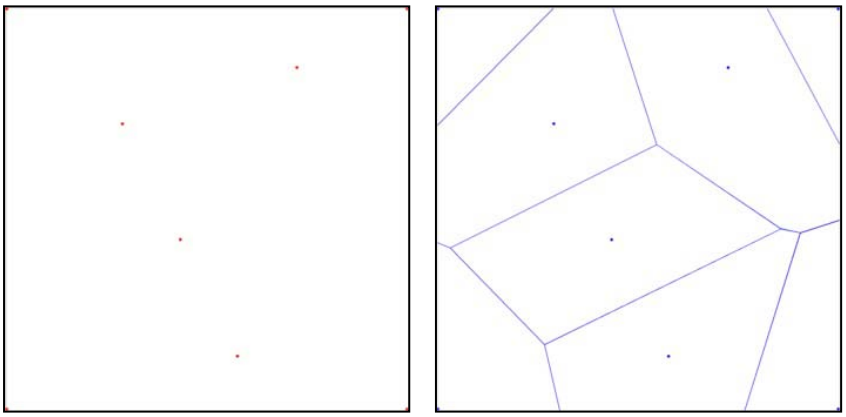


Figure 30: An Example of Voronoi Diagram Creation

A generated Voronoi diagram from a set of points in the plane is shown in Figure 30. To the left, we have the original set of points, and to the right we have the generated Voronoi diagram, which is obtained by calling voronoi on the array that contains the coordinates of the original points, the specific call being voronoi(x, y). Every point on the edge is equidistant from

[17] MATLAB help page on the voronoi function: http://www.mathworks.com/help/techdoc/ref/voronoi.html

the generating points in the two adjacent cells, and every point where three edges intersect is equidistant from the generating points of the three adjacent cells. In Figure 31, the values of x and y for the figure are represented, as well as the value of X which is used in the next section.

x
1
1
145
217
288
362
499
499

y
1
499
144
288
433
74
1
499

X	
1	1
1	499
145	144
217	288
288	433
362	74
499	1
499	499

Figure 31: The Values for the Coorfinates of the Generating Set of Figure 30

7.2. Voronoin

For the topology of the Voronoi diagram, i.e., the vertices for each Voronoi cell, one needs to use another function provided by MATLAB, voronoin.

The voronoin function uses the Qhull algorithm first described in (Robertson, Sanders, Seymour, & Thomas, 1996) and explained in Section 8.4 of this document.

The syntax of the voronoin function is [V, C] = voronoin(X) [18]. The input X is an m-by-n array, representing m n-dimensional points that constitute the centers of the Voronoi cells (the generating set for the diagram), where $n > 1$ and $m \geq n + 1$. In our case, since we are dealing with 2D images, $n = 2$. X is basically the concatenation of x and y, since c represents the x-coordinates of the points and r represents the y-coordinates of the points. X is also represented in Figure 31.

[18] MATLAB help page on the voronoin function: http://www.mathworks.com/help/techdoc/ref/voronoin.html

The function returns the Voronoi vertices V and the Voronoi cells C of the Voronoi diagram of X. Here, V is an array of dimension k-by-n, where each row represents the n-dimensional coordinates of the Voronoi vertex; in our case n = 2. The first row of V is a point at infinity.

V

Inf	Inf
-103.267	250
16.134	298.433
250	924.765
133.624	418.708
273.153	169.924
543.730	250
250	-301.246
200.356	-55.746
451.794	279.636
427.762	274.682

C

[9,2,1,8]
[5,3,2,1,4]
[9,2,3,6]
[11,5,3,6]
[11,5,4,10]
[11,6,9,8,7,10]
[8,1,7]
[10,4,1,7]

Figure 32: Vertice and Cell Arrays for the Voronoi Diagram in Figure 30

As discussed in the chapter on graph representations, the array C is an array of cycles in the Voronoi graph; in MATLAB, C is of a special data type, the vector cell array[19]. Each element of C contains the indices into V of the vertices of the corresponding Voronoi cell, in consecutive order, such that there is an edge between two consecutive vertices as well as between the first and last one.

If any index in a cell of the cell array is 1, then the corresponding Voronoi cell contains the first point in V, a point at infinity. This means the Voronoi cell is unbounded.

[19] MATLAB page on cell arrays: http://www.mathworks.com/help/techdoc/matlab_prog/br04bw6-98.html

Unbounded Voronoi cells create problems with the conditions for the threshold auto-adjustment program in Section 9. This can be overcome either by ignoring the respective cell (but loosing perhaps relevant information) or by using the image border as a limit (Section 9).

The result of using voronoin on the X in Figure 31 is shown in Figure 32.

It is interesting to note that in order to show the Voronoi diagram it is sufficient to call the voronoi function, while in order to retain any useful information it is necessary to use the voronoin function. I am sure there is the way to accurately plot the Voronoi diagram using the information from voronoin (thus avoiding recomputing the same information), but as of this version of the project, voronoin is used to gather useful information and voronoi is used to accurately plot the diagram.

7.3. DelaunayTri and triplot

The function to compute the Delaunay Triangulation that is used in this project has the form DT = DelaunayTri(x, y)[20]. It creates a Delaunay triangulation from a set of points, specified as column vectors, each column representing the x or y coordinates of the points.

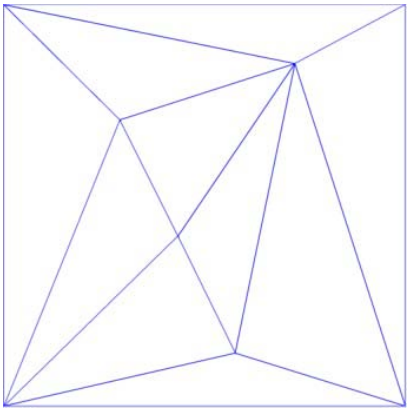


Figure 33: Delaunay Triangulation of the Points from Figure 30-left

[20] The DelaunayTri MATLAB reference is: http://www.mathworks.com/help/techdoc/ref/delaunaytri.html

The function used to accurately plot the computed triangulation is triplot(TRI,x,y)[21]. It displays the triangles defined in the m-by-3 matrix TRI. A row of TRI contains indices into the vectors x and y that define a single triangle. The default line color is blue.

The result in the running example from Figure 30 is shown in Figure 33.

7.4. The Quickhull Algorithm

MATLAB uses the quickhull algorithm (Qhull) to compute both Voronoi Diagrams and Delaunay Triangulations. The algorithm was first described in 1996 in (Barber, Dobkin, & Huhdanpaa, The Quickhull Algorithm for Convex Hulls, 1996)

The following is taken from the Qhull manual, in (Qhull.org, 1995).

The convex hull of a set of points is the smallest convex set that contains the points. Qhull computes the convex hull in 2D, 3D, 4D, and higher dimensions. The algorithm is similar to is similar to the randomized, incremental algorithms for convex hull and delaunay triangulation. If the set S is finite, the convex hull defines a matrix A and a vector b such that for all x in S, Ax+b <= [0,...].

Qhull represents a convex hull as a list of facets. Each facet has a set of vertices, a set of neighboring facets, and a half-space. A half-space is defined by a unit normal and an offset (i.e., a row of *A* and an element of *b*). Qhull accounts for round-off error. It returns "thick" facets defined by two parallel hyper-planes. The outer planes contain all input points. The inner planes exclude all output vertices.

[21] The triplot reference on the MATLAB site is: http://www.mathworks.com/help/techdoc/ref/triplot.html

Qhull may be used for the Delaunay triangulation or the Voronoi diagram of a set of points. It may also be used for the intersection of half-spaces.

8. AUTOMATIC THRESHOLD ADJUSTMENT

The original version of the program allowed the user to select the parameters manually, from a figure similar to the one in Figure 34. It soon became apparent however that in order for the desired seamless image duplicate detection, we needed the program to auto-adjust the threshold value when detecting the corner points. It was unexpected, however, that I couldn't find any good paper out there to do parameter adjustment the way I desired. So I decided to write my own.

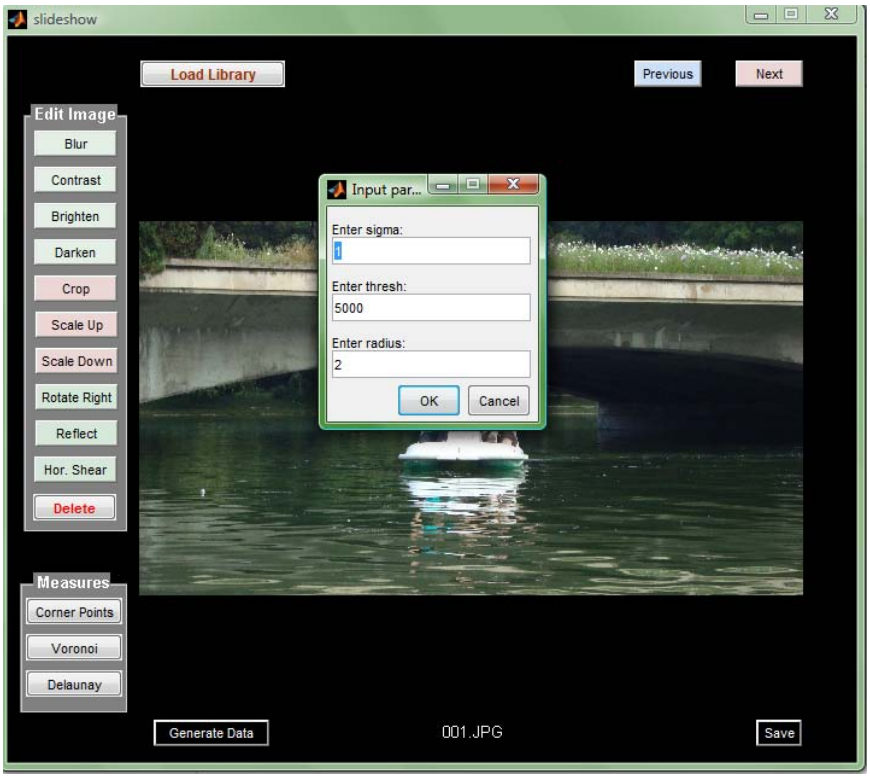


Figure 34: Harris Corner Points Parameter Selection Window

I chose to only modify the threshold, and keep the values of *sigma* (standard deviation of smoothing Gaussian) and *radius* (radius of region considered in nonmax suppression) to 1 and 2, respectively. Changing only the threshold should allow for as much flexibility as need, while simplifying the program.

