# A Deep Learning-Based Perception-Driven Vectorization Approach For Semi-Structured Imagery

A Special Problem by

Sean Francis N. Ballais
2015-04562
BS CS

Presented to the Faculty of the
Division of Natural Sciences and Mathematics

In Partial Fulfillment of the Requirements
For the Degree of
Bachelor of Science in Computer Science

University of the Philippines Visayas
TACLOBAN COLLEGE
Tacloban City

Month Year

This special problem, entitled "**A DEEP LEARNING-BASED PERCEPTION-DRIVEN VECTORIZATION APPROACH FOR SEMI-STRUCTURED IMAGERY**", prepared and submitted by **SEAN FRANCIS N. BALLAIS**, in partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE** is hereby accepted.

PROF. VICTOR M. ROMERO II
Special Problem Adviser

Accepted as partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE**.

DR. EULITO V. CASAS JR.
Chair, DNSM

# Acknowledgements

First of all I would like to thank the Lord for his guidance during the course of my research and for making this thesis possible. I would like to thank my family who served as my inspiration, and never failed to support me all throughout my studies. Thanks to ...

# Abstract

Although the abstract is the first thing that appears in the thesis, it is best written last after you have written your conclusion. It should contain spell out your thesis problem and describe your solution clearly.

Make sure your abstract fits in one page !

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In computer graphics, most images are typically stored as a sequence of dots in a rectangular grid (see Fig. 1.1). Each dot is called a pixel, a small part of an image that holds one specific colour, Photographs, also called natural images [15], are one of, if not the most, common images that are stored in this manner. Many digital forms of art or any graphics work, such as paintings, posters, and icons, are also stored the same. Digital images stored in this manner are called *raster images*. These images are stored in various image formats. The most commonly used formats are JPEG, GIF, BMP, TIFF, and PNG. Each have their pros and cons, from quality of the resulting image to the file size. Nevertheless, they all still accomplish the task of holding raster image data. Everything you see in the displays of devices such as laptops and mobile devices is a raster image. Computer displays are collections of pixels, in the common definition of a dot on the screen, which computers map images to to be able display them. This is the reason why **all** *displayed* images are raster images.

A positive aspect of raster images is their simplicity. As mentioned earlier, raster images consists of a grid of pixels (also called a pixel matrix in other literature [27]). This pixel grid can simply be assigned a combination of colour values to create an


Figure 1.1: When zoomed in enough, each individual pixel of a raster image is visible. Meme image obtained from `http://thesismemes.tumblr.com/post/73483120281`.

image. As such, working with raster images can be analogous to painting in the real world [6]. Given the right combinations of colours, we can produce natural images, i.e. photographs [15]. Intuitively, this means that we can store fine details in a raster image [23]. This is in contrast to *vector images*, which use a series of points and mathematical calculations to form lines and shapes. Vector images are unable to display lush colour depth and keep granularity, as found in raster images, as they use solid colours or gradients [6][7]. There are studies that have been conducted in improving and utilizing *gradient meshes*, a vector graphics primitive that allows for intricate colour gradients in regular quadrilateral meshes first introduced by Adobe Illustrator, to produce photorealistic vector images. However, as noted in the paper by Jian, S, Liang, L., Wen, F., and Shum, H., simple gradient meshes are insufficient to keep the fine details of images [9][23]. It is also important to mention that vector graphics, despite represented as mathematical calculations, are still converted to

raster format in a process called *rasterization* for it to be displayed on-screen, since many modern screens are raster displays [24].

## 1.1 The Problems of Raster Graphics

With all the pros raster graphics have, it does not mean raster graphics are not without their caveats. Raster graphics have their own disadvantages which could affect the image quality and their use.

Image at 64px by 64px.

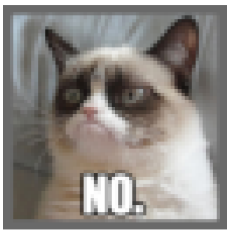


Image scaled to 350x350 pixels with nearest neighbour interpolation.

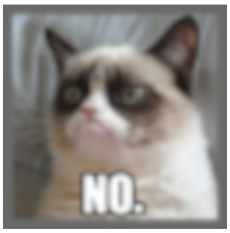


Image scaled to 350x350 pixels with bicubic interpolation.

Figure 1.2: Different interpolation algorithms will produce different results. As seen in the image, the quality will also differ, from an image looking blocky to an image looking blurry. Cat meme image obtained from `https://www.hercampus.com/school/uwindsor/school-thoughts-told-grumpy-cat-memes`.

Raster graphics are **resolution-dependent**. This simply means that raster images are in their highest quality in the resolution they are initially created in and attempting to scale it up will gradually degrade the quality of the image as the image size or resolution grows larger. Rasters only have a finite number of pixels. Increasing

the size of an image (also called upsampling [15] or single-image super resolution [29]) would entail moving the individual pixels into different locations depending on the scaling factor, the distance the individual pixels will be moved to horizontally and vertically. Upsampling will create empty pixels in between the shifted pixels when there is nothing done to substitute the empty pixels. This will create an unusable image [12]. We can utilize interpolation to fill these empty pixels with colour. *Classical* image upsampling approaches approximate colour and intensity values of these empty pixels are calculated based on the values of surrounding pixels, typically the shifted pixels. However, the specifics are dependent on the interpolation algorithm used in upscaling the images. Commonly used classical interpolation methods for resizing images include Nearest-Neighbour, Linear Interpolation, and Cubic Interpolation, which most, if not all, are readily available in popular raster image editing applications, such as Adobe Photoshop [2] and GIMP [1]. The results produced by these interpolation algorithms typically suffer from blurring of sharp edges and ringing artifacts due to the fact that the algorithms do not assume anything about the data [17][21]. See Fig. 1.2 for an example of blurring caused by scaling. There are adaptive image scaling techniques that consider image features such as edge information and texture to scale images with better quality than the classical methods. Examples of adaptive techniques are content-aware image resizing, seam curving, and warping-based methods. These techniques have their downsides as they take more computational time than their non-adaptive counterparts and may produce unexpected or even unsatisfactory results [21]. One of the latest advancements in upscaling images involves the use of artifical intelligence, specifically *neural networks*, as found in the works of AI Gigapixel and by Yang, C. Ma, C. and Yang, M.. They produce high quality upscaled images

and is a significant improvement over previous non-AI based upscaling techniques. However, they require high-end expensive hardware to produce results in the shortest amount of time possible. In the case of AI Gigapixel, a laptop with an integrated graphics card takes 20 minutes to produce a final high resolution image. For the work by Yang. C, Ma.C., and Yang. M., they utilized an Nvidia Titan Xp, a high-end GPU that costs $1,200 as of November 18, 2018 [4], to upscale a 520x520px image 2x, 4x, and 8x its size, and took 0.8s, 2.1s, and 4.4s, respectively, to complete [28][25].

