THE COOPER UNION
FOR THE ADVANCEMENT OF SCIENCE AND ART
ALBERT NERKEN SCHOOL OF ENGINEERING

A HYBRID APPROACH FOR IMAGE VECTORIZATION FOR SEMI-GEOMETRIC IMAGES

Jonathan Lam, Derek Lee, Victor Zhang

ECE396 Final Report
Professor Sam Keene
Spring 2022

Contents

1 Abstract

Image vectorization is the process of converting a raster (pixel-based) image to a vector (shape-based) image. While raster images are the dominant mode of image representation, vector graphics may be more efficient for highly geometric images, such as logos, fonts, and maps. Edge tracing methods for vectorization produce clean edges but assume a color-thresholded image. Sampling-based methods work well over color gradients but produce a mesh that may not be well-aligned with edges. We aim to create a hybrid pipeline that combines the benefits of these two methods: performing well over color gradients and producing clean edges. We demonstrate that our method tends to perform better in terms of accuracy (MSE) and visual presentation of edges than the base methods, at the cost of some efficiency of representation.

2 Introduction

2.1 Project Overview

Our project aims to provide an end-to-end *image vectorization* tool for a wide class of image types. Image vectorization is the process of generating a *vector* (shape-based) image that is faithful to the input *raster* (pixel-based) image.

Highly geometric images may benefit greatly from a vector image representation. For example, image vectorization for maps [3] or charts [9] have been very successful. Potrace [18] works well for shapes with well-defined geometric patches. More modern machine learning methods such as Im2Vec [17] can also identify simple geometric shapes.

We wish to study a more general class of raster images: those that remain highly-geometric (so that a vector image representation is useful), but with no other strong assumptions. For example, Potrace requires that an image is binary-thresholded or otherwise color-thresholded, and does not provide a great representation of gradient patches. The classical edge tracing methods for maps or charts also assume a color-thresholded image and may use a knowledge-based system [12] to improve the results. Im2Vec makes simplifying assumptions about the number of shapes it is attempting to identify.

A possible use case for our project is with architectural photographs. Architecture tends to be very geometric; however, complexity is introduced by many factors such as textures, coloring and lighting, fine details, and image quality, that will complicate existing methods of image vectorization. It will be useful to extract a vector representation of an architectural work from a photograph, for further use in machine learning or perhaps as a reference model in computer-aided design.

Due to the complexity of architectural images, we are not too concerned about some information loss; the goal of this project is to explore some heuristic

methods to blindly (i.e., without a knowledge-based system) produce a reasonable representation of a highly-geometric image. Several metrics will be used to quantify the performance of our vectorization method compared to the Potrace and blue-noise sampling methods.

2.2 Overview of Methods

Multiple methods have been considered for this project, including traditional edge tracing, machine learning, and blue-noise sampling (followed by triangulation). The result of our experimentation is a method that augments a sampling-based method with additional information about "strong" edges. This gives us a hybrid method that combines the benefits of sampling methods (robustness to color gradients) and edge tracing (robustness of edges).

2.3 Potential Applications

Our project can potentially be applied to the architecture design process. We envision that an architect may take a photo of an existing architectural design, use our project to process that image, and use the vector-based output to easily edit the image. Alternatively, the input image may be exported to a line-drawing representation generated from the SVG output.

Another potential use case is for machine learning. In computer vision, image data used as input is traditionally in raster format – there is little research performed on how well deep learning performs on vector-based image inputs. We imagine that due to the efficiency of its representation, especially for highly-geometric shapes, we may be able to have more concise information in the deep learning model. This representation may be used for new types of vector image-based ML models, which are currently not widely used. Alternatively, existing machine-learning methods may be used after rasterizing the output.

3 Background

3.1 Raster Graphics

Raster images are the matrix representation of an image. A raster image is conceptually a matrix of pixels, or a bitmap. Each pixel may contain multiple data points representing channels (colors). Historically, bitmaps have been the dominant representation for images due to their conceptual simplicity and the array-based display (and framebuffer) of modern screen technology. Many image-based algorithms depend on the grid-like representation of raster images, such as image compression, parallelized image processing algorithms, and image-based machine learning algorithms. As a result, standards for raster images tend to have wider support than vector graphics.

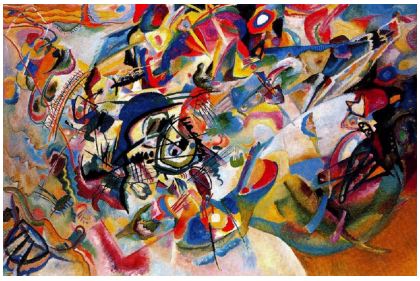
3.2 Vector Graphics

Vector images use a shape-based parameterization of an image. One of the immediate benefits is an efficient representation for purely geometric images, and the efficient and infinite scaling of geometric objects. In order to display a vector image onto a pixel-based screen, the vector image first has to be rasterized, or rendered. Vector graphics are especially useful for web graphics and other highly geometric designs, such as logos, maps, computer-aided design (CAD), and typography. However, vector-based designs tend to be more inefficient for arbitrary image data.

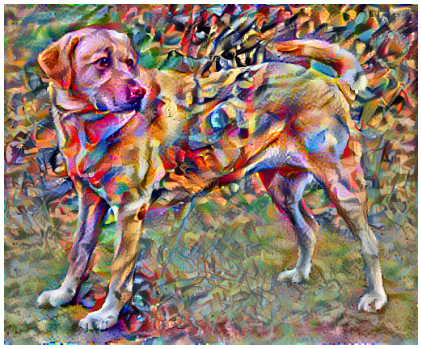
3.3 Scalable Vector Graphics File Format

Scalable Vector Graphics, or SVG, is a standard [16] for vector graphics that uses the XML text format. All elements are represented using combinations of seven geometric shapes: Path, Rectangle, Circle, Ellipse, Line, Polyline, and Polygon.

(a) Yellow Labrador Looking, from Wikimedia Commons



(b) Composition VII by Wassily Kandinsky



(c) Generated image produced with style transfer

Image 1	Image 2	Content Loss
Fig. 1a	Fig. 1c	132352.05
Fig. 1a	Fig. 1b	190958.28
Fig. 1b	Fig. 1c	190266.86

(d) Sample content loss for example images

Figure 1: Content loss example

Since it is a textual format, it can be easily examined and manipulated by computers or by humans. The SVG standard is stable and supported by many modern applications, including most PDF viewers and web browsers. Due to its wide support as a vector image format, we will be using the SVG format as the default format for our vector graphics.