8.1. Purpose

In order to match two near-duplicate images, we need to get two graphs that are as close to each other as possible, representing the images. By applying the same threshold on two images, one of them being the near-duplicate of the other, we can and will obtain different results, depending on what type of image modification was done. If the near-duplicate image was subjected to only geometric changes other than changes in scale since scale affects the number of corner points detected as long as the threshold value is constant (refer back to chapters 3 and 5), then the graph of the second image will most likely be a subgraph of the original graph, or a graph isomorphic to it. If the second image was subjected to (linear) photometric image changes though, there is more likely that the new graph would be a subgraph or a minor of the original graph (or the other way around). In this case then, a limit on what constitutes a subgraph or a minor should be placed, so as to limit the number of false positives obtained.

8.2. Preliminaries

The conditions the Voronoi diagram needs to meet in order for the threshold that generated the Harris Corner Points set that underlies the diagram are expressed below, in section 8.3. But first, I need to explain some preliminaries.

Let W be the width of the image, and H be the height. Then, the area of the image is $A = W \cdot H$. Let *numcells* be the number of cells in the Voronoi Diagram, computed based on the resulting Harris Corner Points generated using a set preliminary threshold, *thr*. Then *numcells* is also the number of Corner Points detected.

The threshold adjustment program is based on the areas of each cell, Ac_i, as related to the area of the image. Since *voronoin* works in such a way that the first row of vertex coordinates is

[*Inf Inf*], the cells that contain this first vertex have an infinite area, so in order for a more accurate area, considering the picture is finite, we could either disregard these areas or take the polygons relative to the margins of the picture. Right now, I am disregarding these areas by making them comply with the conditions in 8.3, no matter what their area relative to the image border is. In Figure 35 (a) for example, the cells on the margins are disregarded since they have an infinite area (the actual result is NaN); another option would be to calculate the area of the visible part of the cell.

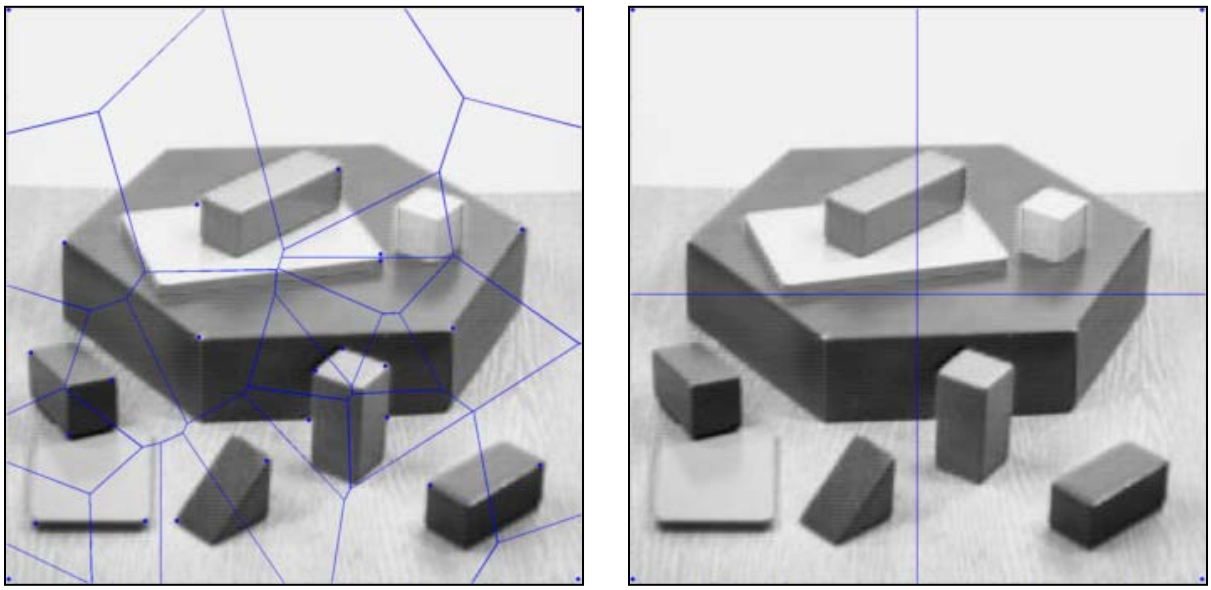


Figure 35: Voronoi Diagrams (a) with threshold 5,000; (b) with threshold 15,000

8.3. Conditions

Although it is not guaranteed to give the best possible threshold for an image, the auto-adjusting program as it is written right now will adapt depending on the image size and some characteristics of the Voronoi Diagram based on a preliminary set of Harris Corner Points, generated given a set threshold. Since we are starting with a small value for the threshold (*thr* = 100 initially) and increasing it from there, we know that we will keep the smallest value of the

threshold encountered that generates a diagram complying with the conditions. This means that a value for the threshold of *thr* = 5000 (Figure 35 (a)) will never be reached (though the optimal value of *thr*=2,500 won't be established either).

There are four conditions imposed on the cells of the Voronoi Diagram, as well as on the number of Harris Corner Points per image in order to auto-adjust threshold, as follows:

1. The biggest cell area has to be smaller than one fourth the area of the image, or $\max(Ac_i) < \frac{A}{4}$. In other words, there will be no cell in the image bigger than, or equal to, a quarter of the image. The diagram in Figure 35 (b) would violate this condition.
2. All cell areas have to be bigger than the area of the image over *numcells*^{2}, that is, $\mathrm{Ac_i} > \frac{A}{numcells^2}$. This sets a relative limit as to how small the small areas are allowed to be. Further limits apply though, as expressed below.
3. The sum of the areas that are smaller than the area of the image over *numcells* is less than the area of the image over *numcells*, that is, $\sum_i Ac_i < \frac{A}{numcells}$, for all i = {1, 2,…, *numcells*} where $Ac_i < \frac{A}{numcells}$.
4. The number of corner points detected, *numcells*, has to be at least one fifth of the square root of the area of the image, that is $numcells \geq \frac{\sqrt{A}}{5}$; this condition has to be met even if the ones above aren't met: the program stops execution once this condition is not met, regardless of whether or not one of the other conditions is met. This empirical number of points is based on the assumption that the image is in some sense significant.

8.3. Algorithm

```
function thr=adjust_thresh(img)

%img=imread('photos_in/022.jpg');
img=rgb2gray(img);

% get the width and height of the image, and compute the area
[w, h]=size(img);
A=w*h;

[thr meet]=adjust_thresh1(100, A, img);

while meet==1
    [thr meet]=adjust_thresh1(thr, A, img);
    %disp(thr);
end;

function [thr meet]=adjust_thresh1(thr, A, img)
% thr is the threshold, eventually adjusted at this step
% A is W*H
% meet is 1 if conditions are met, so we need to continue

% find the row and column coordinates of the Harris Corner Points
[~,r,c]=harris(img,1,thr,2);
x=[];
x(:,1)=c;
x(:,2)=r;

% find the coordinates of the vertices of the Voronoi diagram,
% as well as which vertices compose the cells of the diagram
[vert,cell]=voronoin(x,{'Qbb','Qz'});
% the number of cells
numcells=length(cell);

% the fraction of the image area
An=A/numcells;
%disp(['An=' num2str(An)]);

% free up some memory
clearvars x r c;

% the sum of all "small" areas
S=0;

% the maximum
Ac_max=0;

% meet is 1 if one of the conditions is met, and 0 if not
meet=0;

% meet == 1 when:
```

```
% * Ac_max >= A/4
% * Ac <= A/numcells^2
% * S >= A/numcells, where S = Sum[Ac | Ac < A/numcells]
% meet == 0, even if above conditions are met, if:
% * numcellss < sqrt(A)/5

for i=1:numcells
    % number of vertices in cell i
    len=length(cell{i});

    % XY is a matrix of the vertex coordinates
    XY=zeros(2,len);

    for j=1:len
        % the x coordinate of the j'th vertex of cell i
        XY(1,j)=vert(cell{i}(j),1);
        % the y coordinate of the j'th vertex of cell i
        XY(2,j)=vert(cell{i}(j),2);
    end;

    % calculate the area of the current cell
    Ac=polyarea(XY(1,:),XY(2,:));

    % if the area contains a vertex at infinity, ignore this cell
    if(isnan(Ac))
        Ac=An;
    end;

    %calculate the maximum
    if Ac>Ac_max
        Ac_max=Ac;
    end;

    if(Ac <= A/(numcells*numcells))
        meet=1; % if t
    else
        % calculate S
        if(Ac<An)
            S=S+Ac;
            %disp(['S=' num2str(S)]);
        end;
    end;
end;
if(Ac_max >= A/4) || (S >= An)
    meet=1;
end;
if (numcells<sqrt(A)/5)
    meet=0;
end;
if (meet==1)
    thr=thr+100;
end;
```

Code Snippet 10: The Threshold Auto-Adjust Function

The 3-channel image is first converted to grayscale and its area, *A*, is computed. The program starts with a preliminary threshold *thr* = 100, and tests if that is enough to meet the conditions. The conditions are met if *meet* == 0, and the program increases the threshold if *meet* == 1. The coordinates of the Harris Corner Points as well as the coordinates of the Voronoi vertices are computed; the number of corner points / Voronoi cells is retained in *numcells*. The variable *An* retains the value of the fraction *A/numcells.*

For every cell, the area of the cell, *Ac*, is computed; if the area is not a number (NaN), then it is assigned a value that would meet the conditions, *An.* If condition **(2)** from Section 8.2 is not met, then the conditions are not met: *meet* = 1. Otherwise, if the area of the current cell is smaller than *An*, it is added to the sum of the small areas. Similarly, if the current area is bigger than the maximum area until nou, *Ac_max*, the current area becomes the maximum.