## 1.2 Vectorization To The Rescue

Vectorization is the process of converting raster images into vector images [30]. Vector graphics uses collections of geometric primitives, such as points, curves, and points, and mathematical calculations to form an image [6]. Unlike raster graphics which uses a large pixel matrix (which will require large spaces without using proper image compression), vector graphics are able to smoothly scale to different resolutions, large or small, without any degradation in image quality [27][9]. This makes them **resolution-independent**. Vector graphics innately have this property due to their reliance on mathematics, instead of context-free pixel grids. Each primitive have their own mathematic formulas which, obviously, stay the same no matter what the size of an image is. As such, the primitives can simply be re-rendered whenever the image is scaled [6][7]. Vector graphics also allow for easier editing [15][23], as you only need to modify individual polygons, lines, and curves, instead of dealing with individual pixels like you normally would when using raster images editors.

Vectorization would often be done manually. In a study by Hoshyari, et. al. [15], each of their raster images, which includes icons and small graphic illustrations, take

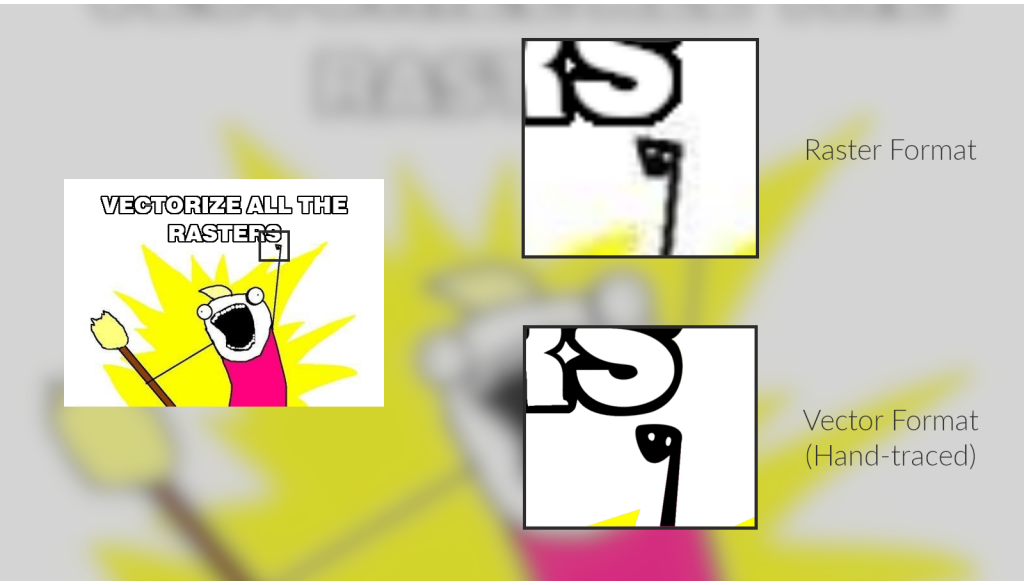


Figure 1.3: Zooming in closely at the same image, but one being in raster format (top right) and the other in vector (though hand-drawn; bottom left), quickly reveals the quality differences of both image formats. Raster images will show you individual pixels when close enough, but vectors will remain smooth.

.

30-45 minutes to be vectorized by an artist. More than 7 million man hours are being spent on vectorizing raster graphics in the United States every year, according to a survey in the PhD dissertation of J.R. Diebel entitled, "Bayesian image vectorization: The probabilistic inversion of vector image rasterization" [30]. Demand is, therefore, there for a robust vectorization algorithm.

Vectorization have been applied in many cases including, but not limited to, 2D maps and natural images. There are multiple methods that can be utilized in the vectorization of raster images. Their results differ from one method to another and even from one input to another, as many vectorization methods are fine tuned to specific inputs, such as those by Hoshyari, S., et. al. (semi-structured images) [15], Kopf, J. and Lischinski, D. (pixel art) [17], and Bessmeltsev, M. and Solomon, J. (line

drawings) [10]. Additionally, the ability to utilize GPUs for computational tasks has allowed parallelization of vectorization such as in the paper where a GPU was used to vectorize a video stream in real time [27].

Many vectorization methods target natural images. As noted by Hoshyari, S., et. al. many of these natural image vectorization methods utilize *image segmentation* to identify the portions of the image that will be converted into geometric primitives. These primitives are then filled with solid colors (e.g. such as what was done by Birdal, T. and Bala, E. [11]), gradient meshes (which are used in Adobe Illustrator and Corel CorelDraw [23][9]), and/or diffusion curves (such was the case in the paper by Xie, G. Sun, X., Tong, X., and Nowrouzezahrat, D. [26]). These vectorization methods produce differing results whose image qualities vary. Some results are as close as possible to the original raster image. Typically, gradient meshes and diffusion curves were utilized to achieve these results [26][23][9]. Others produce results with obvious colour segmentations, as seen in results that purely utilize solid colours (see [11] for an example).

Natural images are not the only raster images that are being vectorized. Images called semi-structured images are also candidates for image vectorization. *Semi-structured images* (SSIs) are images that consists of distinctly coloured regions and have well-defined boundaries [15]. Logos, cartoons, clip art, computer icons, and even simple graphical illustrations (such as flat 2D art) can considered to be semi-structured images. 60% of the 10 million images to be vectorized are semi-structured images [30]. This makes the demand for vectorizing these types of images evident. Various methods for vectorizing semi-structured images have been proposed by numerous papers. The common methodology of these proposals is that they attempt

to fit curves or Bezier splines on the boundaries of each region in SSIs and fill in the appropriate colour. However, each of these methods naturally have their own unique ways of fitting curves. In the work of Kopf, J., and Lischinski, D. [17], they use similarity graphs and additional intermediate steps in identifying the regions of an image, though their work is targeted at pixel art. Another paper by Yang, M., et. al. would directly optimize the shapes of individual Bezier segments connecting each boundary transition vertices to produce high fidelity vectorized images [15][30]. The work by Hoshyari, S., et. al. can be seen as a complement to the aforementioned paper. Their work utilizes human perceptual cues, primarily guided by Gestalt psychology, to produce vectorizations of semi-structured images that align much more closely to what viewers expect from a raster image [15].

