**Applications of Delaunay Triangulation Algorithms and Edge Flipping in Facial Recognition**

*(subtitle: Make Yourself Cubist™)*

Nika Korchok

There is a brief moment in the interim, between pressing the screen and when a cascade of filter options appear, when the Snapchat screen makes itself vulnerable — or seemingly so. Perhaps the algorithm (point location, triangulation, alpha mask generation, etc.) is simply not fast enough to go unnoticed. Or perhaps, in hinting a glimpse at your triangular decomposition, a user feels that tantalizing endorphin-surging rush of peeking behind the curtain as the Wizard soliloquizes about the mysteries of the “cloud.”

(<https://petapixel.com/2016/06/30/snapchats-powerful-facial-recognition-technology-works/>)

Or perhaps, like most users, you never noticed the triangular mesh on your hurried way to transfigure yourself into some canonical dog-eared millennial archetype. That’s about to change.

Why?

We delineate spaces without thinking. This is a way to think about how we create referential landmarks to ourselves and to other points. This is an exploration of space, divisions of space and the landmarks we forget to notice on the landscapes that we most frequently apotheosize and excoriate concurrently — our own faces.

**1. Refinement using Gaussian filter**

Understanding the process of triangulating a selected face region (i.e. what we need so we can put a cutesy infantilizing dog filter on it!) requires understanding how a computer understands a photograph.

A pixel - unit of measurement - a square on your screen - the number of which per inch (the correct usage of “ppi” not “dpi” will earn you bonus points with the 𝜋 people who care and put you ~3𝜎 away from the fickle favors of the general population, but now you know) can reveal more and more ugly details of any given image.

A single pixel contains a wealth of knowledge that is both bountiful and generally rather accessible; there are no offshore Cayman Islands accounts for pixel info — not yet, anyways.

Image pixels are stored as one-dimensional arrays. While we view an image as having two dimensions (Euclidean x,y coordinates or “height” and “width”), the pixels are stored in one contiguous strip. A pixel at location (x,y) in a picture is indexed in the pixel array at [x+ y\*width]. This is good for space and time efficiency — indexing a pixel is O(1), memory allocation is slightly cleaner — but annoying for figuring out where the fuck pixel(x,y) actually is in an array that is one-dimensional.  
  
Essentially, this is just some annoying math. Numbers are hard.

The input image that is generally fed to the filtering framework used by Snapchat and other similar products of image-filtering/manipulation, is complex. The sheer wealth of information available in pixel data is useful for performing complex operations. It is not useful when trying to distill an image into a basic framework — a skeletal structure if you will (not really the best term to use here but “skeletonization” makes an appearance in section 3 and we’re just trying to evoke some Egon Schiele conceptualizations of the emaciated body of an image so hold tight)

A myriad of factors can make an image salt-and-peppered with noise, which comes through generally. The input image is convoluted. So we are going to convolute it.  
  
(Note: Computational geometry is basically Sylvia Plath with shapes; lots of moody introspective analysis of interior layers, slant rhyme, and self-destructive decomposition of those layers.)

**Convolution**

Take a picture. Put another over it. This is like every problematic makeover scene in every teen movie that defined my early years before I found Hal Ashby and Agnès Varda: whatever veneer you choose to smear over the face of the original image will make it unequivocally better than its previous iteration. Did you know that self-hate is a native data structure to the JPEG??  
  
We’re really never going to get to making shapes.

Remember how I said that the image is made of pixels that are not really divided in Euclidean terms but in those of a one-dimensional array?  
  
We’re going to throw that out the window for a Euclidean moment (it’s a vector joke. What’s the magnitude/direction of that fuck we’re throwing out the window? -infinity.)

Pretend that we are looking at a really cheesy window from a Sr.Kindergarten art class — refined enough to not just be a box, but not really elevated enough to understand that most windows aren’t just tacky divisions of four equal, Chicklet pieces of glass stuck together by an X, tilted 45 degrees.

The window is a 2x2 matrix. The JPEG IMAGE WINDOW \*WOW\* is also a matrix for these purposes. A convolution is like putting a piece of stained glass over a window pane — it does not technically alter the original data of the image, but it does alter our view — we “convolve” an image with a convolution matrix, by passing a small amount of window paint across the //

window pane

(Sorry// I know this is pain

full of so many puns.)

Once you’ve painted the window with this convolution matrix, the image will look different. A swipe of lipstick and sad.jpeg becomes missCongeniality.jpeg to our program’s eyes.  
  
Ceci n’est pas une fenêtre? Remember, we’re not looking at something real — we’re looking at a representation of something real. It’s not really your face on the screen, it’s a representation of your face on the screen that you have been fooled to believe is truth; your face is actually just malleable numbers organized into a straight line.

ii.

Next, having calculated the average brightness of each pixel using a convolution matrix, we can normalize these values with a Gaussian filter.

What?

Essentially a reduction in noise. More formally, setting value of pixel to a value that is the normalized, weighted average of the pixels in a set neighbourhood around that pixel. We are making the picture less crisp, less defined, less interesting — perhaps an Orwellian convolution of a pixel via a given matrix such that a flatter and more complacent range of values correlating to thinkspeak results.