After going through all the cells, conditions **(1)** and **(3)** from Section 8.2 are verified, and if neither one is met, then *meet* = 1. In the end, the last condition, **(4)**, is checked. If it is not met, then the current threshold is enough, so, *meet* = 0.

If the conditions are not met, then the threshold increases by 100, and the previous steps are repeated.

8.4. Problems

The program runtime is quadratic in the number of cells! There might be ways around this problem (a Divide and Conquer algorithm for starters), but I have not yet addressed them. The way around this that I worked on for presenting the project is running it all once, saving a text file with the name of the file and of the threshold, and using this file to lookup the appropriate threshold for each image. Of course such an approach takes a lot of time for the first

run of the program, and is highly unreliable since a user can manually change the name of the file to trick the system, but that is not the purpose of this project.

Images that add text or any sort of highly concentrated number of points are confusing for the system (see). That is, images that differ from the original image by adding text can be problematic. My first version of the program was highly confused by text added in the image, but the newer version deals better with it. Nevertheless, since the project only explores the types of image changes that were described in Section 3, this problem doesn't matter in light of the current research question. It is interesting to note however that someone who has knowledge regarding this issue can easily trick the program.

The program does not compute the perfect threshold. Better thresholds could be given to the program by a user. Also, the thresholds the program comes up with do not guarantee that the same points will be computed in near-duplicates, and thus does not conform to the windowing function in Section 4.6. Nevertheless, computing the threshold in this way allows for a finite, bounded dimension of the graphs generated.

Also, threshold values increase in steps of 100, so the threshold is not necessarily exact. A better approach would be to start decreasing the thresholds in smaller and smaller steps in order to coin the best value. Nevertheless, since the program cannot come up with the best possible threshold by design, there is no reason to slow it down even further.

9. ENCODING AN IMAGE AS A GRAPH: DELAUNAY TRIANGULATIONS

9.1. Methodology

The premise of the project is that images can be encoded as graphs, and that there are some benefits in encoding them as such. The reason to choose graphs as opposed to just corner points comes from the belief that people and machines should be able to perform the same task in a transparent way, and matching points is something that we think people find more complicated than matching graphs.

Since Harris Corner Points work fairly well in identifying some duplicate images, they have been chosen to be used as a basis for creating more suggestive graph-encodings of the images. The Voronoi Triangulation seemed like the natural choice, but it turned out to be too unstable, changing too easily when points were added or removed. The Delaunay Triangulation, although the dual graph of the Voronoi Diagram, proved to be more stable for the purposes of our project.

The algorithm that converts an image into a graph is fairly simple:

1. Given an image, we can compute the preliminary Harris Corner Points on it and use the threshold adjustment algorithm described earlier to auto-adjust the threshold to the most appropriate value;
2. Given the new threshold, we can recompute the best Harris Corner Points on the image;
3. Given the locations of the Harris Corner Points, we can use the DelaunayTri MATLAB function to compute the Delaunay Triangulation, and save the result as the

graphical encoding of the image – this graph can be weighted or unweighted: for now, we are saving it as an unweighted graph.

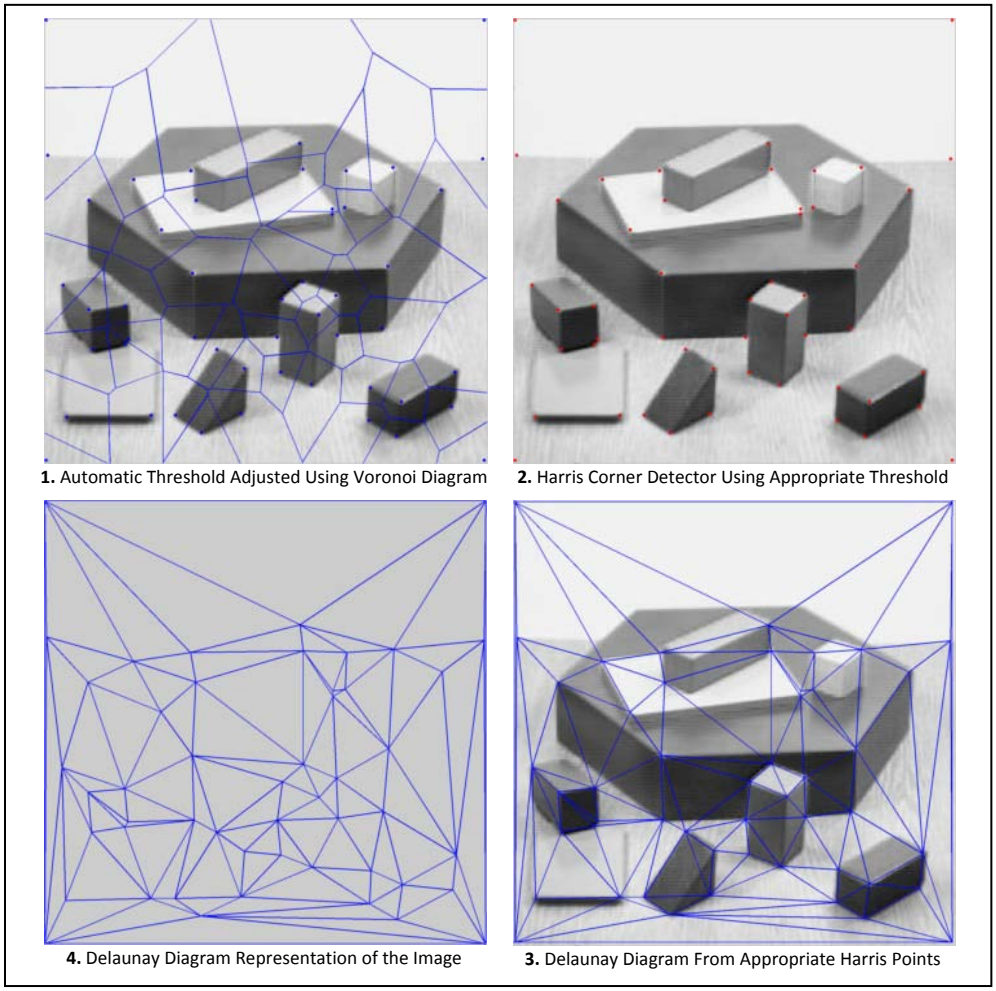


Figure 36: Steps in Obtaining a Graph from an Image

9.2. Encoding Code

```
[cim,r,c]=harris(img,1,thr,2);
dt=DelaunayTri(c,r);
triplot(dt);
```

Code Snippet 11: Delaunay Image Encoding

The row and column coordinates of the Harris Corner Points are retrieved, using the auto-adjusted threshold. The Delaunay Triangulation is computed and displayed.

9.3. Problems with Current Approach. Future Plans

Encoding an image as a Delaunay Triangulation is just like hashing: two images can hash to very similar entries, but that won't occur very often. I have not investigated what the likelihood of two different images being encoded by similar (in the sense this project assumes similarity) diagram is.

Assuming the pictures are taken by different people, with different cameras and in different positions, and that the pictures are of various natural scenes, then the likelihood of having two pictures of completely different scenes encode to similar diagrams is very low.

This likelihood would, most likely, go even lower if the graphs that we are comparing were in fact weighted, with the weight representing the distance between the two respective vertices that represent the endpoints of the edge.

As part of my possible future plans (see Section 10.2), I am planning to conduct a study on how people decide what diagrams are related or not. In such a case, it is almost certain that the diagram 4 from Figure 36 is a better representation of the original image in diagram 1 than the diagram in Figure 37, although they are both diagrams of the same image.

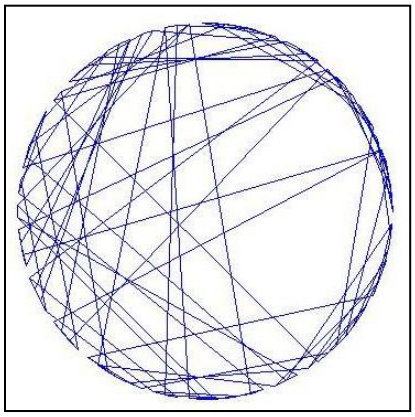


Figure 37: Other Representation of the Delaunay Graph

In other words, the placement of vertices, as well as the distances between them, most likely matter for human observers; whether or not they should be taken into account when writing a program that finds near-duplicates is another interesting research question.