3.4 Content Loss

To evaluate our results, we want an evaluation metric to quantify the accuracy relative to the original image in terms of visual similarity. Content loss [5] was introduced as a loss function to train generative adversarial networks for style

transfer, and serves our purpose.

The authors define the content loss between two images as the Euclidean distance between the high-level outputs of a trained classifier. This stems from a theory about deep convolutional neural networks: the early layers in a classifier are used to extract low-level features, such as edges, while the later layers of the classifier use the low-level features to create high-level features, such as an arm or a leg.

We tested the content loss metric on three images related to style transfer[1]. Fig. 1c is generated using the content from Fig. 1a and the style from Fig. 1b. The content loss between Fig. 1a and Fig. 1c should be the lowest, compared to the content loss between Fig. 1a and Fig. 1b, because the two images have the same content but with different style. As seen in Fig. 1d, we get the expected results.

[1]The sample images shown come from `https://www.tensorflow.org/tutorials/generative/style_transfer`.

4 Related Work

4.1 Tracing Methods for Vectorization

One method of image vectorization is referred to as *tracing*. The intuition behind this is that a raster image can be thought of as a collection of adjacent image patches, and we can vectorize an image by detecting edges of shapes.

A noteworthy implementation of image tracing is the Potrace algorithm [18]. As the name suggests, Potrace first attempts to convert a raster image into a series of polygonal paths via edge detection and straight-line detection, and then attempts to simplify (optimize) polygons by reducing path cardinalities and introducing Bezier curves. It employs many useful heuristics to improve image quality, such as removing speckles smaller than a given "turd size," detecting and smoothing corners, redundancy coding in the target format, scaling and rotating a small set of parameterized curves, and data quantization. An illustration of the stages of the Potrace algorithm is shown in Fig. 2. The implementation of Potrace is open-source, and the program is highly configurable via command-line options.

This interpretation of vectorization is useful for simple raster images that are indeed a collection of adjacent shapes, such as map data, floor charts, typography, or charts. For such images, the Potrace algorithm is both reliable and efficient. We use Potrace in our implementation to address some of the limitations in our method.

One of the drawbacks of tracing is that we can only trace edges on a binary thresholded image; if there aren't clearly defined edges, or if there are image gradients (as is often the case), it doesn't represent an image as well. Tracing can be applied to color images by thresholding the image by color or brightness level, and producing vector images for each thresholded layer, but this may seem choppy and low-quality. Tracing also does not recognize non-contiguous

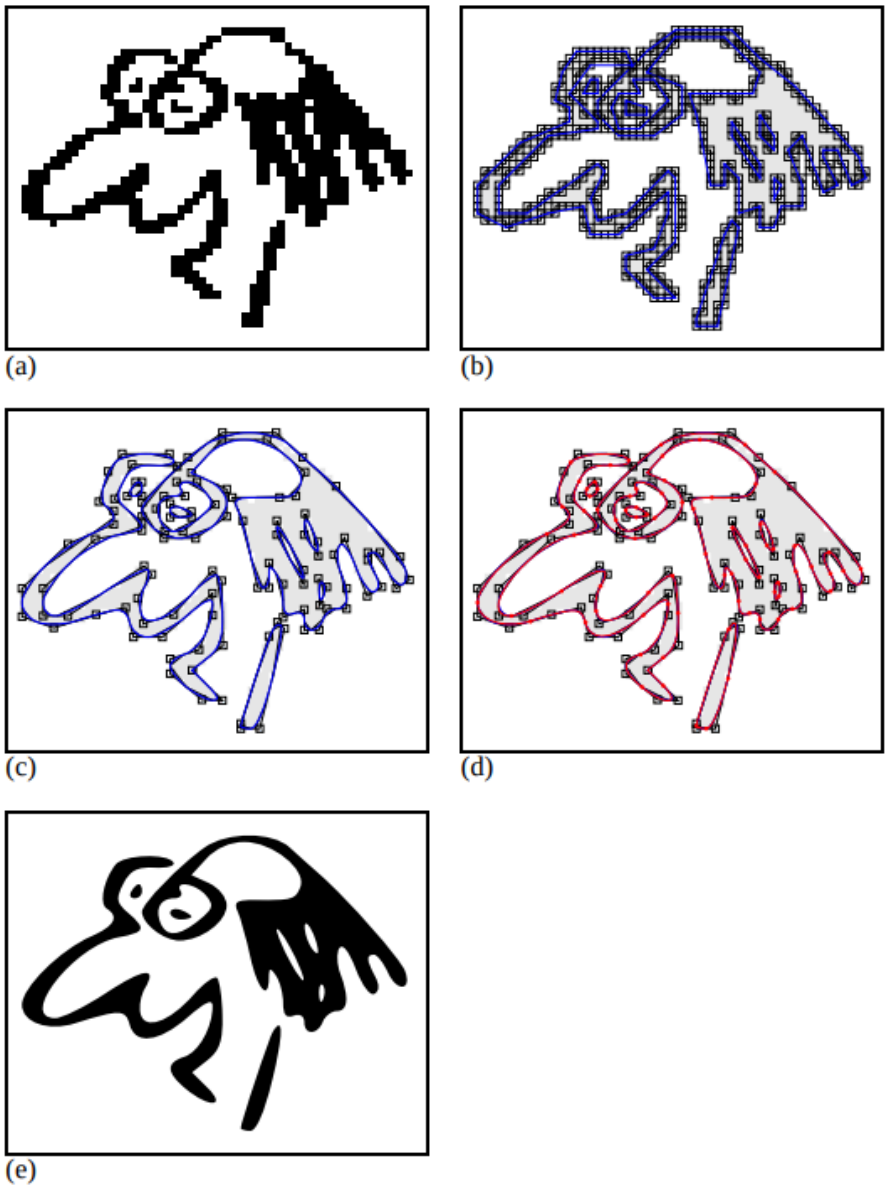


Figure 2: An illustration of the Potrace [18] vectorization process

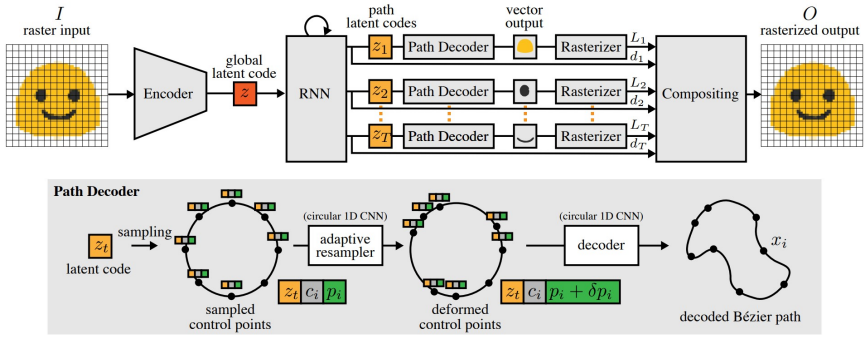


Figure 3: Architecture overview for Im2Vec from [17]

shapes (e.g., simple shapes that intersect other shapes), which can allow for more aggressive optimization or semantic segmentation (which may be achieved by specialized neural network approaches such as [11]).

