Image Vectorization For Flat Design Oriented Artwork Purpose: Project Report for Visualization

Peiyi Hou

Abstract

Flat design has become industry standard since the beginning of this decade as silicon valley mega companies adopted it on their mobile and web platforms, which in the meanwhile, contributed to the reduction of skeuomorphic design. The project conducted in the paper, over less than 9 weeks, explored a tiny bit of image vectorization, specifically focusing on produce vector artworks that are suitable for flat design style, from raster images. The project resulted in a tiny Python written raster to SVG converter program, which reads and processes raster image input, and produce flat styled vector image in SVG.

Introduction

In today's UI/UX design field, from logos, frameworks, to decorative contents, vector contents dominated the interfaces, each in a stylized way under the flat design routine as popularized by leading IT companies like Microsoft since early 2000s. This design style embraced the advantages of vector representation, which is scalable and compact compared to raster image. By abstracting real life objects into basic shapes consisting minimal line or curves, with even simpler shading approaches, flat design style contributed to faster responses on software applications across platforms. In addition, the flat design style reduced the time for designers and artists to generate design elements, which allowed quicker update of products and contents, as opposed to a skeuomorphic approach, which is closer to the photo-realistic result as sought by raster image vectorization researchers.

Even though flat design reduced time and effort compared to skeuomorphic style, it is still non-trivial for artists to create vector works from scratch, using vector editing softwares like Adobe Illustrator and CorelDraw, as a single artwork may requires hours or days of usage of those softwares. Specifically, low poly style artworks (a popular form of flat design style), which involves tracing from real-life object, while using raster image as background, adjust polygon shape and fill color accordingly, is fairly tedious to do.

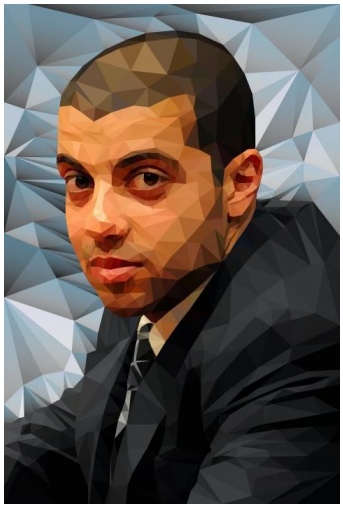

Figure 1 A low poly artwork created manually

Since most low poly style works is mapping polygons to raster image to create an artistic representation, the project sees the potential to adopt triangulation methods to as well as other techniques to automate such process, and make designer's life easier by using the project's demo system. To do so,

this project produced an automated low poly style vector image generator programme, to allow user select an raster image, who would like to work on, and generate a low-poly style vector representation of the raster image at users' preferences by tuning some parameters, which can be subsequently opened and edited by common vector image editor for further refinements.

The demo is written in Python with popular libraries like openCV and numpy.

Point Set Construction

There are three kind of point sets for the system, producing three different flat styles of vector artworks. In General, the three style can be categorized as edge-feature based stylization, greyscale weighted stylization and ortho grid based stylization. Under these three categories, a series of polygonal representation is used to adapt to popular flat design practice, including delaunay triangulation, voronoi tessellation, regular or skewed grid partition and some experiments with spanning tree.

Edge-Featured Oriented

The first implementation is edge-feature based stylization, generating polygons based on the edge-feature of the original raster image. Specifically, the desired result is that, at area of rich features, more polygons will be needed to capture the feature, whereas at area of coarse features, less polygons is needed. To achieve such representation, the crux is to obtain a point set over the raster image that captures the richness of the image, in which points are dense at area of rich feature and coarse on the other hand. The point set will be used as the foundation to build whatever polygon or polyline as vector output.

There are large amount of study and algorithms to obtain edge feature in raster image. In openCV, there are ready to use implementation of those algorithms. For our purpose of gathering point set over edge feature, the Sobel operator was adopted, which produces a grayscale map of detected edge features, to serve as the reference of building point set. The resulted sobel saliency map as computed in both x and y direction and combined together, highlights edge features with light color, and keep area where there is no feature at all black, and intermediate features are highlighted between 0 and 255. Hence, it is easy to extract pixels given a certain threshold and build points from the pixels of the saliency map.