An alternative method we can perform for image vectorization is to use machine learning via convolutional neural networks, a type of neural network that is well suited for image classification for their ability to learn various image features [14], to create vectorizations of semi-structured images by having the computer learn to produce Bezier curves of image region boundaries that align well with the expected vectorization. This is the method that this paper is proposing. A similar work, in that convolutional neural networks were for vectorization, to this is that of the paper of Simo-Serra, E., Iizuka, S., Sasaki, K., and Ishikawa, H. where they convert and simplify paper-and-pencil sketch drawings to vectorized images [22].

## 1.3 B-Splines

Many image vectorization techniques, such as those of Hoshyari, S., et. al., and Yang, M., et. al., involves the use of fitting curves to pixels that form a border for

a region in an image. The geometric primitive used in such approaches is the bézier curve [15][30][17]. A collection of bézier curves, called a b-spline, is also used for image vectorization. B-Spline is short for *basis spline*. B-splines allow for curves with higher complexity, such as those with curves resembling squiggly lines.

A $k$-degree B-spline curve is defined to be

$$C(u) = \sum_{j=0}^{n} c_j N_j^k(u)$$

where $c_j$ are the control points, $u$ is the non-decreasing knot vector $u = (u_0, \ldots, u_n)$ with $u_0$ and $u_n$ having a multiplicity of $k+1$, and a B-spline functions $N_j^k(u)$. The B-spline functions are referred to as the basis functions, from which the name of b-splines was obtained from. The basis functions are defined using the Cox-de Boor recursion formula [20] which is defined to be:

$$N_j^k = \begin{cases} 1 & \text{, if } u_j \leq u \leq u_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_j^k = \frac{u - u_j}{u_{j+k}} N_{j-1}^k(u) + \frac{u_{j+k+1} - u}{u_{j+k+1} - u_{j+1}} N_{j+1}^{k-1}(u)$$

Knots in B-splines determine the basis functions, which affects the shape of the B-spline curve [20]. The number of knots in $u$, disregarding the multiplicity of $u_0$ and $u_n$, is defined to be

$$|u| = k + n + 1$$

## B-Spline Approximation

Suppose that we are given a point sequence $p = (p_0, \ldots, p_m)$ with each point being $p_i = (x_i, y_i)$. According to the work by Laube, P., et. al. [18], we can compute the control points $c_j$ of the B-spline curve $C$ that will approximate $p_i$ by using the least square problem defined as

$$\sum_{i=0}^{m} |p_i - C(t_i)|^2 \to \min$$

with precomputed parameters $t_i, i = 0, \ldots, m$ combined in the parameter vector vector $t = (t_0, \ldots, t_m)$ and end points $C(t_0) = c_0 = p_0$ and $C(t_m) = c_n = p_m$. This will give us a normal equation

$$(N^T N)c = q \tag{1.3.1}$$

where N is an $(m-1) \times (n-1)$ matrix

$$N = \begin{pmatrix} N_1^k(t_1) & \ldots & N_{n-1}^k(t_1) \\ \vdots & \ddots & \vdots \\ N_1^k(t_{m-1}) & \ldots & N_{n-1}^k(t_{m-1}) \end{pmatrix}$$

and c and q are vectors defined to be

$$c = \begin{pmatrix} c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}, \; q = \begin{pmatrix} \sum_{i=1}^{m-1} N_1^k(t_i) q_i \\ \vdots \\ \sum_{i=1}^{m-1} N_{n-1}^k(t_i) q_i \end{pmatrix}$$

and

$$q_i = p_i - N_0^k(t_i) p_0 - N_n^k(t_i) p_m$$

for $i = 1, \ldots, m-1$. If there are no constraints for end point interpolation, then 1.3.1 reduces to

$$(N^T N)c = N^T p \tag{1.3.2}$$

The control points $c_j$ can be computed using 1.3.1 if

$$\sum_{l=1}^{m-1} N_i^k(t_i) N_j^k(t_i) \neq 0 \tag{1.3.3}$$

# Chapter 2

# Review of Related Literature

Image vectorization has been a long and well-researched field. One of the earliest works on the field dates back to 1982 with the paper by Jimenez, J. and Navalon, J. Their works have been focused on vectorizing digital images from natural scenes [16]. The techniques they have used, such as contour following, have been adapted in later works. Later works have made significant progress over their work and made use of hardware advancements such as GPU utilization for computational tasks.

The primary goal of vectorization is to support multiple resolutions without any loss in the original data. Vectorization is suitable for the task. However, various methods have been proposed that do not involve vectorization. In this chapter, we will explore the previous works done for vectorization and making images support larger resolutions.

## 2.1 Image Upscaling

Raster images are commonly used in representing images. Therefore, these types of images tend to be scaled to different resolutions. The process of scaling these types of images to higher resolutions is called *image upscaling* [15], and is the most common

forms of having raster images support larger resolutions. Naively scaling images would result in images with empty pixels between the shifted pixels, producing a mostly empty image. Thus, the empty pixels are coloured based on their position from, and colour and intensity values of their neighbouring pixels. This process is called *interpolation* [12]. Traditional interpolation algorithms can be classified into two types: Non-adaptive techniques, and adaptive techniques. Each of these types have their own method of resizing images and would produce different results, which may fit in with different objectives of different users [21]. Most recent works uses artificial intelligence to construct a scaled image with as little image quality loss as possible. These works uses neural networks, specifically convolutional and adversarial neural networks, to produce photorealistic upscaled versions of images with little to no distinguishable loss of image quality [28][25][19].

## Non-Adaptive Techniques

Non-adaptive interpolation techniques only scale images horizontally, vertically, or both, by a certain scaling factor, the amount in which the image will be scaled to. They do not assume anything about the underlying image data, except that it is band-limited [17]. This makes them computationally cheap [21], but also suffer from artifacts such as sharp edge blurring and ringing artifacts [17]. Nevertheless, these non-adaptive image interpolation techniques have been widely used in the industry and is standard across different raster image editing programs, such as Adobe Photoshop and GIMP [2][1]. The common non-adaptive interpolation techniques used are:

- Nearest Neighbour

- (Bi)linear Interpolation
- (Bi)cubic Interpolation

**Nearest Neighbour**

Nearest Neighbour is the simplest interpolation method to understand. At the high-level, nearest neighbour simply enlarges the size of each individual pixel by a certain factor.