Most research on tracing predates other methods, but there have been some recent innovations. For example, Yang et al. [20] implemented beziregion (bezier region) approximation for clipart, which directly optimizes beziregions rather than going through intermediate polygons, and there have been several methods targeted at vectorizing line drawings [1, 6, 13].

4.2 Machine Learning Approaches to Vectorization

There have been several recent neural-network based approaches to image vectorization. One notable example is Im2Vec [17], a deep neural-network to vectorize images without supervision.

We explored using this model as part of our approach, but it appears that the authors hard-coded the number of shapes in the input raster image, along with the colors of each shape. We are looking to apply our model to more generic images, which will likely not be as well-defined as the emojis used in their experiments.

Neural-network-based vectorization like Im2Vec appears to be mostly limited

to simple images for the time being, but they can achieve some useful effects that may be difficult using conventional algorithms. For example, Kim et al. [11] achieves "semantic segmentation," which is a complex task that will be difficult to approximate using classical deterministic heuristics.

4.3 Sampling Methods for Vectorization

Sampling methods tackle the vectorization problem by stochastically approximating regions of the vector image. This allows us to achieve a reasonable performance and accuracy. Sampling methods may approximate edges less accurately than edge tracing, but they can overcome some of tracing's limitations, namely being limited to binary or multi-level thresholding. Like tracing methods, it extends fairly well to more complicated images, unlike machine learning methods, which appear to be more limited to simple images.

An example procedure for image vectorization through sampling is shown in Zhao, Feng, and Zhou [21]. The sampling method for vectorization is comprised of three steps. The image is first convolved with a Sobel differential operator to generate an *importance matrix*. Importance corresponds to spatial gradients in the original image; larger gradients may indicate regions with more detail. We then apply blue-noise sampling to the image using the importance matrix. Blue-noise sampling [19] is a general technique to non-uniformly sample an image, such that areas of higher importance are sampled at a greater density. Thus, the sampled points are more tightly clustered around more detailed regions, giving a better representation of the image. The sampled points are then triangulated using a Delaunay triangulation, which is then exported to a vector image format such as SVG.

A similar work is vectorization of cartoon images via shape subdivision [23]. This triangulates the input image, and then performs heuristics to merge trian-

gles to mitigate artifacts from triangulation. We take a similar approach, but both attempt to optimize triangles (via a quadric error metric) and augment the triangulation with additional information about "strong" edges.

4.4 Vector Image Optimization

Several methods for optimizing a polygonal path (i.e., a closed polyline) into a smaller polyline, and by expressing curved sequences of edges as a single Bezier curve, are explored in the Potrace algorithm [18]. However, this only deals with simplifying curves, while maintaining the overall topology. The sampling method generates a triangulated mesh rather than a sequence of closed paths; in this method, the optimization scheme may look different and may change the overall topology. For example, we do not have any polylines to curve-optimize unless some edges are removed to form non-triangular polygonal patches; it is then a matter of which edges can be removed while retaining fidelity to the original image.

Garland and Heckbert [7] proposed using the Quadric Error Metric (QEM) for mesh surface simplification. QEM is used for measurement of error that evaluates the distance of a point in mesh from its ideal position. The method computes the QEM for all valid pairs of points and then finds and collapses the pairs of least cost. The method works for 3D meshes and preserves the primary features of the shape. Since our algorithm first samples in 2D space based on importance, the sampled point cloud needs to be extended to 3D using color information. The resulting 3D points should be less dense but still preserve the sampling feature since decimated pairs of points in areas of high importance would result in higher loss under the QEM.

5 Proposed Method

We propose a hybrid method based on both the blue-noise sampling [21] and Potrace [18] algorithms. In this section we describe the major components of the image pipeline.

Our implementation is a mix of Python and C++[2]. The OpenCV library [2] was used to handle image processing tasks; the `numpy` library [14] was widely used for generic numeric operations; the `open3d` library [22] was used for blue-noise sampling; and the `cairo` library [8] was used for exporting to SVG.

5.1 Sampling

The sampling process is based on Zhao, Feng, and Zhou [21]. The input image is a regular raster image, such as Fig. 4. The first step is to apply the gradient operator on the image to get the importance matrix, shown in Fig. 5. This involves convolving the image with four 3×3 Sobel filters (horizontal, vertical, and two diagonals) and finding the elementwise maximum along the four outputs to determine the magnitude of the gradient, which is interpreted as the local information content or "importance" of the area surrounding a given pixel. Next, the importance matrix is thresholded so that only points of high importance are retained, shown in Fig. 6. This is essentially a high-pass filter; low-frequency components of the image are lost. The intuition is that low-frequency elements can be represented with uniformly-colored shapes without much information loss.

The code used for sampling is a C++ package provided by Ostromoukhov, Donohue, and Jodoin [15]. This method is very fast and deterministic, using Penrose tilings and Fibonacci numbers to sample using the importance map.

Now we perform the sampling to obtain Fig. 7. The sampling density at a

[2]The GitHub repository is `https://github.com/Victooooooor/Vectorize-Arch`.

Figure 4: Original image

pixel is a function of the importance of that pixel; a higher importance leads to a higher sampling frequency. This is known as "blue-noise" sampling, and allows us to focus more detail on regions of higher information content. After sampling, the image is converted to a triangular mesh by performing the Delaunay triangulation, as shown in Fig. 8.