In our implementation, the point set is the remaining of the saliency map pixels after two steps of elimination. In the first step, we culled the pixels below our feature threshold. For example, any pixels that has a grayscale value of 100 is culled. The threshold is tunable by user as a way to figuring out better result. After the first step, only pixels at our desired feature area is left, and we will keep a small portion of those points, and cull the rest randomly, so that there is space between each point in the set, thus polygons can fit in between points. Again, the percentage of elimination for this step is at users' preference, as this second step controls the cardinality of our point set, which controls the density and count of polygons that would be produced. For this point set, three polygonal representation can be built, delaunay triangulation, voronoi graph and spanning tree.

Greyscale Weight Oriented

The second category is greyscale weight oriented stylization. Similarly, we will need to build a point set on top of the raster image, and this time, according to the greyscale. The idea is that, we will build a point set that is denser at darker area of the image, and coarser at lighter area of the image. Hence, there will be more polygons at darker area and less polygons at lighter area.

It is pretty straightforward to get this point set, we just need to convert the image to grayscale, and slice it into intervals of between 0 and 255. For each interval, we keep a decreasing percentage of points from it as part of our point set. Instead of using linear function to drive percentage reduction, we used hyperbolic tangent as a sigmoid function to control

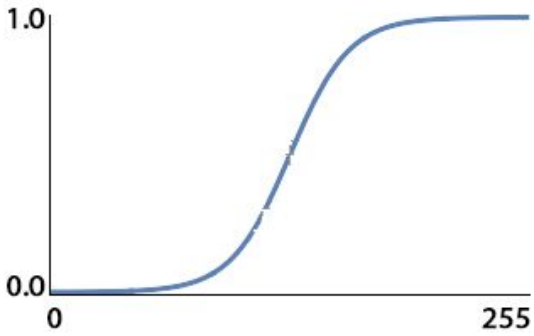

Figure 2. Tanh function mapped

the percent change, as this would steepen the change rate in the middle area, keep the darkest area denser and lightest area coarser. We need to of course map 0-255 to 0-100 percent to do our elimination. Once we have added the culled points from each interval, we have our point set. At this time, we can build polygons to our point set.

There is a drawback on using sigmoid function, that is the requirement of performing histogram equalization on input raster image. Without histogram equalization, the actual image domain could be narrower than 0 - 255. If we do not map the narrower domain to full 0 - 255, we will lose the benefit of larger contrast brought by sigmoid function, as the image domain missed the two ends. Luckily, equalization can be done easily with image editor, and I would strongly suggest preprocess the image before applying vector generator.

Ortho grids

The last major category is regular ortho grid on top of the image area. The grid will serve as the point set to build polygon shapes. In our implementation, we built rectangular grid as our point set. For every other row, we allow user to skew the points to fulfill their design need. Triangles, quadrilaterals and voronoi cells are available to fill between the grid. The demo system also allows generating a spanning tree connecting all the points in the grid.

Polygonal Representation

The polygonal representation in this project by far are delaunay triangulation, voronoi tessellation, and spanning tree on point sets. OpenCV provided standard package to compute delaunay triangulation and voronoi tessellation, since time span of the project is limited, the demo system adopted openCV's standard implementation.

For spanning tree over the point set, the current version implemented Prim's algorithm to find the minimum spanning tree. The edge weight of the graph built from point set is simply the euclidean distance between points. To reduce amount of edges included in the graph, instead of connecting each point with every other point in the point set, we first performed delaunay triangulation on the pointset, and converted the triangles into the graph we want. By doing so, for each point, we only added edges to its closer points to the graph.

As for shading, the demo system fills each polygonal cell with single "flat" color as guided by the flat design style. To determine to the color of each polygon, there are two methods. The first method is simply extracting the color of the centerpoint of the polygon. This method is problematic, because the centerpoint might be a noise pixel which does not represent the area at all. To address this issue, current implementation produces mask on top of each polygon. The mask is a bounding rectangle of each polygon, which is then moved to the center of this polygon. Now the color is we need to represent the polygon is the average color of the mask associated to its polygon.