At the lowest level, though this **may** differ implementation-wise, the algorithm will refer back to the original unscaled version to obtain the appropriate colours for each empty pixels in between the shifted pixels in the scaled image. Each empty pixel in the scaled version will map itself to a corresponding pixel in the original image. This is done by obtaining the square coordinates of the empty pixel and dividing the coordinates by the scaling factor. Note that the coordinate system in images starts its origin from the top left, instead of the bottom left. We may acquire a decimal as a result of dividing the coordinates, of which we will floor the values. The resulting coordinates can be mapped to a pixel in the original image. We will then use the mapped pixel's colour as the colour of the empty pixel we are calculating the colour for. The interpolated pixels can be seen as a set $I$, as quantified by the equation below.

$$I = \{i | i = C((\lfloor \frac{e_x}{s} \rfloor, \lfloor \frac{e_y}{s} \rfloor))\}, s > 1, e \in E$$

$s$ is the scaling factor, $C$ is the function that gets the colour of a pixel based on its coordinate, and $E$ is the set of empty pixels that are in between the shifted pixels when naively upscaled. The resulting image will look blocky since the pixels are just

enlarged. However, this will be ideal for pixel art [12].

### (Bi)linear and (Bi)cubic interpolation

Both of (bi)linear and (bi)cubic interpolations schemes are fairly similar to one another. The colours between the empty pixels between them are based on the values of the colours of the shifted pixels.

Each empty pixel acquire most of their colour from the nearest shifted pixel and least from the farthest shifted pixel. This process can be thought be thought of using graphs. In linear interpolation, the pixel colours can be viewed as the values of the y-axis, and the x-axis consists of the shifted pixels in the scaled image. The data points that are plotted are shifted pixels and are connected by a linear straight line. The colours of the empty pixels can then be computed by obtaining the y-value of the x-value of the pixel. Cubic interpolation is also the same process. However, the lines connecting the data points in the graph are cubic, i.e. smoother.

## Adaptive Techniques

Image upsampling can also be performed with adaptive techniques. Adaptive techniques do not naively increase the increase the number of pixels with some approximate in a scaled image. Rather, they use the information of the raster image to fill up the now empty pixels when scaling images. The features they consider include intensity value, edge information, and texture [21]. However, the results may not be what is expected of.

## Super Resolution Methods

Super resolution methods are processes of generating high-resolution images from low-resolution image input. Each method has different assumptions of the raster input and evaluation criteria. There are super resolution methods that target specific classes of input such as faces, scenes, and graphics artworks. Others are targeted towards generic input, in which the priors are based on image primitives such as edges and segments [29].


Figure 2.1: This a result of an raster image scaled to super resolution using A.I. Gigapixel. The left image is the original raster image. The top right is the normally scaled output. The bottom right is the output produced by A.I. Gigapixel. Image obtained from `https://topazlabs.com/ai-gigapixel/`.

Recent approaches to super resolution involves the use of artificial intelligence. One such approach is a commercial desktop application called A.I. Gigapixel by Topaz

Labs. Images can be scaled by the A.I. Gigapixel with great accuracy. It uses neural networks and the computational power of the GPU to produce super-resolution results. It can also run on a laptop. However, it takes 20 minutes to run on a laptop, and a few minutes on a high-end GPU [28]. Another approach based on artificial intelligence and neural networks has also been proposed by Wang, Y., Perazzi, F., and McWilliams, B. In their work, they use a progressive approach to scale images to super resolution with the use of generative adversarial networks.

## 2.2 Vectorization of Images

Various methods have been proposed throughout the years in the pursuit of supporting larger resolution without any reduction in image quality through vectorization. Typically, vectorization methods target specific inputs, such as natural images or artist drawn images, as certain methods are unsuitable for different inputs. Hoshyari, S., et. al. states that vectorizations targeted at natural images frequently produce inconsistent results when applied to artist-drawn imagery such as logos, and simple graphic illustrations [15]. No matter what the methods used are, they produce vector graphics that will vary in quality, photorealism, and artistic look (view [26], [9], and [11] for a comparison of the results of various vectorization methods).

Despite vector graphics providing a compact and alternative form of representing [27], it is important to remember that due to the inherent characteristics of vector primitives where certain fine details of raster images cannot be accurately represented in vectorized form, vector images will only be giving approximations of the details of images [23]. Nevertheless, finding the perfect balance for image level of detail and image vectorization is an endeavour that is left as an exercise for users [11].

Note that there are multiple possible vectorization outputs for a single raster image, with many outputs possibly being similar to one another and can be considered good enough for use. The *locally* "best" vectorization of some raster input among those possible vectorizations is determined by the vectorization method utilized.

Image vectorization can be dated as far back as the early 1990s, though experiments have started since 1982 at the earliest [16]. Commercial packages, both proprietary and open source, have image vectorization tools whose quality vary. The packages include Adobe Illustrator (Live Trace), Corel CorelDRAW (PowerTRACE), and Inkscape (based on Potrace [3]) [15]. Their wide adoption, though impressive, do not always immediately translate to quality vectorizations. The current available methods still have their own shortfalls and are still in active development. As a result, manual vectorization are still being performed in many industries that heavily require vectorizations [10].

## Natural Images Vectorization

Many raster images widely used today are photographs. They contain fine details and lush colour depth that are prominent in the real world. These photographs are also called natural images in digital image processing. Increasing their resolutions would entail using image upscaling techniques (see 2.1), especially the standard classical approaches. This would give the chance of producing low quality images [17] when super resolution methods are not utilized. Vectorization of these natural images are then an alternative to producing high quality higher resolution images.

Vectorization methods targeted at natural images consist the large body of work that deals with automatic vectorization. The core method of these algorithms involves the reliance on edge detection and/or region segmentations to cluster large quantities

of pixels together into larger regions [17]. These regions are then filled with either a solid color [11], or gradients via gradient meshes [23][9] or diffusion curves[26].