5.2 Generating Multi-Thresholded Potrace Curves

The Potrace command line utility does not provide the ability to perform a multiple-color thresholded scan. This multi-color scan is available in popular software such as Inkscape, but is not available as an accessible command-line tool[3]. As a result, we created our own pipeline for automatically generating a multi-thresholded Potrace vectorization.

Since Potrace requires binary-thresholded images, we threshold the image into its major color groups using the OpenCV implementation of a standard k-means color quantization algorithm. We then run each of the k color groups

[3]To the authors' best knowledge.

Figure 5: Importance function applied to the image

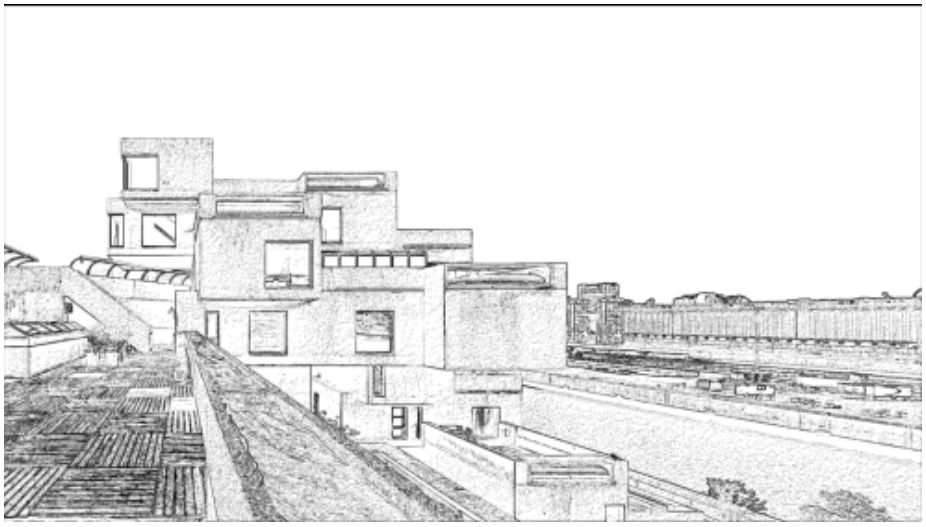


Figure 6: Thresholded points

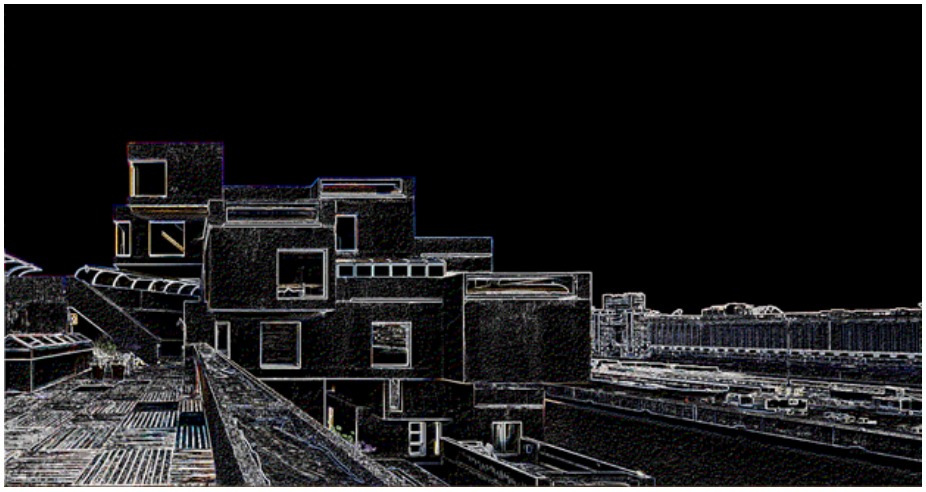


Figure 7: Sampled points

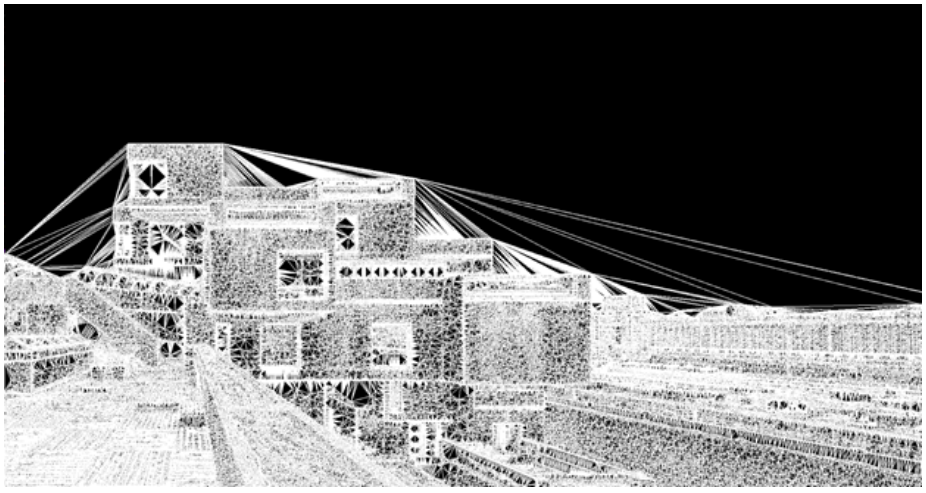


Figure 8: Triangulated image

through the Potrace algorithm separately to generate k SVG images, and merge (concatenate) the results into a single SVG.

5.3 Improving Edges in the Mesh

Blue-noise sampling attempts to represent areas of higher detail with a higher sampling density. The greatest issue with the blue-noise sampling is that it does not represent edges well. Ideally, many of the edges in the image will align with edges of the mesh. We can attempt to manually adjust the sampled points to lie along edges. The following two heuristics are used to try to improve edges.

5.3.1 Sampling Strong Edges

We may attempt to augment edges by simply increasing the number of samples along the edge. To do this, we identify the set of points that form "strong edges." This may come from an edge detection algorithm; in our case, we choose points which have an importance (gradient) above an arbitrary threshold. Then we randomly sample from this set with some small probability, and use these sampled points to augment those from blue-noise sampling. We note that this adds points and increases the file size. This increase in points is proportional to the number of pixels that lie on "strong edges."

5.3.2 Unifying Edges with Potrace

Another method to improve edges is to move sampled points that lie close to an edge to that edge. We use a custom heuristic to combine our sampled points with the Bézier curves that are generated by Potrace. We first rasterize the Bézier curves in order to be able to compare the curves with our sampled points. Next, we replace our sampled points with the closest point on a Bézier curve if the closest point is within a certain distance of the sampled point.