Right now, rather than deal with graphs that maintain the properties of the Delaunay Diagram, the program only maintains the vertices and connections between them and disregards all other information.

So, for the program right now, Figure 37 and diagram 4 of Figure 36 are the same graph.

10. RESULTS. CONCLUSIONS. FUTURE WORK

10.1. Results

The following pages show the images, corner points, Voronoi Diagram and Delaunay Triangulations for all the considered transformations of an image, including the original image.

The first image in the group (Figure 38, Figure 42, Figure 46, Figure 50, Figure 54, Figure 58, Figure 66, Figure 70, Figure 74 and Figure 78) is the original image we are working with, for every group of pictures.

The second image in each group (Figure 39, Figure 40, Figure 44, Figure 48, Figure 52, Figure 64, Figure 56, Figure 60, Figure 68, Figure 72, Figure 76 and Figure 80) proves that the automatically detected threshold value produces acceptable corner points, for all images, regardless of modifications.

The third image of the group (Figure 40, Figure 44, Figure 48, Figure 52, Figure 64, Figure 56, Figure 60, Figure 68, Figure 72, Figure 76, Figure 80) shows the Voronoi Diagram for the image, given the threshold. It shows, in a graphical way, how a diagram should look like in order for all the conditions for the threshold auto-adjustment to be met (Section 8.3).

The fourth image of the group (Figure 41, Figure 45, Figure 49, Figure 53, Figure 65, Figure 57, Figure 61, Figure 69, Figure 73, Figure 77 and Figure 81) shows the Delaunay Diagram for the image, given the threshold. These are the graphical representations that should be compared in order to see if an image is a near-duplicate of the other.

For the original image (Figure 38), the threshold value obtained is an appropriate value, as can be seen from the obtained corner points in Figure 39. As can be seen in Figure 41, the

Delaunay Diagram disregards parts of the image that aren't interesting. The corner points that are on the edge of the image should be disregarded in future versions.

- Image "003.jpg"; auto-threshold value: 17,500

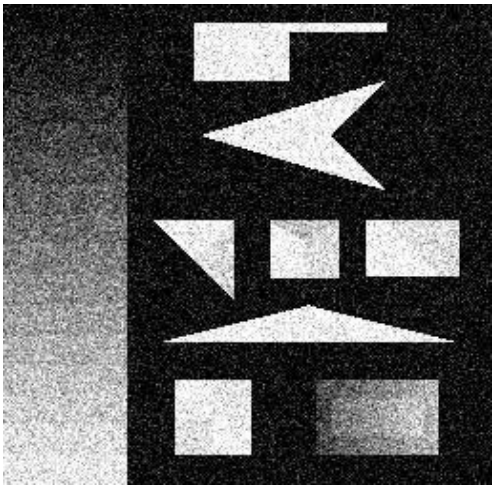


Figure 38: Original Image

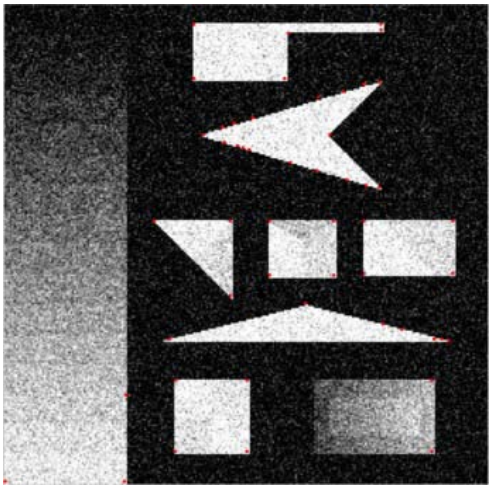


Figure 39: Corner Points

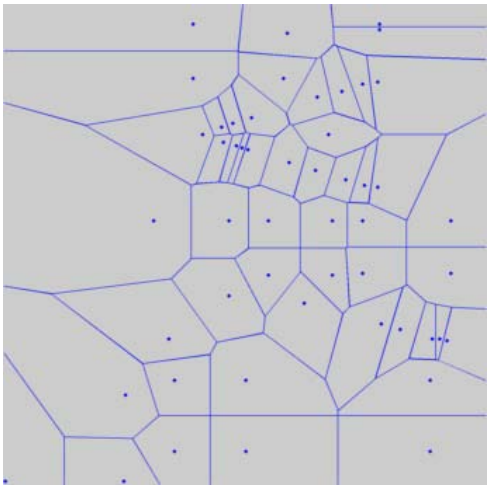


Figure 40: Voronoi Diagram

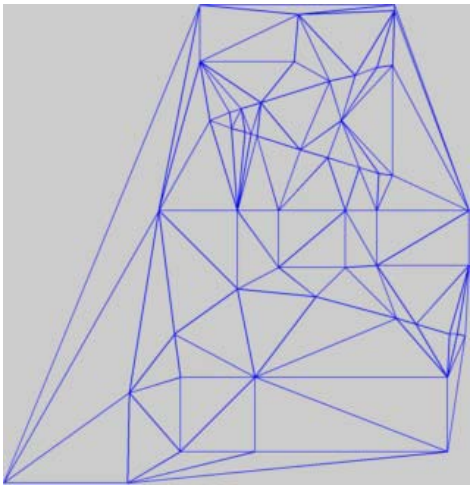


Figure 41: Delaunay Triangulation

The same is true about the threshold value and the Delaunay Diagram for the auto-contrasted (Figure 42), blurred (Figure 46), brighter (Figure 50), darker (Figure 54) and larger (Figure 58) images. One can see the similarity between the Delaunay Diagrams for all of these

images, and the working assumption is that these graphs are either subgraphs or minors of one another.