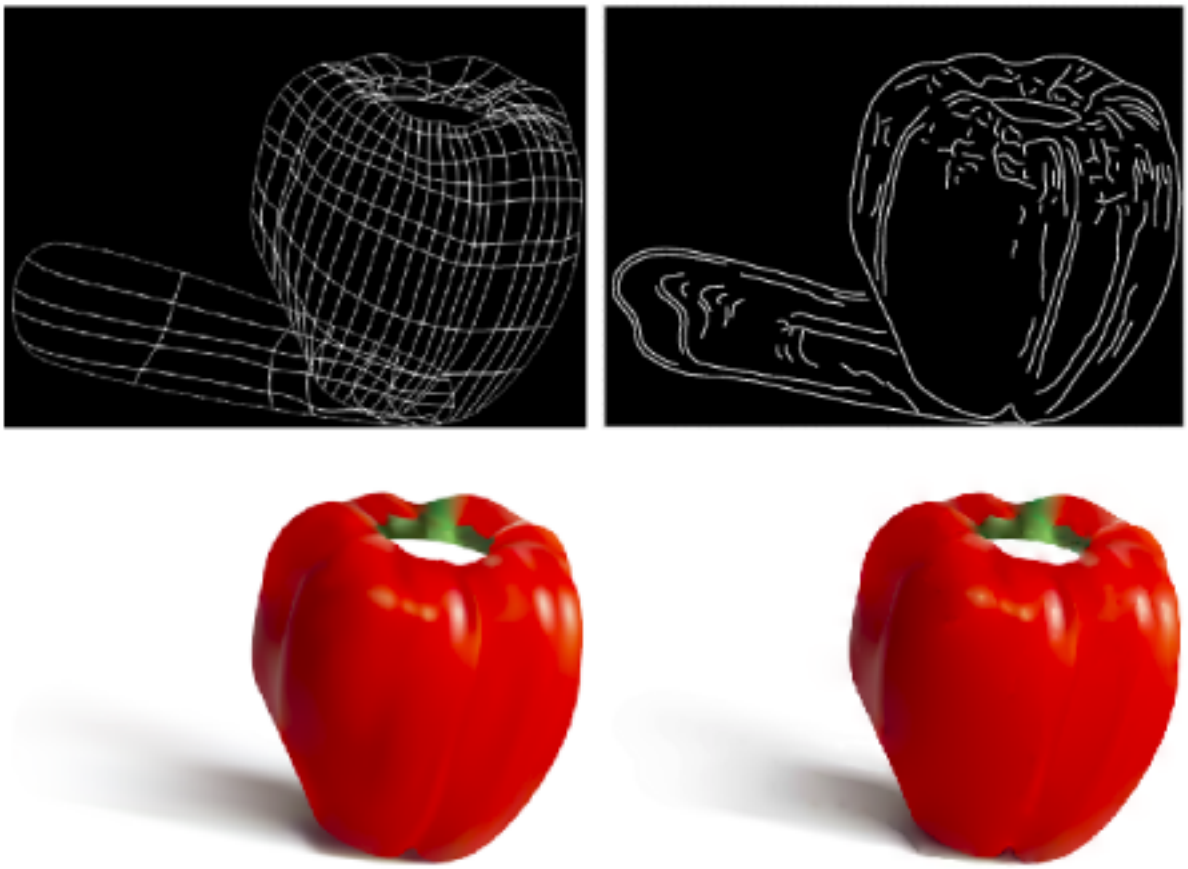

Figure 2.2: The left side shows a result of using gradient meshes when vectorizing natural image. The right side uses diffusion curves (and fewer curves) to vectorize an image.

One relatively recent work for vectorizing natural images uses hierarchical diffusion curves. This work was proposed by Xie, G., Sun, X., Tong, X., and Nowrouzezahrai, D.. Diffusion curves are defined using "curve primitives with colors set on both sides of it". In simplified wording, a diffusion curve is a curve with colours diffusing away perpendicularly to the curve. Their work revolves around tracing the curves in a

raster image in such a way that produces an accurate final vector output. The traced curves are only sparse. They are not restricted to natural images only as their method can be used in cartoon images as well. Their results are comparable to that of those that use gradient meshes. Observably, the results are impressive and matches the raster input. Their work, from their experiments, can take from a few seconds to 11 minutes, at the most. Note that they experimented on a PC with an 3.5 GHz Intel Core i7-3770K and an Nvidia Quadro 6000.

Figure 2.3: On the left is the raster input image. On the right is the vectorization output of the work done by Birdal, T., and Bala, E., when applied to the raster input image on the right.

Another work that vectorizes natural image targets is that of Birdal, T., and Bala, E.. Their vectorization approach does not produce a near-accurate vectorization of the raster input. Rather, their vectorization produces a flat art style output. Their approach uses segmentation to identify regions. The authors offer three different segmentation methods, all having an impact on the final vectorization output:

1. **Statistical Region Merging** (SRM) considers the image as an unknown scene. Colours are sampled in this method from a set of distributions that represent

image pixels. To summarize, the pixels are segmented based on the observed mean values of the pixels.

2. **Colour Structure Coding** (CSC) is a technique that is hierarchical and region-growing that consists of two phases that essentially groups pixels of similar colours together. The colour similarity is determined as similar when the Euclidean distance of two colours is smaller than a certain threshold.

3. **Graph-based Segmentation** is another approach that can be used and is similar to SRM. In their work, they used a segmentation technique proposed by Felzenszwalb, P., and Huttenlocher, D. [13].

Boundaries are then extracted from the segmentation outputs. The boundaries are then fitted with a spline. In their work, they used cubic Bezier curves to represent the vectorizations of those edges. The final final vector curves, which will now form shapes when connected, are filled with the approximation of the colour of the region in the raster image enclosed by the region [11].

Video is just a series of natural images. As such, it can be a target for vectorization. The work of Xiong, X., Feng, J., and Zhou, B. deals with vectorization of video. They are not vectorizing video as a post-processing step. Rather, they are vectorizing video in real time. They made it possible through the use of GPUs. Graphics processing units (GPUs, for short) have been in use to accelerate processing of tasks. Their parallelized nature, since they consist of small, efficient cores that handle multiple tasks that run in parallel, allow for faster computations [5]. In vectorizing video in real time, they first detect the boundary pixels of each line. These boundary pixels are pixels that determine the start and end of some region in a line. This stage essentially

segments the current video frame. This process is done via a scanline approach. Due to the relative independence of each line during this stage, parallelism can be utilized to reduce the amount of time needed to perform this stage. Pre-countouring is then performed to identify the relationship of the regions in a line to the regions in adjacent lines. This is done to identify how regions in one line is connected to larger region in the video frame. This stage is also parallelized since the relationship is identified only being determined between two lines at a time. Pre-contouring will now lead to contouring which identifies the different regions in a video frame. The edges of each region is computed a loop from which a vectorization is will be performed upon. This is done with the CPU due to the frequency of memory access necessitated by this step. The loops are then simplified and vectorized with a set of line segments through a process similar to that of Active Countour Modelling, where each loop is processed with a divide-and-conquer strategy. The independence of each loop from one another allows for this step to be highly parallelized. Their experimentation shows that they were able to vectorize videos with an average video frame rate of 48 fps, and vectorize a 2500x1800 resolution typed page in just 40 ms. They have performed their experimentation using CUDA with a PC that has a 2.5 GHz quad-core CPU (brand unspecified), 2.75 GB RAM, and an Nvidia GeForce GTX 260+ [27]. Their work is a demonstration that GPU utilization for parallel execution is beneficial to speeding up vectorization. However, due to their use of line segments as the vector primitive for their work (unlike other works such as those of Kopf, J., and Lischinski, D. [17], and Xie, G., Sun, X., Tong, X., and Nowrouzezahrai, D. [26]), the result may not be what is expected of the final vectorization.