If there are multiple points on the Bézier curve within the same distance, we use all of the points. This distance is determined by the attraction distance ϵ, which represents a square centered on a point on the Bézier curve. The square's corners are ϵ pixels up/down and left/right of that point. For example, one corner would be at $(+\epsilon, -\epsilon)$ relative to the point on the curve. If the sampled point is within this square, the point on the Bézier curve is a valid candidate for replacement. We check all of the Bézier curves generated by Potrace, and for each sampled point, we replace it with the closest point across all of the Bézier curves, assuming a closest point exists.

For our implementation of this algorithm, we first preprocess the sampled points in order to efficiently find all valid sampled points that are within the given distance (determined by ϵ). We accomplish this by creating a 2D matrix that is the same size as our image, where $cell_{i,j}$ represents all sampled points within the given distance. We iterate through each sampled point, and insert that sampled point into all indices within a square centered at that sampled point.

5.4 Sampling Points Around Perimeter

In order for the triangulated mesh to cover the whole image, we need to add additional points around the perimeter of the image before triangulation. The current implementation uniformly samples points around the perimeter. While this does not guarantee that every pixel is part of the mesh, it causes most of the pixels to be covered.

5.5 Mesh Optimization

To increase the efficiency of the representation, we wish to reduce the number of points from the sampling without greatly reducing accuracy. To do this,

we decimate points following the method by Garland and Heckbert [7], which decimates points in such a way to optimize the quadric error metric (QEM). To transform the 2D sampled point cloud to 3D, we use the color information at the sampled points as the third dimension. The resulting 3D point cloud is then simplified using the QEM decimation method implemented by the `open3d` library [22].

5.6 Triangle Mesh Coloring

We experiment with three methods for determining the color of a mesh triangle. The first method takes the mean of the RGB values at the three points of the triangle. The second method randomly samples a set number of integer points inside of the triangle and takes the mean of the RGB values of those points. The third method takes the mean of the RGB values at every integer point in the triangle. We note that the third method approximates the triangle, so it may sample integer points outside of the triangle. In addition, the third method utilizes the first method for small triangles, where the number of integer points inside the triangle is less than three. We currently use the second method, which randomly samples points inside of the triangle. This method has good performance and gives good color estimates for the triangles.

5.7 Writing to SVG

The `pycairo` and `cairosvg` libraries are used to export the triangulated representation to SVG and PNG files. The PNG file is a rendered (rasterized) version of the SVG image, used for content loss and MSE metrics.

5.8 Evaluation Metrics

We evaluate based on several metrics. A couple of our metrics are familiar: file size and mean-square error (MSE). We also implement a custom content loss metric based on Dumoulin, Shlens, and Kudlur [5]. Similar to the methodology used by the authors, we use a pre-trained VGG-19 and strip off the $conv_5$ and fully-connected blocks. We note that although the authors used a VGG-16, we use a VGG-19, similar to Huang and Belongie [10]. In order to get the content loss between two images, we take the Euclidean distance between the outputs of the stripped VGG-19.

5.9 Overview of Proposed Model

Name	Default Value	Description
Importance Scalar	100.0	A positive scalar used at the stage of importance sampling, the importance matrix is scaled by this parameter. A higher value results in higher sampling density.
Decimation Scalar	10	A constant larger than 1, used to calculate the expected number of points for QEM mesh simplification. The expected number of points is calculated as the number of sampled points divided by the decimation scalar. A higher value results in stronger point decimation.
Potrace Scans	4	A positive integer, the number of scans of different color threshold chosen by k-means clustering. Same as number of colors in the resulting SVG.
Attraction Distance	15	A positive integer, the maximum distance (L^∞) between sampled points and curve generated by Potrace for the point to tangency point.
Edge Density	0.01	A float in the range $(0, 1)$, the probability of a point on an edge to be chosen. A higher value results in a higher point density for an edge.

Table 1: List of hyper-parameters

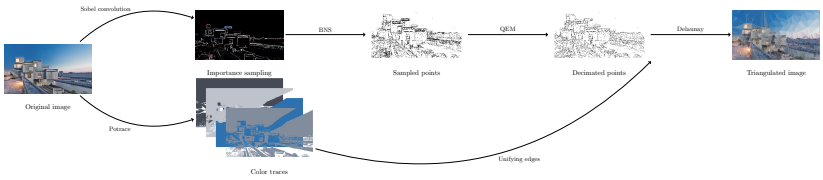


Figure 9: Architecture diagram

6 Results

As seen by Table 2, Potrace produces images that are almost always smaller than our hybrid method. Merging the blue-noise sampling points with Potrace usually significantly increases the file size, although in rare cases, such as with simple images (e.g. Fig. 17), this can reduce the file size.

As seen by Table 3, Potrace almost always performs better than our hybrid method, according to the content loss metric. However, it is important to note that content loss may not be the best metric for measuring the accuracy of the vectorized image relative to the original raster image. For example, as seen by Fig. 17, the Potrace image does not look similar to the original image at all, yet its content loss is lower than both the BNS method and our hybrid method.

As seen by Table 4, our hybrid method consistently performs the best (with the exception of a single experiment), according to the MSE metric. Our hybrid method likely performs better than the BNS method because our hybrid method typically has significantly more points. Additionally, our hybrid method likely performs better than Potrace because it is able to capture color more effectively. Potrace is restricted to a predetermined number of unique colors (we use four colors in our experiments), while our method can use as many unique colors as possible.

On a more qualitative measure, Fig. 13 shows that our hybrid method significantly improves the edges in our final method. The BNS method captures the colors, but the edges are unrecognizable.