Gaussian filtering opens the doors for other operations — Canny Edge detection, corner detection, and greater accuracy for finding facial landmark points.

http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

<https://www.researchgate.net/publication/220646162_Face_recognition_using_Histograms_of_Oriented_Gradients>

<http://www.learnopencv.com/histogram-of-oriented-gradients/>

<https://www.learnopencv.com/image-recognition-and-object-detection-part1/>

<http://setosa.io/ev/image-kernels/>

<http://www.cse.cuhk.edu.hk/leojia/papers/portrait_eg16.pdf>

https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering\_1up.pdf

<https://www.geometrictools.com/Documentation/Skeletons.pdf>

**2. Face Detection**

Face detection in an image essentially works in two parts here: (I) detection of the “blob,” or “contour” or “shape” of what is recognized as a face; (II) detection of the landmark points that make up a face — i.e. eyes, nose, mouth, jawline.

For ease and simplicity, I have used the OpenCV module for Face Detection to create a bounding box from which to work for the remainder of this visualization.

<https://github.com/atduskgreg/opencv-processing-book/blob/master/book/tracking/face_detection.md>

<http://www.magicandlove.com/blog/2013/04/05/the-new-face-detection-and-processing/>

**3. Point Location**

The first step is simply recognizing a face in a given input image.

a) Create alpha mask to determine threshold — separate face from background  
<http://homepages.inf.ed.ac.uk/rbf/HIPR2/threshld.htm>

b) Use convolution matrix of Sobel Edge Detection algorithm to “smooth” image; poetically, this is the blurring of lines that separate our bodies from the space that surrounds them; algorithmically, this is a way to quantize pixel brightness values to refine the pool of potential landmarks while still retaining the majority of original “edges” in an image.

c) Use Shi-Tomasi Corner Detection - using the stack of detected edge pixels, we can use this smaller data set to determine “corners” — which we will then then refine to interest points.

<http://www.ai.mit.edu/courses/6.891/handouts/shi94good.pdf>

<http://www.cse.psu.edu/~rtc12/CSE486/lecture06.pdf>

<https://classes.soe.ucsc.edu/cmpe264/Fall06/Lec5.pdf>

<http://web.stanford.edu/class/ee368/Handouts/Lectures/2014_Spring/Combined_Slides/11-Edge-Detection-Combined.pdf>

<http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/morph.html>

d) Facial Landmarks

1. This is the step where we determine “facial landmarks.”   
   There are several different libraries used commonly for Machine Learning related to facial detection/ recognition which rely on libraries of hand-drawn facial landmarks on “generic” faces (this is perhaps playing into standards of Euro-centric standards of Western beauty and “accepted” facial structures — an important point to note that I will not delve into here but which can be explored here (<https://www.theatlantic.com/technology/archive/2016/04/the-underlying-bias-of-facial-recognition-systems/476991/> )and here (<https://www.theguardian.com/technology/2017/may/28/joy-buolamwini-when-algorithms-are-racist-facial-recognition-bias> ) and more in-depth here (<https://www.ajlunited.org/>) ).

These libraries (i.e. iBUG etc) use these input facial “maps” as a guide — once interest points are detected, the generic 68-point mapping of facial landmarks can be used to refine and adjust the location of the landmarks on an input image.

Euclidean distances between detected interest points and the location of landmarks on inputted maps can verify or discard interest points until an approximate set of 68 points is determined.

For this visualization, I have chosen to forgo utilizing ML technologies and external libraries of facial landmarks for simplicity’s sake. I used a singular image of my own face, manipulated in Adobe PhotoShop to add my approximations for the 68 landmark points, as an alpha mask and compared detected corner points to those 68 landmark points. This is a rather naive, perhaps brute-force way of approximating facial features, but it accomplishes the task needed for this visualization — which is simply, to make your face “Cubist” via a triangulation mesh.

ii) Using a 3x3 matrix of input pixels (reasoning for which comes from this article: ), we calculate the average difference in pixel brightness to find areas — to be further refined to points — of high contrast.  
Why?

Contrary to the Photoshop trigger-happy poreless finish of the advertising world, our faces (all surfaces) are made up of planes which, courtesy of undulating concavities and convexities, reveal dimensionality — also known as what the hair-care world has co-opted as “volume.”

Find these areas where light meets deep shadow, and you have located facial landmarks, which we can use for a myriad of recreational purposes like mapping filters of mildly fetishized forest creatures or for creating an NSA database for quick and painless identification. Endless possibilities.

Let’s make some triangles for now.

<http://docs.opencv.org/3.2.0/d7/d4d/tutorial_py_thresholding.html>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.7883&rep=rep1&type=pdf>

<http://aircconline.com/vlsics/V6N5/6515vlsi02.pdf>

3. **Skeletonization — connecting valid face points**

Not an early Tim Burton claymation flick, Skeletonization is the process by which layers of a form are successively peeled away so as to reveal a final, bare-bones polygonal chain — a skeleton.