## Vectorization of Semi-Structured Images and Artworks

Artwork, especially its subset, semi-structured images, is undeniably widely used around the globe. They are used to convey information, and express ideas. Both purposes would infer the necessity to maintain the high, or at least good enough quality images of those artworks to properly fulfill their tasks. Vectorization of these images would ensure that the quality of the image is kept at any resolution possible without any degradation. In the pursuit for quality vectorizations, many papers have been proposed that target specific inputs (such as pixel art, or small resolution images) and give out varying vectorization results.

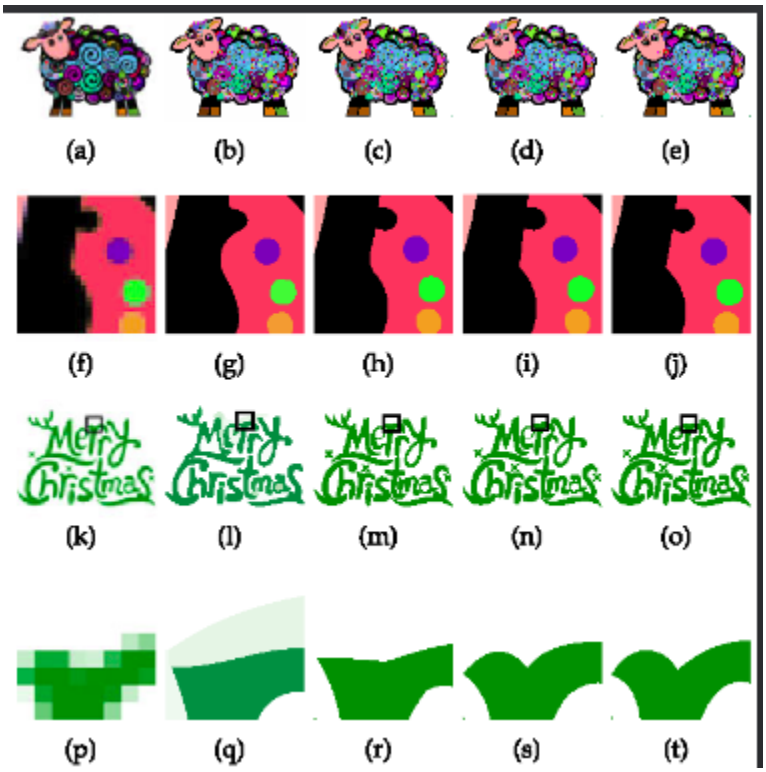


Figure 2.4: Vectorization performed using various methods. From left to right: the raster input, Adobe LiveTrace's result, Vector Magic's result, and the result by Yang., M. et. al.

A framework that optimizes bezigons, closed paths composed of Bezier curves, to match their raster counterparts as much as possible was proposed by Yang, M., et. al. as a vectorization process. Their work takes bezigons as input, of which they obtain either using pre-existing vectorization methods or extracting them from the raster image by segmenting the image into a set of regions then fitting piecewise Bezier curves on the region boundaries.

The input is called the *initial bezigons*. These initial bezigons are then optimized to reflect the raster image as close as possible. In optimizing the bezigons, they use a non-linear optimization algorithm (to be referred as *NLOAs* from hereafter) such as NEWUOA and conjugate gradient. Being an optimization-centered process, their process requires a method to evaluate their optimizations. Evaluating the vectorization output would involve getting the rasterization of the output and comparing it with the original raster image. Since there are various rasterization functions, a good enough rasterization function that works well with NLOAs is required. Using discontinuous and piecewise rasterization functions yields poor results during optimization. As such, a continuous rasterization function is required. Yang, M., et. al. chose to use a rasterization approach that utilizes a hierarchical Haar wavelet representation.

The key component, which we view as a primary contribution of their work, is the energy $E$ used to evaluate the vectorized outputs. $E$ is the sum two parts: (a) the data energy, and (b) the prior energy. The data energy refers to the distance of the input raster image and the rasterization of the vectorization output. The prior energy refers to the severity of unreasonable bezigons. These bezigons would typically fall under one of the following categories, as per the work's authors intensive experimentation: (a) self-intersection, (b) false corners with small angle variations, (c) short handles,

and (d) twisted sections. A larger $E$ would indicate a more inaccurate vectorized output. As such, optimization of bezigons, as it is their work, would be to minimize $E$.

Basing from their experiments, they produce high quality vectorizations that are superior to the results of Vector Magic and Adobe LiveTrace. However, the frameworks yields poor results when vectorizing noisy or low-resolution inputs. Assuming that the experimental results be any sign for the actual theoretical speed, the execution time of this ranges from 10 seconds to 10 minutes, depending on the complexity of the shapes being vectorized. It is important to note that their implementation is not optimal and is written in Python, which provides a significant overhead. An implementation in a static, compiled language is expected to yield faster optimization speeds [30]. Basing from personal experience in hand vectorization, this framework resembles manual vectorization in that a bezigon is initially created that does not immediately match the raster image. The bezigon is later adjusted to fit the raster image boundaries.

Being that they have proposed a framework, their work can be viewed as an optional and/or complimentary post-processing step to tweak pre-vectorized accuracy-ambiguous images, rather than a complete alternative or replacement of pre-existing and well-established vectorization methods, such as Adobe LiveTrace, and Potrace. At least one work, specifically that of Hoshyari, S., et. al., has already stated that their work is complimentary to this [15]. The only disadvantage to this framework is that it requires multiple iterations before settling on an optimal vector solution, which, consequentially, require some time to complete. It is also unclear whether the framework becomes more computationally expensive as the raster image size becomes

larger as their paper does not indicate whether it is so or not.

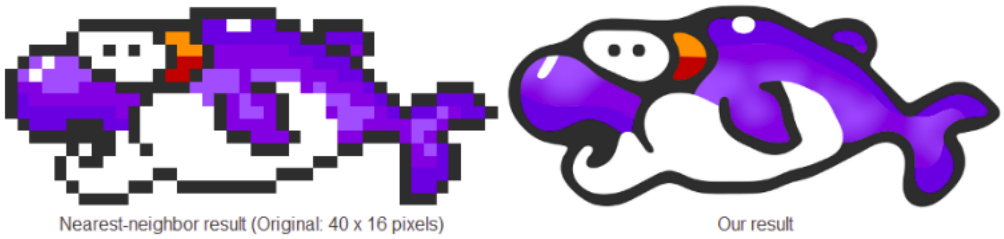


Figure 2.5: One of the results retrieved from using the vectorization method for pixel art as proposed by Kopf, J., and Lischinski, D. The left image is the pixel scaled using nearest-neighbour. The right iamge is the result done by their work.