- Image “003_autocontrast.jpg”; auto-threshold value: 15,300

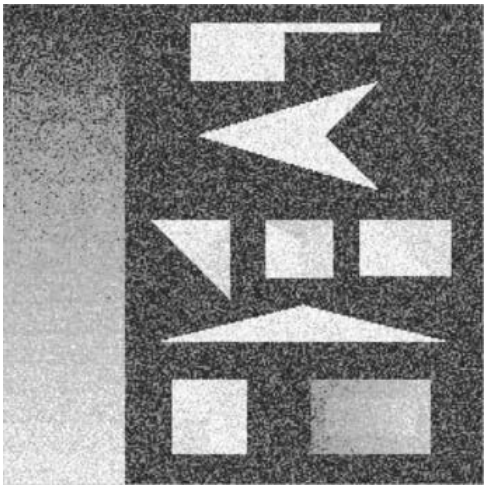


Figure 42: Auto-Contrast Image

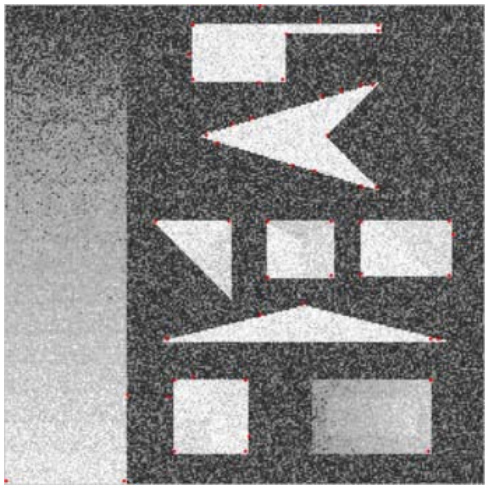


Figure 43: Corner Points

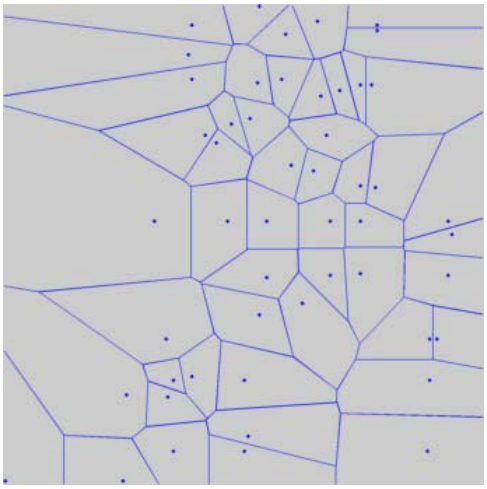


Figure 44: Voronoi Diagram

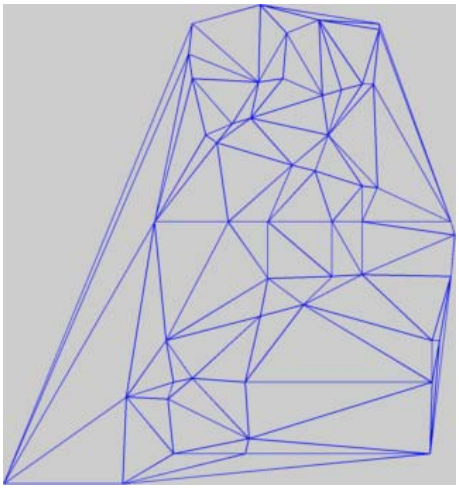


Figure 45: Delaunay Triangulation

- Image “003_blur.jpg”; auto-threshold value: 300

Figure 46: Blurred Image

Figure 47: Corner Points

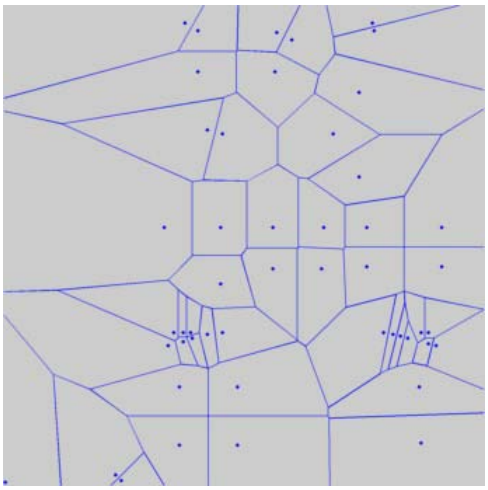


Figure 48: Voronoi Diagram

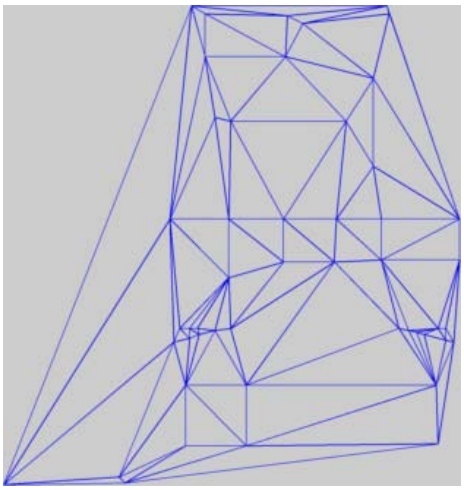


Figure 49: Delaunay Triangulation

- Image “003_brighter.jpg”; auto-threshold value: 17,500

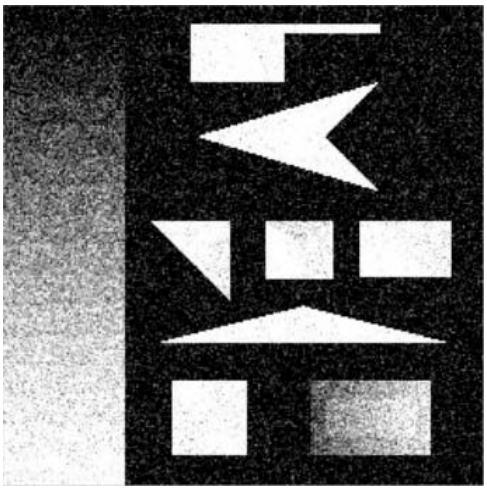


Figure 50: Brighter Image

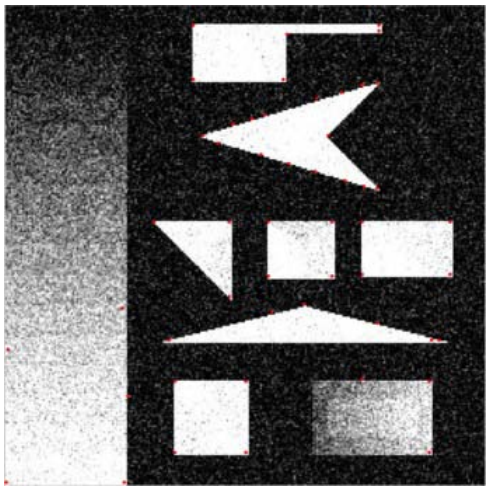


Figure 51: Corner Points

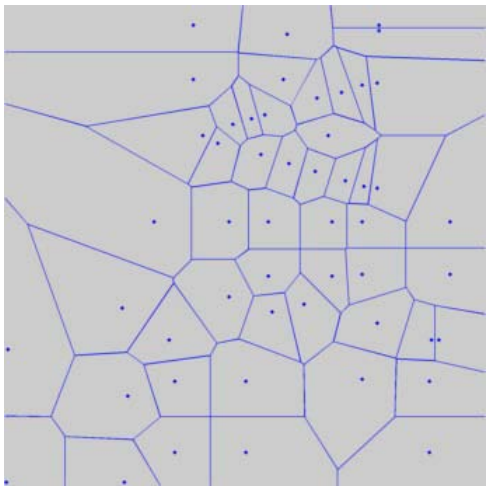


Figure 52: Voronoi Diagram

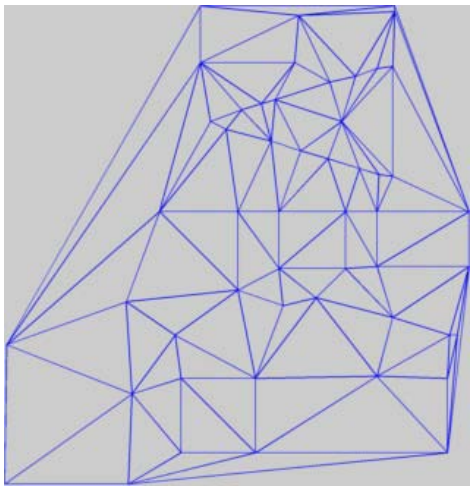


Figure 53: Delaunay Triangulation

- Image “003_darker.jpg”; auto-threshold value: 19,100

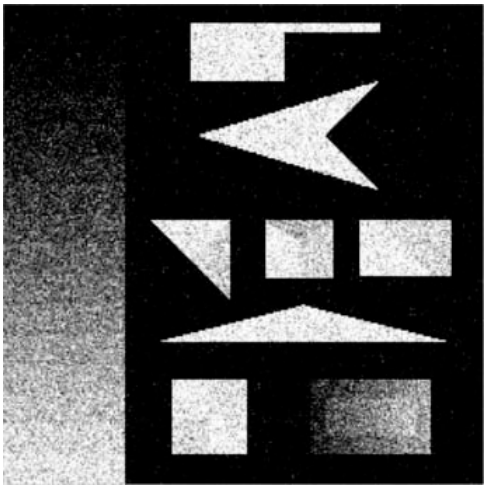


Figure 54: Darker Image

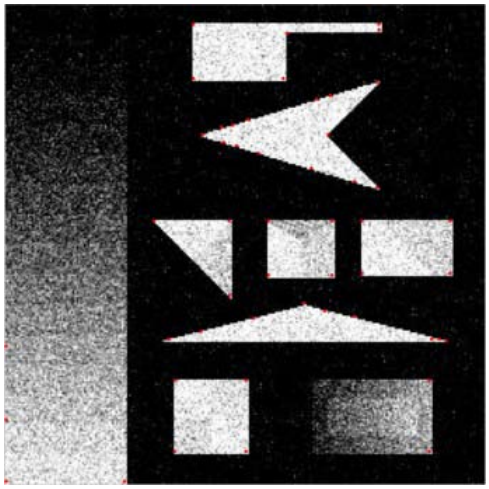


Figure 55: Corner Points

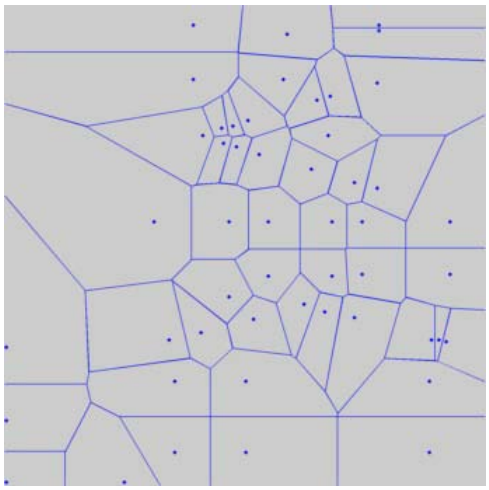


Figure 56: Voronoi Diagram

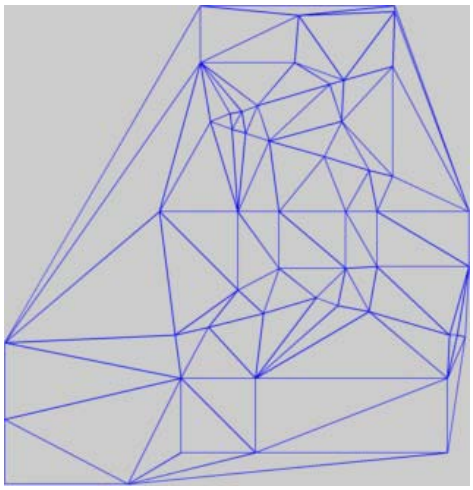


Figure 57: Delaunay Triangulation

- Image “003_larger.jpg”; auto-threshold value: 15,100

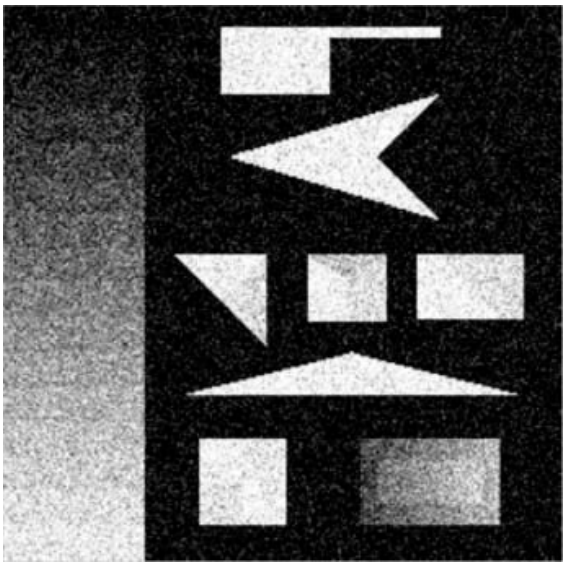


Figure 58: Larger Image

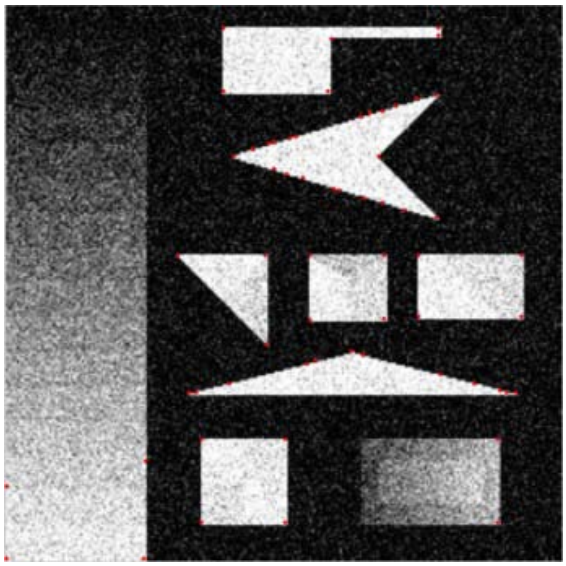


Figure 59: Corner Points

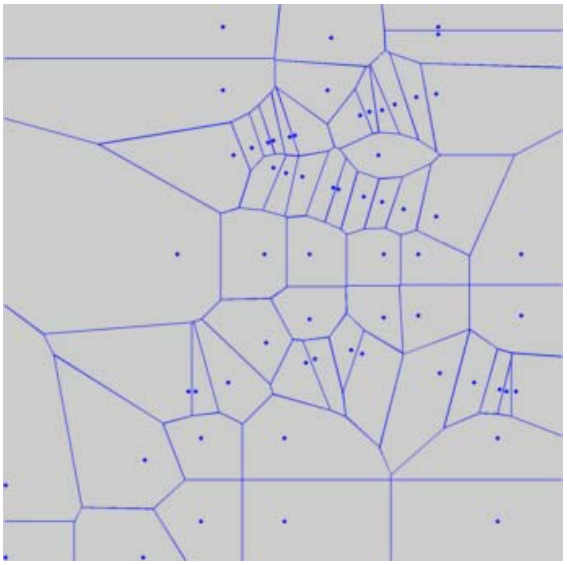