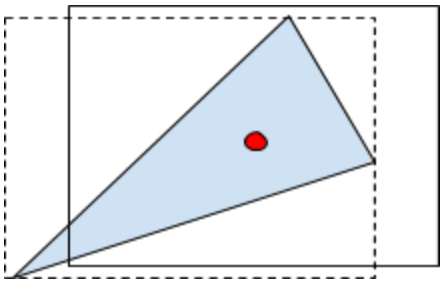

Figure 3. Construction of mask

GUI and Output Options

The demo system has a simple GUI for interactive editing and updating vector design. The open button and convert button does what they say. The drop menu at the top let user selects the kind of vector arts he or she wants. There are total 7 kinds of selection:

Delaunay
Voronoi
Tri-grid
Square-grid
Voronoi-grid
Ortho-Tree
Random-Tree

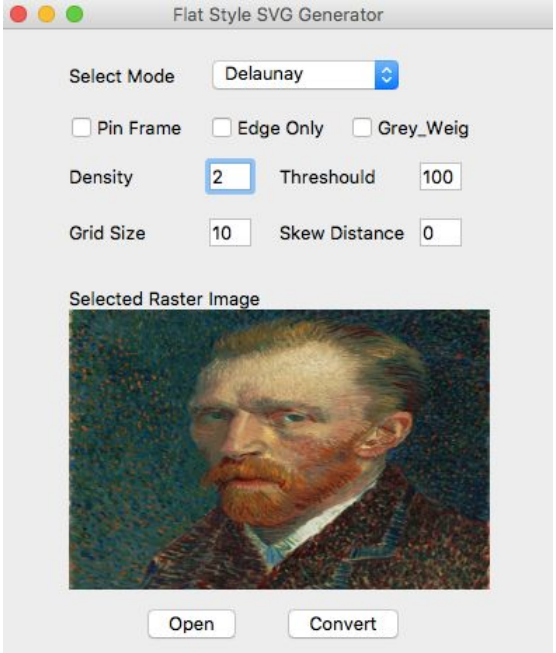

Figure 4. GUI of demo system

Delaunay mode performs delaunay triangulation based on edge feature point sets. Density controls the size of the point set, as we had discussed before, this is the elimination parameter for the second step of point construction for a edge feature based point set. Threshold is the parameter setting the hard boundary of edge feature, as we had discussed that this is the threshold used in the first step. These two parameter is also effective for Voronoi mode and Random-Tree mode, which builds voronoi cells and MST based on edge feature oriented point set respectively.

For Delaunay and Voronoi, if we are processing image with large blank area, then only area with features will be filled with polygons, corners and boundaries may be empty. We can forcibly add some pin points at corners and boundary to the point set to modify the behavior by checking the Pin Frame checkbox.

The Grey_Weig(hted) checkbox switches the point set from edge-feature oriented one to a grayscale weighted one. This checkbox is effective for Delaunay, Voronoi and Random-Tree. If switched, the Density option and Threshold is no longer effective.

The Tri-Grid, Square-Grid, Voronoi-Grid and Ortho-Tree are modes based on ortho grids. For these modes, we can adjust the grid size and skew distance, other parameters will not affect ortho grids based modes.

For any mode except for Ortho-Tree and Random-Tree, Edge only checkbox will get rid of shading of polygons and set polygon's edge stroke color to black.

After setting image and parameters, clicking convert will start the vectorization process and show the produced vector image in a new window. User can repeatedly convert with different settings to sort of interactively select better output. The vector output of the demo system is in SVG format, since it is based on standard xml and more web-friendly than other formats like EPS.

Results and Limitation

In General, the demo system produced preliminary results, and most results are capable of being further adjusted in vector image editors. Hence, the project achieved the basic goal, to provide a tool to help automate and facilitate the creative process. Due to the limited time span, some core functionality is not deeply invested, particularly the delaunay triangulation algorithm and the construction of points set. Many of the hidden parameters in the code need

to be optimized for better result, which need many more test cases and longer time.

The major limitation of the tool is that, this tool is hard coded. Many popular usage of flat vector art is to portray specific object or figure, which all has unique features. The randomness in the point set construction is not smart at all to capture the keypoint representing unique object, such as keeping a point at the eye corner or keep a relatively regular distribution along the jawline. Hence, for example, this kind of tool can definitely incorporate face recognition algorithm or other recognition algorithm to optimize the construction of point set given specific target. However, this is beyond the scope of nine-week span.

For **extra credit**, if applicable, the MST part is beyond the original proposal, although I personally am not quite satisfied with the mst's style and it is buggy from time to time.