Another form of digital art that is a target of vectorization is pixel art. Pixel art is a form of art where the level of detail is limited to the pixel level. This type of art tends to be blocky, and is reminisce of the art style of retro-era games. The work of Kopf. J, and Lischinski, D. specifically targets pixel art. As with other vectorization methods, their work detects edges of which they fit curves to create a final vector result. Due to the inherent property of pixel art, their work is met with a few non-trivial challenges. The authors have identified four:

1. Every pixel matters, since a pixel can possibly represent a different feature given its colour is different enough from its neighbours, Additionally, each pixel can be viewed as an approximation of certain details in the overall image.

2. Pixels in a pixel art look connected at original scale, but disconnected when magnified.

3. The reduced level of detail in pixel art gives way to locally ambiguous configurations, in which it is unclear which pixels are part of which features.

4. Lastly, jaggies, collection of pixels that we perceive as forming jagged lines, in pixel art make differentiate features from pixelization artifacts hard.

The proposed vectorization method of Kopf, J., and Lischinski, D. would first involve creating a similarity graph that would connect similar pixels with one another via edges. In this case, similar pixels would refer to pixels with similar colours. It should be noted that their work has some basis on Gestalt psychology in order to closely reflect what a human will imagine the final vector output to be.

In integrating Gestalt psychology, they utilized heuristics to help identify which nodes in the similarity graph are to be connected or not. Once a similarity graph has been constructed, a Voronoi diagram, which consists of Voronoi cells, is made based on the similarity graph. The diagram now represents a rough shape of the object. The connected visible edges of the diagram, simplifyingly, are converted into quadratic B-spline curves.

Related to the work done by Yang, M., et. al., optimization of the resulting quadratic B-spline curves are performed. Optimization involves the "minimization of a sum of per-energy nodes". A node, in this context, refers to a single endpoint of a curve. The energy $E$ of each node $i$ is computed to be the sum of both smoothness and positional terms, $E_s^{(i)}$, and $E_p^{(i)}$, respectively.

$$E^{(i)} = E_s^{(i)} + E_p^{(i)}$$

The smoothness term, $E_s^{(i)}$, measures the absence of curvature in the curve region influenced by node $i$. On the other hand, the positional term, $E_p^{(i)}$, measures the distance a control point node has moved away from their initial positions. The latter term is a prerequisite to prevent objects from changing too often. The control points

are allowed to move freely within a relatively small region around their origin location, but are penalized when they deviate too far away. Not all nodes are optimized as a random walk is performed per iteration do identify which nodes will be optimized.

The resulting vector image, after optimization, may have nodes that were not constrained. As such, these unconstrained nodes are computed a new location using harmonic maps. This tweaking results in distortion of cells in the Voronoi diagram. Additionally, corners in the pixel art are detected via heuristics to make sure these corners are given sharp lines, rather than a smooth curve.

This work has yielded excellent results that closely resembles that of the original pixel art image. Comparing this with the results of previous works for vectorization, especially those with an inclination towards pixel art, this work has produced results that are comparable or superior (for example, when comparing against Adobe Live-Trace). The published execution time of the work opens it up for the possibility of use in vectorizating videos of retro games, especially when done in real time. The work, however, is limited hand drawn pixel art. Downsampled pixel art tend to look anti-aliased, which makes these inputs closer to natural images — inputs this work was not designed for. The results produced by this work will not always agree with human perception [17].

When viewed at a naive resolution independency angle, it would seem that this work is a step in the wrong direction as a simple nearest neighbour scaling algorithm can be applied to pixel art to support higher resolutions, as the algorithm is compatible with this type of input. However, when looked from a far enough distance, pixel art is viewed to be smooth and continuous. Simply upscaling such images would have it retain its blocky nature. This work by Kopf, J. and Lischinski, D. offers a way to

vectorize pixel art to **not only** support higher resolutions, but as well as providing a *smoother*, not blocky, output that closely matches the perception of the figure.

Humans naturally have an expectation of the vectorization output of a certain image. Hoshyari, S., et. al. proposed a vectorization method that exploits human perception to produce accurate vectorizations of images. Their work is primarily driven by Gestalt psychology in the attempts to create a vectorization that is close to human expectations. The authors have identified four principles that they integrated with their work:

1. **Accuracy**, which states that we expect the rasterization of the vector output to be the same pixelized content as the original raster image.

2. **Continuity**, which states that despite the blocky nature of pixel art, they are perceived as a smooth line or curve and not necessarily full of corners.

3. **Simplicity**, which states that viewers have a preference for simpler geometric interpretations of the raster image.

4. **Closure**, which, aside from being a part of human breakups (sometimes not at all), states that human observers would mentally segment objects at points, with a negative curvature minima, with a pair point opposite of them with a negative minima.

Unlike previous vectorization methods, this work utilizes machine learning in one stage of the work to help generate vectorizations. Machine learning is used for detecting corners in a raster image. Corner detection is essentially to allow for accurate vectorizations especially at, well, corners. In the work, they have used random forests,