Image	BNS	Hybrid	Potrace
1	367	485	**267**
2	203	196	**106**
3	**126**	405	399
4	**208**	2890	1596
5	**696**	4270	5879
6	**358**	1369	702
7	**299**	5723	1725
8	89	77	**23**
9	205	327	**70**

Table 2: File size (kB) by method

Image	BNS	Hybrid	Potrace
1	110	108	**84**
2	84	84	**62**
3	75	68	**42**
4	141	98	**89**
5	125	**89**	90
6	128	105	**86**
7	116	66	**47**
8	67	71	**60**
9	81	**71**	99

Table 3: Content loss ($\times 10^3$) by method

Image	BNS	Hybrid	Potrace
1	63.02	**61.75**	88.85
2	45.73	**40.87**	67.10
3	26.29	**19.11**	73.03
4	75.35	**47.76**	92.25
5	86.29	**77.17**	88.86
6	69.59	**62.58**	87.65
7	59.63	**37.03**	77.63
8	**50.86**	56.15	100.04
9	79.21	**60.41**	100.76

Table 4: MSE (rounded to nearest hundredth) by method

7 Future Work

7.1 Experiments with the Pipeline

7.1.1 Order with Pipeline Components

There is a large design space for our the model. While the plain BNS method is relatively simple (perform importance map, sample, and triangulate), our method involves many more components intended to improve the performance of edges, and decimates points to achieve some improvement in representational efficiency. Transposing or moving certain components of the pipeline (e.g., moving the QEM decimation step to after merging with Potrace) may result in improved results. This requires a large number of additional experiments.

7.1.2 Hyperparameter Search

In addition to moving large components around, performing an exhaustive hyperparameter grid search may be useful to determine which parameters generate the best results.

7.2 Alternative Methods to Strengthen Edges

Currently, two heuristic methods are used to strengthen edges. There is a large design space for both of these methods.

The current algorithm to merge points with Potrace, as described in Section 5.3.2, is fairly simple. It is a quadratic ($\mathcal{O}(N^2)$) algorithm, and thus fairly slow. Also, it uses a simple L^∞ distance metric rather than a Euclidean distance, out of simplicity. There may be a better algorithm from computational geometry to perform the merging task.

Points are randomly sampled from a set of "strong edge points," which are points for which the importance (gradient) is above a certain threshold.

Alternatively, points may be sampled from edges generated from another edge detection method, such as the Canny edge detection algorithm [4].

7.3 Curve Simplification

The result of the proposed algorithm is a triangulated mesh. While this is a simple representation and easily generated using the Delaunay triangulation algorithm, it is not especially visually appealing, especially when groups of edges may be joined together into Bezier curves.

It may be beneficial to perform curve optimizations to improve visual output similar to Selinger [18]. Yang et al. [20] works directly with beziregions, although this may be difficult to integrate into our pipeline.

7.4 Image Size Sensitivity

Certain aspects of our pipeline are sensitive to file size, such as the importance scaling factor, which affects the density of sampling. It may be beneficial to ensure that all parameters are image-size-invariant.

7.5 Hybrid Method Runtime Performance

Runtime or algorithmic performance was not a major concern for this work, as it was a secondary goal to improving the quality of vectorization. However, it is a very practical concern, and should be considered in the future. Some of the pipeline may be amenable to parallelization, such as the sampling and Potrace multi-scan passes.

7.6 Mesh Coloring Methods

We briefly experimented with three methods for coloring triangles in the mesh. However, further experiments would be useful in determining which method is

most beneficial (also taking into account runtime performance).

7.7 Using the Output of the Pipeline

7.7.1 Architectural Design Process

The original intention for our model was to aid in the architectural design process. It will be useful for architects to generate line drawings or CAD models. Our model does not currently do this, but such a representation may benefit from the mesh outputted by our model.

7.7.2 Machine Learning Preprocessing

Most, if not all, image machine learning models take raster images as input. In order to use our proposed method as input for a machine learning model, the output must be rasterized, or a vector-based machine learning model must be developed. This lies outside the scope of our work.

7.7.3 Art Generation

The outputted vectorized images have a distinct "stained glass" texture that may be desirable in an art context. A number of peers have said that the output "looks cool" and this may be desirable simply for visual effect.

7.8 Mathematical Model of Pipeline

The nature of this work is highly heuristic and emipirical; it would be pleasing to have a mathematical model behind the enhancements to estimate the gains in vectorization and to check the empirical results against.

7.9 Improved Evaluation Metrics

7.9.1 Controlling for Features of the Vectorized Images

The current evaluation metrics are very basic. Moreover, when comparing images in an experiment, we do not attempt to control features of the output images (e.g., number of mesh points, filesize, etc.) but instead only control the model parameters. In other words, the current comparison may be less fair than if the outputted images of BNS and our hybrid method had the same filesize.

7.9.2 Evaluation Metrics for Visual Perception

Much of the motivation behind this work is highly related to visual perception: Potrace appears to do worse for gradients, and BNS tends to produce much worse edges. However, none of the evaluation metrics directly test either of these visual-based perceptions.

We use content loss as an attempt to measure overall semantic fidelity of the vectorized image, but this may not be the best metric. For example, in experiment 8, which contains the gradient, the Potrace output has a lower content loss despite being much less accurate than the other methods. We hypothesize that a better metric may be style loss [5].

8 Conclusion

We have implemented a basic framework for vectorizing raster images, primarily based on blue-noise sampling [21] and Potrace [18]. Our proposed framework involves performing blue-noise sampling on an image, merging sampled points with the Potrace output, triangulation, and exporting to an SVG file. While this method currently does not generate a highly efficient representation, it performs better than BNS and Potrace on some metrics, particularly accuracy. Our method is able to both handle gradient patches better than Potrace, and represent edges better than blue-noise sampling. There is much future work remaining to further tune this model for efficiency.

References

[1] Mikhail Bessmeltsev and Justin Solomon. "Vectorization of line drawings via polyvector fields". In: *ACM Transactions on Graphics (TOG)* 38.1 (2019), pp. 1–12.

[2] Gary Bradski and Adrian Kaehler. "OpenCV". In: *Dr. Dobb's journal of software tools* 3 (2000), p. 2.

[3] Girija Dharmaraj. *Algorithms for automatic vectorization of scanned maps*. Citeseer, 2005.

[4] Lijun Ding and Ardeshir Goshtasby. "On the Canny edge detector". In: *Pattern recognition* 34.3 (2001), pp. 721–725.

[5] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. *A Learned Representation For Artistic Style*. 2017. arXiv: 1610.07629 [cs.CV].

[6] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. "Fidelity vs. simplicity: a global approach to line drawing vectorization". In: *ACM Transactions on Graphics (TOG)* 35.4 (2016), pp. 1–10.

[7] Michael Garland and Paul Heckbert. "Surface Simplification Using Quadric Error Metrics". In: *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics* 1997 (July 1997). DOI: 10.1145/258734.258849.

[8] Cairo Graphics. *Cairo: A 2D graphics library with support for multiple output devices, version 1.1*. 2006.