Figure 60: Voronoi Diagram

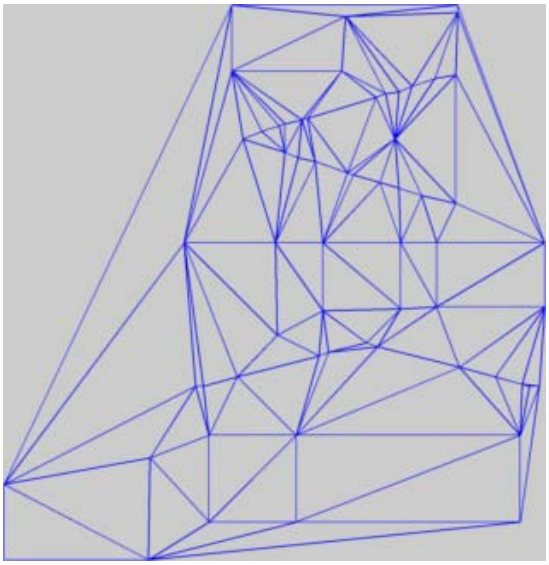


Figure 61: Delaunay Triangulation

The threshold value for the cropped image (Figure 62) is also appropriate. The Delaunay Diagram seems to be a subgraph of that of the original image's.

- Image "003_crop.jpg"; auto-threshold value: 18,200

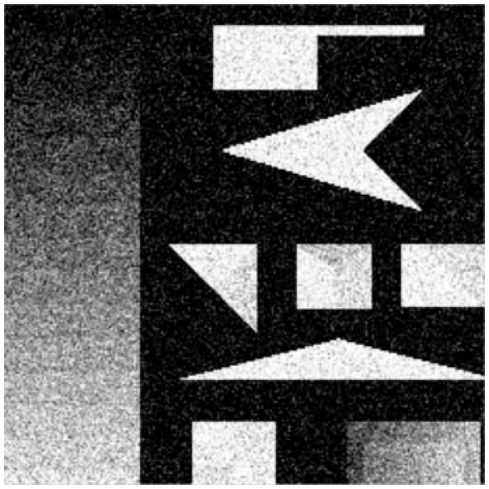


Figure 62: Cropped Image

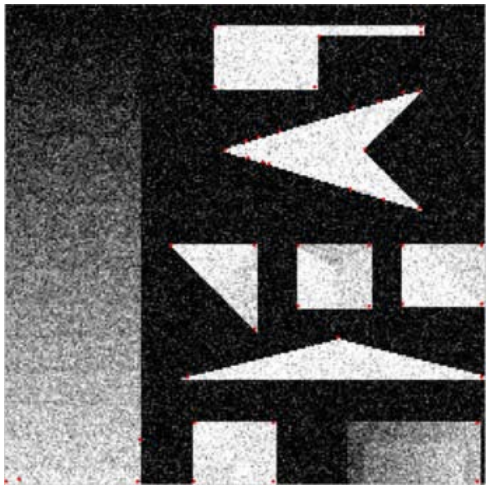


Figure 63: Corner Points

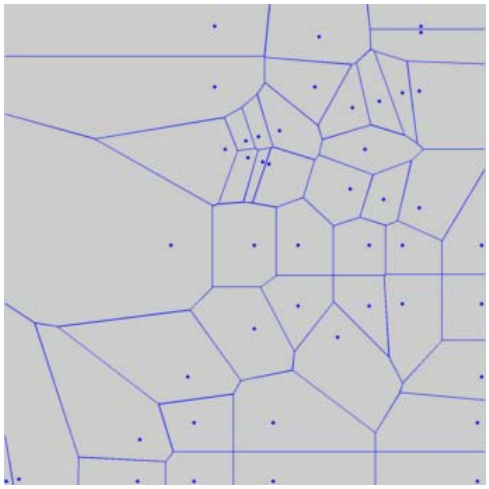


Figure 64: Voronoi Diagram

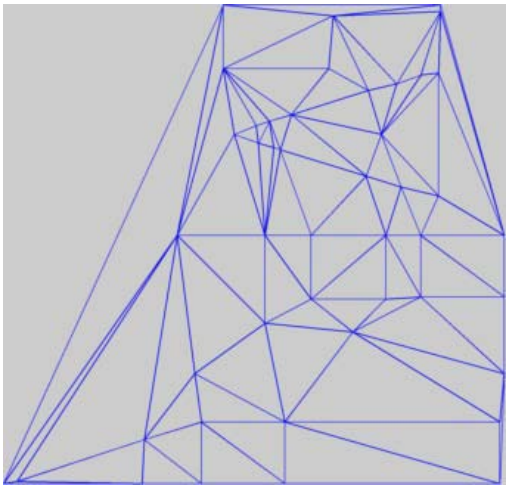


Figure 65: Delaunay Triangulation

A reflection mirrors the image, horizontally. The threshold of the reflected image (Figure 66) also produces appropriate corner points. The graph of the reflected image is the same as the reflected graph of the original image, or a subgraph of it.

- Image “003_ reflected.jpg”; auto-threshold value: 17,500

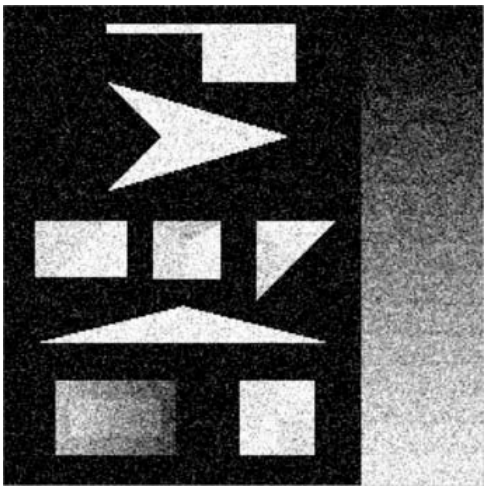


Figure 66: Reflected Image

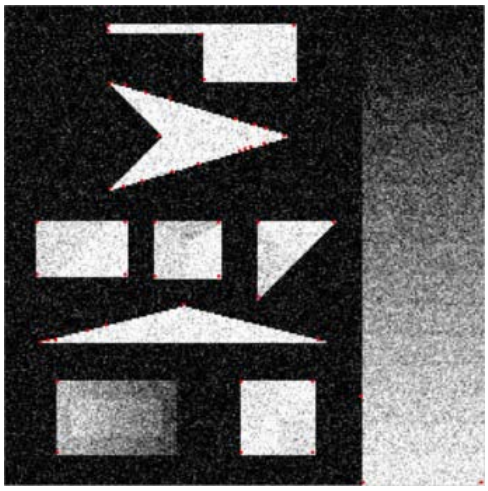


Figure 67: Corner Points

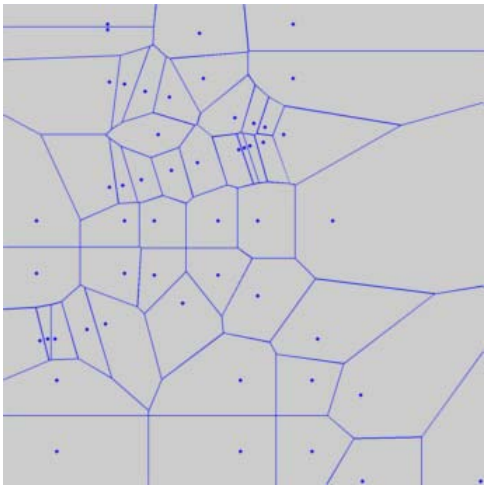


Figure 68: Voronoi Diagram

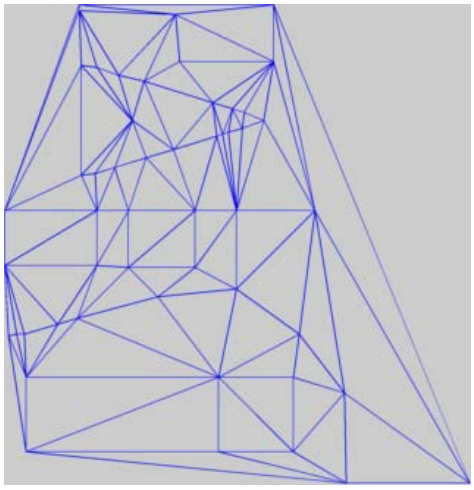


Figure 69: Delaunay Triangulation

A 90^{o} rotation to the right produces a threshold that is not a little bit lower than the threshold of the original image. The threshold of the rotated image (Figure 70) also produces appropriate corner points. Since the threshold differs by 100, the graphs are likely not isomorphic; they are either a subgraph or a minor of one another.