Morphology:

<http://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm>

<https://en.wikipedia.org/wiki/M%C3%B6bius_transformation>

**4. Convex Hull**

So we have points. That’s nice. Remember when your 2nd grade art teacher critiqued your family portrait because you drew the faces askew and the dots for the eyes melting into the bottom of the chin and you told her to go fuck herself because André Breton would really enjoy your embrace of a visual perspective that veered away from conventional aesthetics and more towards a way of seeing that embraced the ideal of *rêver en éveillant*? (I wouldn’t know, I skipped second grade but I felt that this would be a good time for you to unload some emotional baggage. That’s really the entire point of AI isn’t it? (See Her)).

Yeah, that’s not this moment. Hopefully, this didn’t detect the points for the eyes as being off the bounding box and the nose has a couple of points to be classified as a blob (not the horror flick. Also, a real term. I know, I was also surprised. Science is astounding, sometimes.)

Either way, you’ve got face points. Now we’re drawing a boundary for our points.

But WAIT. Aren’t those points already within a bounding box (you shout at the sky, tears streaming down your face)? Isn’t this a frame within a frame already? When does the frame end and the picture begin? What about a post-structural space of viewing that doesn’t align itself within conventionality in the restrictive setting of a gallery which itself plays a part in institutional systems of oppression?

Again, please see here and here:

Astoundingly, a convex hull, while a frame within a frame (so fucking meta, I know. Please don’t start sobbing.), is also a beautiful way of demarcating space. We want the smallest boundary. Almost as though we want the boundary that can hug all of its points while still remaining aloof (do you get the portrait now, Ms. Richards?! Symbolism is not something I invented when I was seven, it’s a really poignant device for explaining relational aesthetics without implicating an explicit narrative in a work.)

So here it is.

I’m using a linear time algorithm simply because I like Chazelle’s elegance the most. If you want to look at some Jarvis march thing, please just Google it.

**5. Triangulation**

Sets of three are not fun if you want a college romance but they are nice when you are looking for ways to group points sets in a way that will translate to an even distribution of space and a division of a convex polygon into pieces that are “the most stable shape.”

**6. Delaunay Triangulation**

It is called the “most beautiful triangulation” amazingly not because it buys into the 90s heroin chic aesthetic of emaciation = beauty, but because, rather than preferring the thinnest triangles, this triangulation prefers an organization of triangles which maximize the minimum angle in each triangle.

Maximize the minimum. How do you not see poetry in something like that? How does everyone not start weeping in class when you hear phrases that beg your blood to cease hibernation?

Here you go.

**7. Voronoi**

Voronoi is a tie-breaker algorithm — it determines the delineations between points such that the dividing line between each two input points is equidistant from those two points. A medial axis, if you will.

There are some nice real-world understandings of this:   
The range of a group of cell towers in a given city.

Sad cubicles in a generic, soul-eviscerating job that functions as a cog in (\*insert your favorite bureaucratic corporate conglomerate here\*), meant to keep each person as close and as far from each other as possible at all times.  
This Creative Commons image of cells also illustrates a Voronoi diagram in nature — of which there are a plethora.

Voronoi diagrams can perhaps feel isolating but they can also be protective — as in determining the optimal spacing of hexagonal plates on a tortoise shell so that the convex structure can maximize interior room (understanding that this is also looking at a 3D representation of a Voronoi diagram but, for poetic license, just allow me this example).

Voronoi diagrams map distances from neighbors. Intuitively it follows that a Voronoi diagram (a derivative of the Delaunay Triangulation) stems from the Minimum Spanning Tree and Nearest Neighbor Graphs of a given point set.

GG ( NNG ( DT

General:

<https://processing.org/tutorials/pixels/>

<https://mgold.github.io/Triangulating-Star-Shaped-Polygon/>

<http://www.learnopencv.com/image-recognition-and-object-detection-part1/>

<http://aircconline.com/vlsics/V6N5/6515vlsi02.pdf>

<https://github.com/ironwallaby/delaunay/blob/master/delaunay.js>

<https://forum.processing.org/two/discussion/3705/vertices-of-chain-shape-are-too-close-together-help>

<https://www.geometrictools.com/Documentation/Skeletons.pdf>

Enums in Processing:

<https://forum.processing.org/one/topic/enums-in-processing-1-5-1.html>

**Viola-Jones Algorithm:**

<https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf>

<http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf>

Gaussian Filter:

<http://libweb.surrey.ac.uk/library/skills/Number%20Skills%20Leicester/page_19.htm>

<http://www.winspc.com/what-is-spc/ask-the-expert/305-what-is-a-standard-deviation-and-how-do-i-compute-it>

Feature Detection Algos:

<http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html>

<https://github.com/mariusmuja/flann/blob/master/src/python/pyflann/index.py>

Histogram of Oriented Gradients:

<http://www.learnopencv.com/histogram-of-oriented-gradients/>

Adaptive Thresholding:

<http://aircconline.com/vlsics/V6N5/6515vlsi02.pdf>

In Defense of Nearest-Neighbor Classification:

<http://www.wisdom.weizmann.ac.il/~irani/PAPERS/InDefenceOfNN_CVPR08.pdf>

Etc:

<http://www.cse.cuhk.edu.hk/leojia/papers/portrait_eg16.pdf>

**References:**