Appendix

Figure 5 is the original raster image
Figure 6 is delaunay based on edge feature
Figure 7 is delaunay based on grayscale weight
Figure 8 is voronoi based on edge feature
Figure 9 is voronoi based on grayscale weight
Figure 10 is Tri-grid
Figure 11 is Squared-grid
Figure 12 is Voronoi-grid
Figure 13 is Ortho-Tree
Figure 14 is Random-Tree based on edge feature
Figure 15 is Random-Tree based on grayscale weight

Figure 5

Figure 6

Figure 7

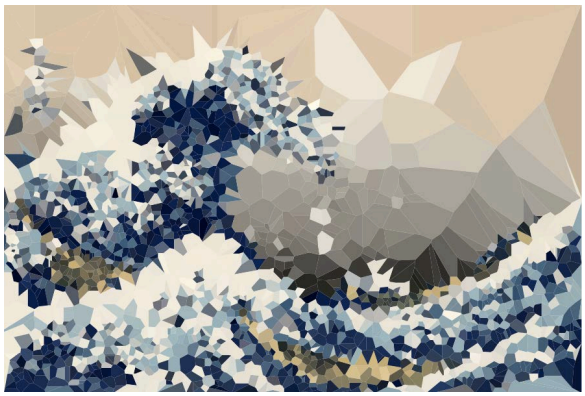

Figure 8

Figure 9

Figure 10

Figure 11

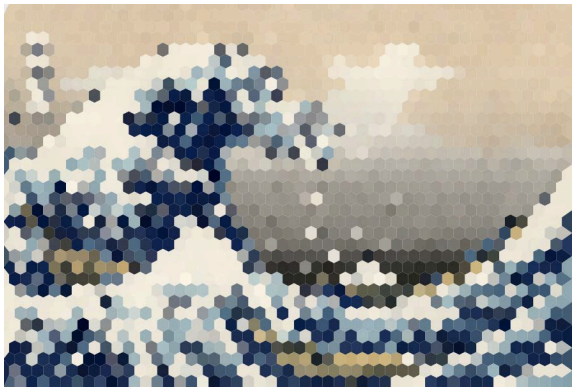

Figure 12

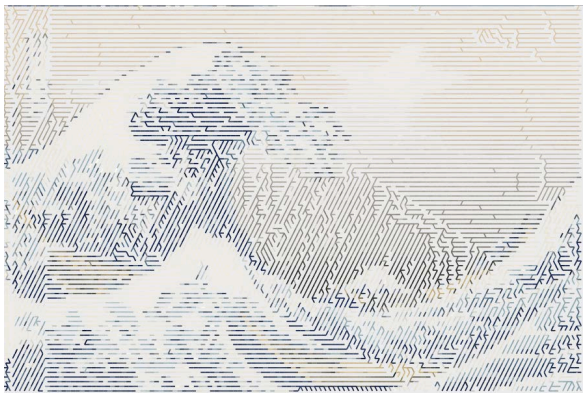

Figure 13

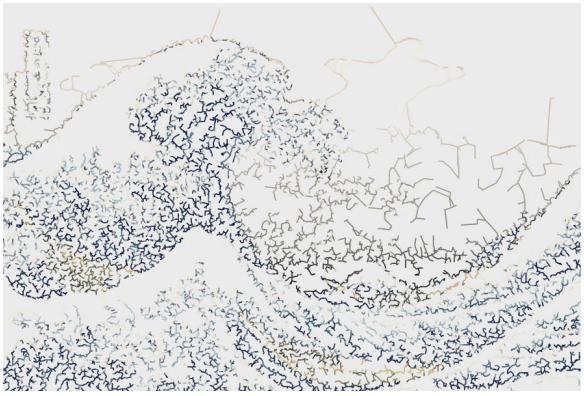

Figure 14

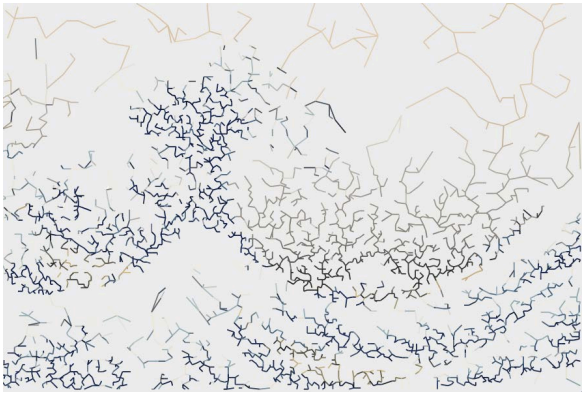

Figure 15