- Image “003_ rotated.jpg”; auto-threshold value: 17,400

Figure 70: Rotated Image

Figure 71: Corner Points

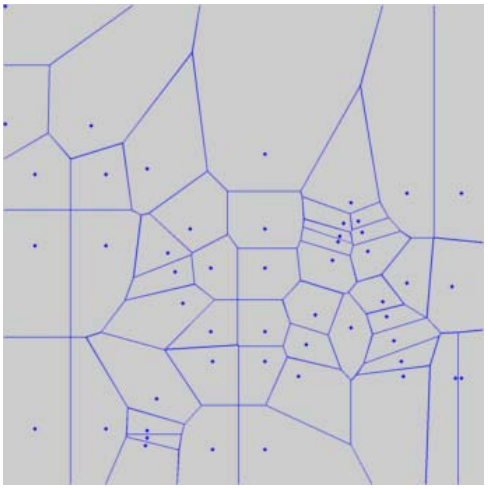


Figure 72: Voronoi Diagram

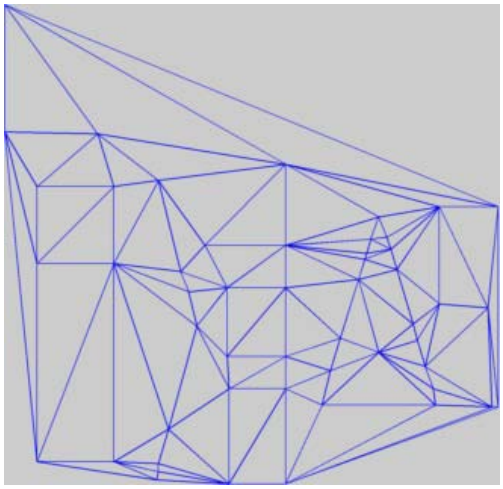


Figure 73: Delaunay Triangulation

Shearing produces some artificial corner points, as well as changes the distance between points on the image. The threshold for the sheared image (Figure 74) is, again, appropriate. The Voronoi graph differs completely because of the artificial points, but the Delaunay graph is again, similar for the naked eye.

- Image “003_sheared.jpg”; auto-threshold value: 11,800

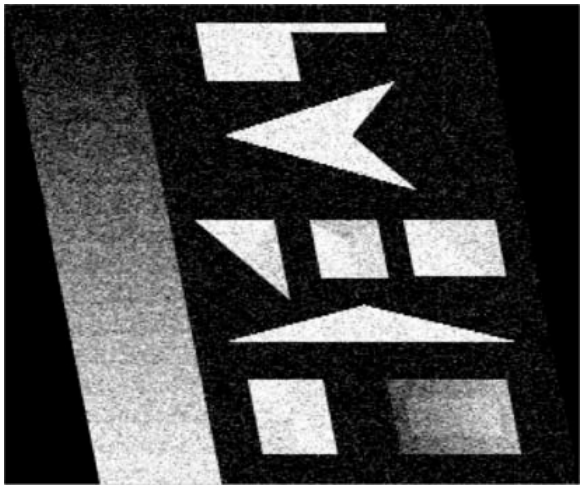


Figure 74: Horizontally Sheared Image

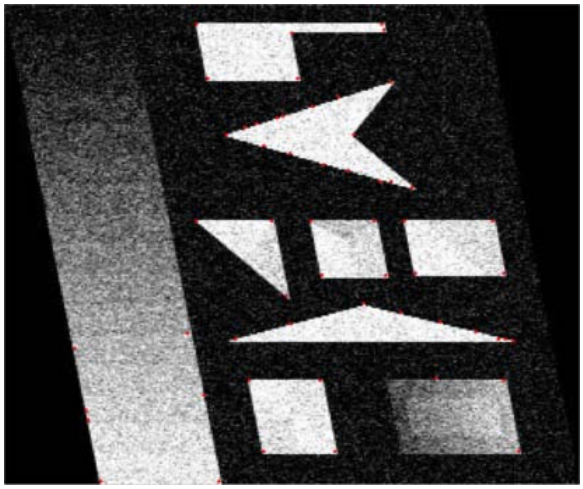


Figure 75: Corner Points

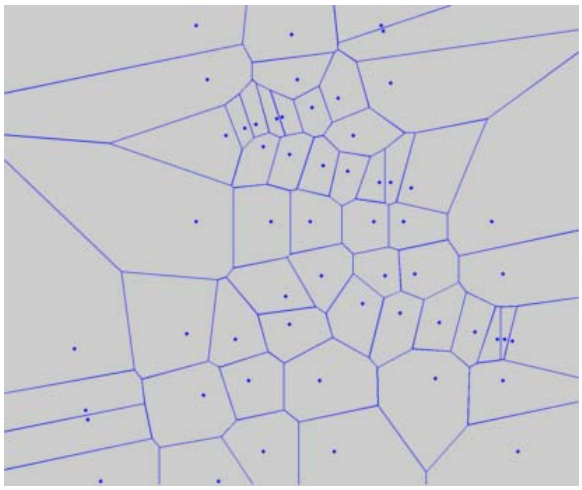


Figure 76: Voronoi Diagram

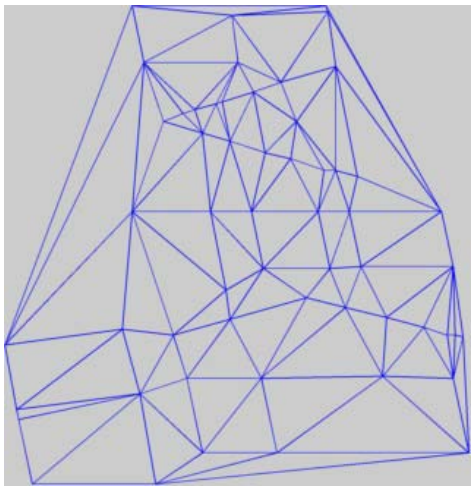


Figure 77: Delaunay Triangulation

The only image transformation that creates problems in threshold auto-detection is the smaller image (Figure 78). The threshold value is too big, and the number of corner points detected is too small compared to the corner points for the other transformations. As a result, the Delaunay Triangulation has a lot less points. Nevertheless, the graph is clearly related to the others.

- Image “003_smaller.jpg”; auto-threshold value: 45,700

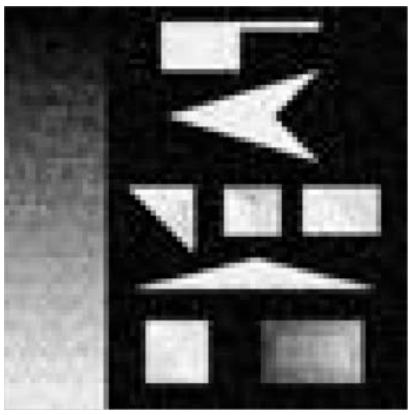


Figure 78: Smaller Image

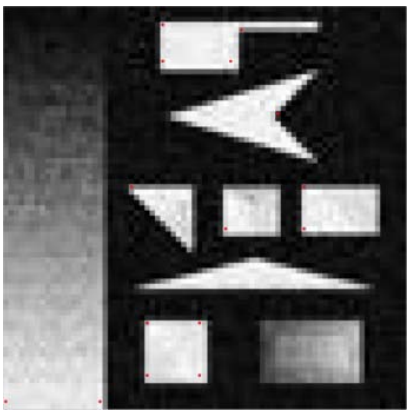


Figure 79: Corner Points

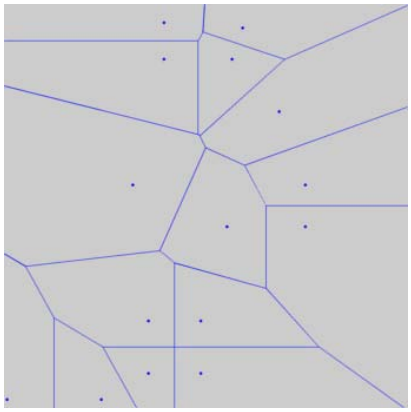


Figure 80: Voronoi Diagram

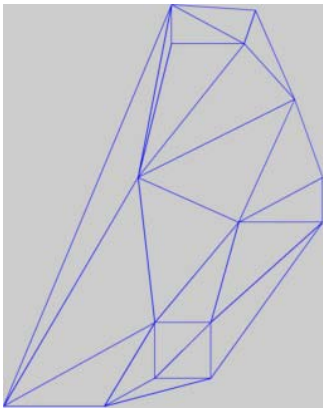


Figure 81: Delaunay Triangulation

10.2. Conclusions and Future Work

Some claims made throughout this paper have not been thoroughly investigated, mainly because of the short duration of the project (roughly 7 months), that included the time to come up with a research question, the time to do background reading on the material already existent (Darya Frolova, March 2004) as well as the time for finding a niche that hasn't been explored before. Although Voronoi Diagrams have been explored as image segmentation techniques before (Arbelaez & Cohen, 2003), (Sinclair, 1999), (Cheddad, Mohamad, & Manaf, 2008)], a threshold auto-adjustment algorithm based on features of the image Voronoi Diagram has not been previously explored, at least not from what I found. Nether have Delaunay Triangulations been used to encode image characteristics for duplicate detection based on graph similarity.