[9] Weihua Huang, Chew Lim Tan, and Wee Kheng Leow. "Elliptic arc vectorization for 3D pie chart recognition". In: *2004 International Conference on Image Processing, 2004. ICIP'04.* Vol. 5. IEEE. 2004, pp. 2889–2892.

[10] Xun Huang and Serge Belongie. *Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization*. 2017. arXiv: 1703.06868 [cs.CV].

[11] Byungsoo Kim et al. “Semantic segmentation for line drawing vectorization using neural networks”. In: *Computer Graphics Forum*. Vol. 37. 2. Wiley Online Library. 2018, pp. 329–338.

[12] Kyong-Ho Lee, Sung-Bae Cho, and Yoon-Chul Choy. “Automated vectorization of cartographic maps by a knowledge-based system”. In: *Engineering Applications of Artificial Intelligence* 13.2 (2000), pp. 165–178.

[13] Gioacchino Noris et al. “Topology-driven vectorization of clean line drawings”. In: *ACM Transactions on Graphics (TOG)* 32.1 (2013), pp. 1–11.

[14] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[15] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. “Fast Hierarchical Importance Sampling with Blue Noise Properties”. In: *ACM Transactions on Graphics* 23.3 (2004). Proc. SIGGRAPH 2004, pp. 488–495. URL: `http://www.iro.umontreal.ca/~ostrom/ImportanceSampling/`.

[16] Antoine Quint. “Scalable vector graphics”. In: *IEEE MultiMedia* 10.3 (2003), pp. 99–102.

[17] Pradyumna Reddy et al. *Im2Vec: Synthesizing Vector Graphics without Vector Supervision*. 2021. arXiv: `2102.02798 [cs.CV]`.

[18] Peter Selinger. “Potrace: a polygon-based tracing algorithm”. In: *Potrace (online), http://potrace.sourceforge.net/potrace.pdf (2009-07-01)* 2 (2003).

[19] Dong-Ming Yan et al. “A survey of blue-noise sampling and its applications”. In: *Journal of Computer Science and Technology* 30.3 (2015), pp. 439–452.

[20] Ming Yang et al. “Effective clipart image vectorization through direct optimization of bezigons”. In: *IEEE Transactions on Visualization and Computer Graphics* 22.2 (2015), pp. 1063–1075.

[21] Jiaojiao Zhao, Jie Feng, and Bingfeng Zhou. “Image vectorization using blue-noise sampling”. In: *Imaging and Printing in a Web 2.0 World IV*. Vol. 8664. International Society for Optics and Photonics. 2013, 86640H.

[22] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “Open3D: A modern library for 3D data processing”. In: *arXiv preprint arXiv:1801.09847* (2018).

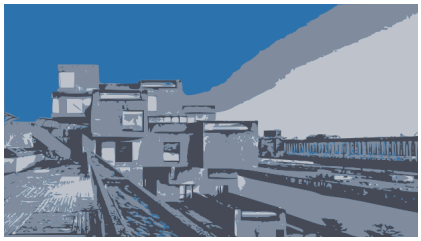
[23] Ju Jia Zou and Hong Yan. “Cartoon image vectorization based on shape subdivision”. In: *Proceedings. Computer Graphics International 2001*. IEEE. 2001, pp. 225–231.

