**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

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) 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 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.

**4. Convex Hull**

**5. Triangulation**

**6. Delaunay Triangulation**

**7. Voronoi**

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>

**References:**