Referring back to the definitions in Chapters 5 and 6, there are some similarities we can infer between graphs of image near-duplicates. The similarities have been referenced throughout the text, and I will summarize them here.

- Identical graphs: two Delaunay graphs of the same image would be identical, as long as the threshold values are identical, both in number of vertices, edge connections and in vertex labeling; if we move to weighted graphs, the graph weights will also be the same.
- In a perfect world, in which the threshold detected would be exactly the same, the graphs of scaled and sheared images would also be identical to the original graph if the graphs are not weighted; for changes in image size the graph weights would be proportional, and for horizontal shear they would be mathematically computable.
- Again, in a perfect world, the graphs of rotated and reflected images would be identical to the original image's graph, but with different vertex labeling, that is isomorphic.

- The graphs of cropped images would be subgraphs of the original image graph.
- The graphs of images obtained by blurring, contrast auto-adjustment, and changing luminosity will consist of either induced subgraphs or of minors of the original graph.
- Other measures that can be used to measure similarity are chromatic numbers and the graph diameter.

Nevertheless, we are not dealing with a perfect world: the number of corner points differs between an image and its rotated counterpart for example, even if the same threshold is used. In such a case, the only inference we can make is that the graphs of near-duplicates are either (induced) subgraphs or minors.

I didn't have enough time to test this hypothesis, and there are also some problems that were not foreseen, in the sense that performing the similarity search on graphs is not possible in MATLAB, and some of the calculations are even NP-complete. A way around the first problem would be exporting the resulting graphs in a format readable by Wolfram Mathematica and importing them there in order to use that software to perform the similarity comparison. Also, since we are dealing with finite graphs that have a relatively small number of vertices, a brute-force approach to solving those problems can be taken.

Another problem with the current idea is the fact that we are considering the graphs unweighted, losing important information as a result. An alternate approach would be to retain the distance between vertices as the edge weight, and export the weighted graph to Mathematica. Then again, the notions of minors and induced subgraphs were not defined for weighted graphs, and such a specification will need to be made.

Another important issue that necessitates work is improving the run-time of the threshold auto-adjustment method. There are various ways to make a quadratic algorithm faster, and I will look into them.

An interesting future step in the development of the project would be to design a study in which people are presented with Delaunay Diagrams, such as the ones in Figure 41 and Figure 45 and asked to say whether the two graphs are encodings of near-duplicates. When presented with graphs of at least a hundred images, it might become apparent what criteria people use to classify those graphs. Also, an interesting improvement upon the experiment might ask the subjects to identify which graph belongs to which transformation, and see if any useful conclusions could be drawn from there.

In conclusion, I believe that, during this two semester-long project, I was able to come up with an interesting idea, that has the potential to work and also offers a new way to view images, as well as made the first steps towards its implementation. I believe that these kinds of Delaunay graph representations of images can be useful when searching for near duplicates, especially if the image by which you are actually searching is not-to-be-disclosed to the owners of the database.

Bibliography

[5] Darya Frolova, D. S. (March 2004). *Matching with Invariant Features.* The Weizmann Institute of Science.

[12] Bondy, J. A., & Murty, U. S. (1976). *Graph theory with applications.* New York: Elsevier Science Publishing Co.

[14] Aurenhammer, F. (n.d.). Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure. 345.

[20] Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. T. (1996). The Quickhull Algorithm for Convex Hulls. *ACM Trans. Mathematical Software , 22*, 469-483.

[22] Du, Q., Gunzburger, M., Ju, L., & Wang, X. (2006). Centroidal Voronoi Tessellation Algorithms for Image Compression, Segmentation, and Multichannel Restoration. *Journal of Mathematical Imaging and Vision* , 177 - 194.

[23] Du, Q., Faber, V., & Gunzburger, M. (1999). Centroidal Voronoi Tessellations: Applications and Algotithms. *SIAM REVIEW , 41* (4), 637-676.

[24] Nicu, S., Theo, G., Joost van de, W., & Sietse, D. (n.d.). Corner detectors for affine invariant salient regions: Is color important?

[25] Du, Q., & Wang, D. (2005). The Optimal Centroidal Voronoi Tessellations and the Gersho's Conjecture in the Three-Dimensional Space. *Computers and Mathematics with Applications , 49*, 1355–1373.

[27] William, J., & Hirschfeld, P. (2001). *Surveys in combinatorics.* Cambridge, UK: Cambridge University Press.

Ahn, S. H. (2005). *Convolution*. Retrieved April 11, 2011, from http://www.songho.ca/dsp/convolution/convolution.html

Arbelaez, P. A., & Cohen, L. D. (2003). Generalized Voronoi Tessellations for Vector-Valued Image Segmentation. *Proc. 2nd IEEE Workshop on Variational, Geometric and Level Set Methods in Computer Vision (VLSM'03)* (pp. 49-56). Nice, France: IEEE.

Aurenhammer, F., & Klein, R. (2000). *Handbook of Computational Geometry* (Vol. Ch. 5: Voronoi Diagrams). (J.-R. Sack, & J. Urrutia, Eds.) Amsterdam, Netherlands: North-Holland.

Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. T. (1996). The Quickhull Algorithm for Convex Hulls. *ACM Trans. Mathematical Software , 22*, 469-483.

Cheddad, A., Mohamad, D., & Manaf, A. A. (2008). Exploiting Voronoi diagram properties in face segmentation and feature extraction. *Pattern Recognition , 41*, 3842-3859.

Dirichlet., P. G. (1850). Uber die Reduction der positiven quadratischen Formen mit drei unbestimmten ganzen Zalhen. *J. Reine Angew. Mathematik , 40*, 209–227.

Du, Q., Faber, V., & Gunzburger, M. (1999). Centroidal Voronoi Tessellations: Applications and Algotithms. *SIAM REVIEW , 41* (4), 637-676.

Foo, J. J., Zobe, J., & Sinha, R. (2007). Clustering Near-Duplicate Images in Large Collections. *Proceedings of the international workshop on Workshop on multimedia information retrieval* , 21-30.

Fukuda, K. (2004, June 18). *Frequently Asked Questions in Polyhedral Computation*. Retrieved 04 17, 2011, from http://www.ifor.math.ethz.ch/~fukuda/polyfaq/polyfaq.html

Harris, C., & M., S. (1988). A Combined Corner and Edge Detector. *Proceedings of the 4th Alvey Vision Conference* , 147-151.

MathWorks. (1984-2011). *MATLAB - Documentation*. Retrieved April 11, 2011, from http://www.mathworks.com/help/techdoc/

Moravec, H. (1980). *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* Pittsburgh: Carnegie-Mellon University, Robotics Institute.

Moravec, H. (1979). Visual mapping by a robot rover. *Int. Joint Conf. on Artif. Intell.* , 598–600.

Noble, A. (1989). *Descriptions of Image Surfaces.* Oxford University: Department of Engineering Science.

Okabe, A., Boots, B., Sugihara, K., & Chiu., S. N. (2000). *Spatial Tesselations: Concepts and Applications of Voronoi Diagrams, 2nd ed.* Wiley.

Pless, R. (2003). *Lecture 16: Voronoi Diagrams and Fortune's Algorithm; Lecture 17: Voronoi Diagrams and Delauney Triangulations*. Retrieved April 22, 2011, from http://research.engineering.wustl.edu/~pless/506/l16.html

Qhull.org. (1995, May 17). *Qhull code for Convex Hull, Delaunay Triangulation, Voronoi Diagram, and Halfspace Intersection about a Point*. Retrieved April 22, 2011, from http://www.qhull.org/

Robertson, N., Sanders, D. P., Seymour, P. D., & Thomas, R. (1996). A New Proof of the Four Colour Theorem. *Electron. Res. Announc. Amer. Math. Soc. 2* , 17-25.

Rosenlicht, M. (1985). *Introduction to Analysis.* Dover Publications.

Sinclair, D. (1999). *Voronoi seeded colour image segmentation.* Cambridge, UK: AT&T Laboratories.

Voronoi, G. M. (1908). Nouvelles applications des parametres continus a la theorie des formes quadratiques. deuxieme memoire: Recherches sur les paralleloedres primitifs. *J. Reine Angew. Mathematik , 134*, 198-287.

West, D. B. (1996, 2001). *Introduction to Graph Theory - Second Edition.* Prentice Hall.

Wolfram Research. (2011, 04 22). *Wolfram MathWorld*. Retrieved 04 22, 2011, from http://mathworld.wolfram.com/

TABLE OF FIGURES

TABLE OF CODE SNIPPETS