A Appendix

(a) Original image

(b) BNS image

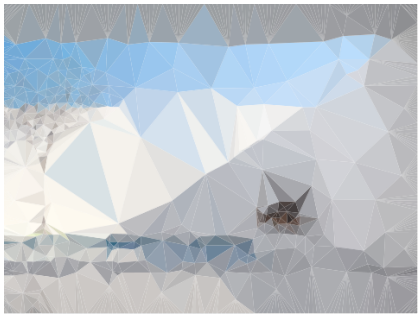
(c) Hybrid image



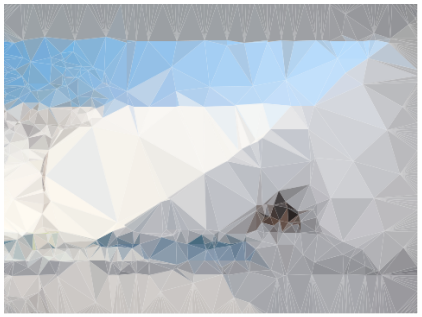
(d) Potrace image

Figure 10: Set of images for experiment 1

(a) Original image



(b) BNS image

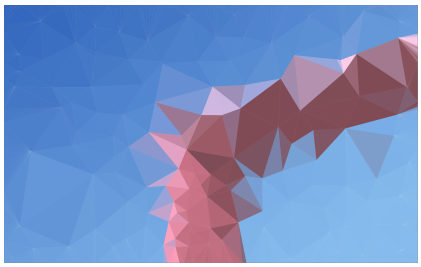


(c) Hybrid image

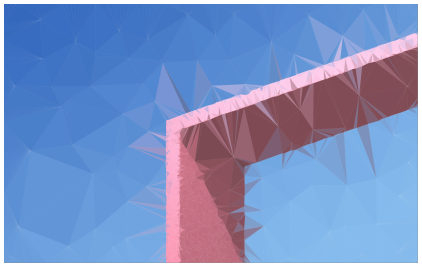
(d) Potrace image

Figure 11: Set of images for experiment 2

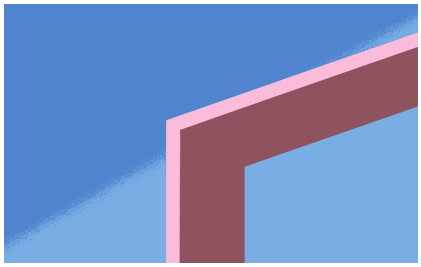
(a) Original image



(b) BNS image



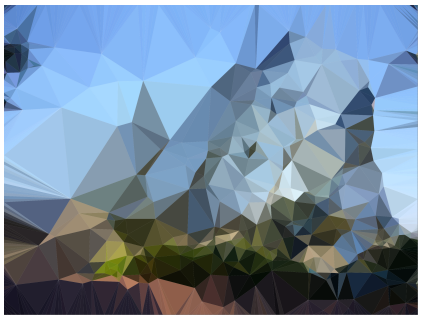
(c) Hybrid image



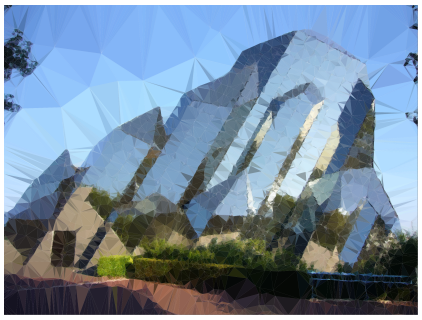
(d) Potrace image

Figure 12: Set of images for experiment 3

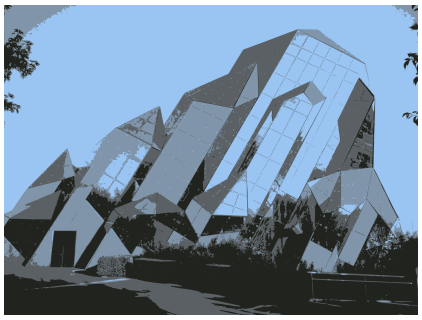
(a) Original image



(b) BNS image



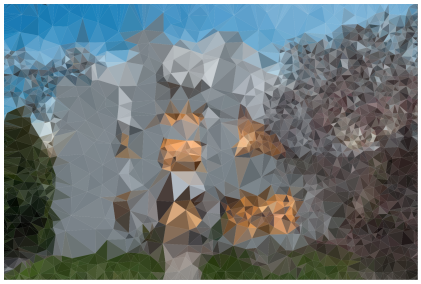
(c) Hybrid image



(d) Potrace image

Figure 13: Set of images for experiment 4

(a) Original image



(b) BNS image

(c) Hybrid image

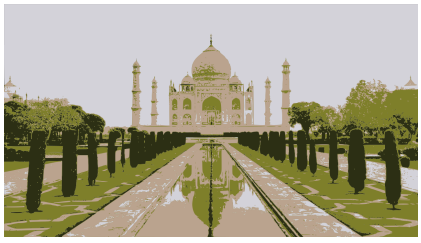
(d) Potrace image

Figure 14: Set of images for experiment 5

(a) Original image

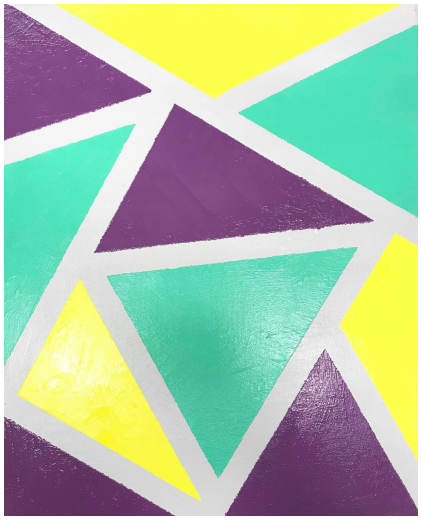
(b) BNS image

(c) Hybrid image



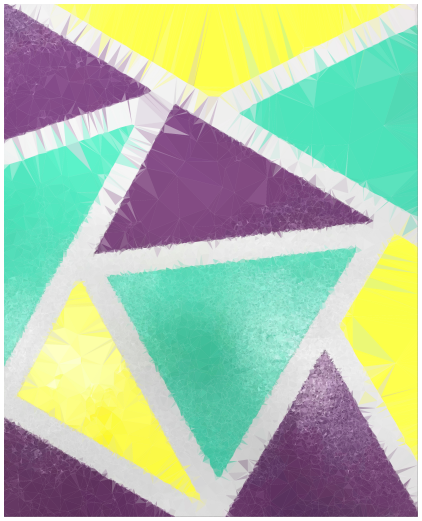
(d) Potrace image

Figure 15: Set of images for experiment 6

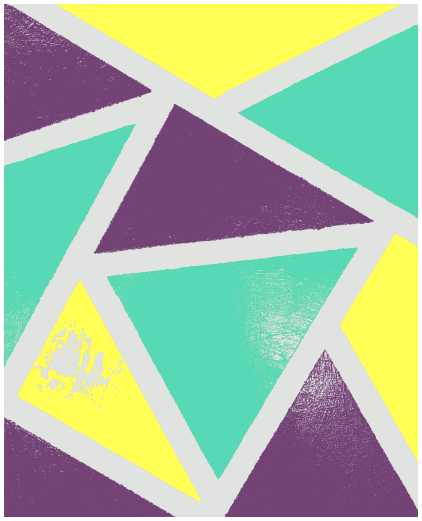


(a) Original image

(b) BNS image



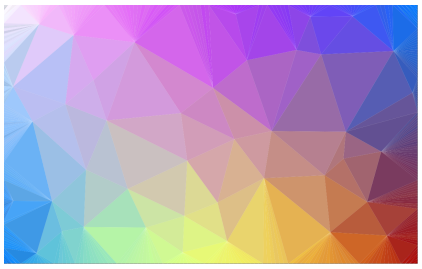
(c) Hybrid image



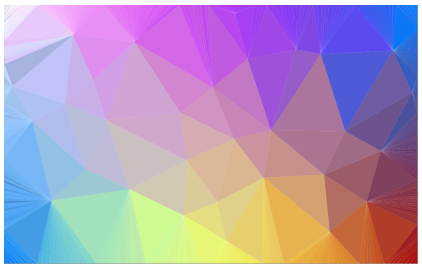
(d) Potrace image

Figure 16: Set of images for experiment 7

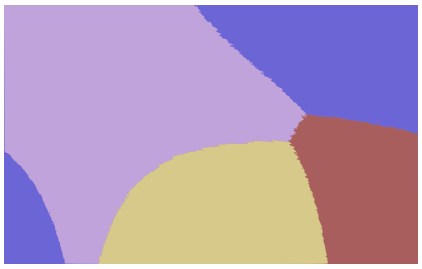
(a) Original image



(b) BNS image

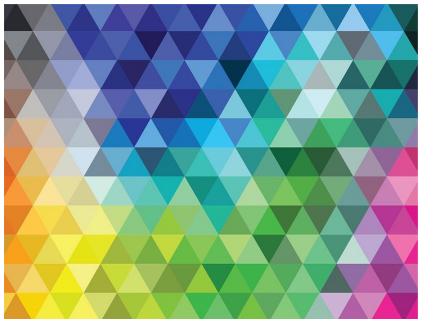


(c) Hybrid image



(d) Potrace image

Figure 17: Set of images for experiment 8



(a) Original image

(b) BNS image

(c) Hybrid image

(d) Potrace image

Figure 18: Set of images for experiment 9