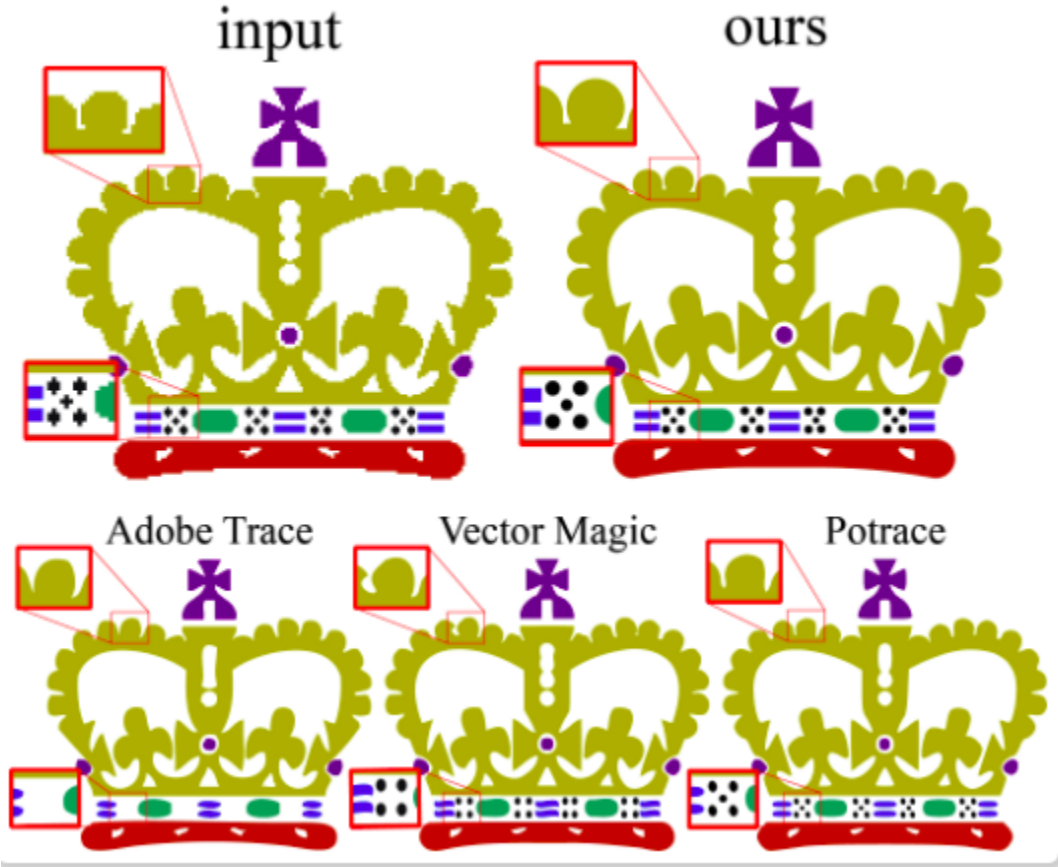


Figure 2.6: Comparison of the vectorizations of various vectorization methods. "ours" in this figure refers to the work done by Hoshyari, S., et. al.

trained through supervised learning with a training/validation data set of 158 raster images (with the training images being 8/16/32/64/128/256 pixels on each size), to detect corners. In their experiments, random forest classifiers have produce more accurate corner detections than neural networks. After corner detection, they would obtain an initial set of possible corners and an initial vectorization performed with the use of a version of the fitting algorithm proposed by Baran, et. al. [8].

This set is expected to possibly contain corners that do not match with human perception. As such, an iterative corner removal process is employed to prune away

corners that do not match with human expectations. In this stage, each corner is systematically removed to evaluate if it is a corner that aligns with human perception in accordance with the four aforementioned principles. It is removed if the resulting vectorization produced without it is better than the one with it. The iterative corner removal process completes once all corners have been evaluated. The final vectorization is then regularized.

Regularization involves locating characteristically similar, but not identical, splines and primitives and enforcing them to be identical. These located primitives and splines are adjusted to have the same characteristics. This process produces a simplified vectorization output, in line with the simplicity principle utilized in the work. According to their experimental results, their results are superior to Adobe LiveTrace, Vector Magic, and Potrace. Observers preferred their results than those generated by the aforementioned vectorization methods. They also find the results on par with manually produced vectorizations. Despite the results, however, their work becomes computationally expensive as the input size becomes larger. Additionally, their work is designed only to vectorizing clean quantized images. Anti-aliased and noisy inputs will likely generate unexpected final vectorization result. Filtering to quantize those types of inputs would produce artifacts that would impact the vector result. Future work includes addressing these issues [15].

The proposed corner detection stage in this work can be improved, possibly by using a different neural network or classification method, to reduce the necessity to use an iterative corner removal process and may be used as a necessary step in future vectorization methods. The emphasis on the utilization of Gestalt psychology principles to produce results near to human expectation is also a component that

must be given note and should be considered by future works.

# Chapter 3

# Statement of the Problem

Raster images are composed of a pixel matrix. This makes their data representation simple. However, this reduces the amount of detail and quality they have. This limitation is clearer when scaling raster images. This motivates the use of an alternative form of representing images. One form is vector images, which uses mathematical equations to represent an image. The process of converting a raster image to a vector image is called *vectorization*. Semi-structured imagery, such as those used in graphic designs, is one of the classes of images that would benefit from vectorization. This process will allow such images to be easily scaled without sacrificing quality nor detail.

There have been numerous works that tackles image vectorization for semi-structured images. Many of these works primarily use a curve optimization algorithm such as variants of NEWUOA and conjugate gradient. A machine learning approach has been used in one of the works, but as a preprocessing step only [15]. Deep learning have been used to solve problems in multiple domains such as natural language processing (NLP), object detection, and playing board games. No other known work has applied deep learning as a core step in image vectorization. This study will deal with such application.

# Chapter 4

# Objectives

This study is primarily aims to apply deep learning via artificial neural networks to the problem of vectorizing semi-structured imagery. However, there are still some key objectives that this study seeks to accomplish:

1. To develop an approach that considers Gestalt psychology.
2. To evaluate the effectiveness of using deep learning for image vectorization.
3. To evaluate the accuracy of results obtained from the proposed approach to the target raster inputs.
4. To evaluate and compare the results of the proposed approach to that of previous semi-structured image vectorization methods.
5. To evaluate and compare the speed of the proposed approach compared to previous semi-structured image vectorization methods.

# Chapter 5

# Proposed Methodology

The proposed methodology will be based on the frameworks proposed by Hoshyari, S., et. al. [15], Yang, M., et. al. [30], Xiong, X., et. al. [27], and Laube, P., et. al. [18]. Additionally, human perception will also be taken account. Thus, the Gestalt psychology principles of accuracy, simplicity, continuity, and closure, as taken from Hoshyari, S., et. al., will be taken into account as well.

## 5.1 Image Preprocessing

Before a raster input image can be fitted with curves, it must preprocessed to simplify the vectorization procedure and align it with .

### Corner Detection

The first step in the approach is detecting the corners in the input image. This is an important step as it will allow us to enforce the simplicity principle in Gestalt psychology and make sure the resulting vectorization be $C^0$ continuous should the raster input be as such.

This stage will be based from the corner detection classifier of the work by Hoshyari, S., et. al. [15]. A random forest classifier is used in the said work. Supervised learning is used as corners are manually annotated. Annotated corners will not be specific pixels. Rather, corners will be between at least two pixels. Training data is available publicly provided by the researchers. However, such data is limited only to quantized data (i.e. aliased data). The target raster input is expected to be anti-aliased data. As such, the training data will have to be built from scratch to support anti-aliased data.

## Region Segmentation

In line again with the simplicity principle, the input image must be divided into regions. This will result in simpler curves being used in the final vectorization output. The additional benefit of segmenting the input into multiple distinct regions is the possibility for parallelism to be used in the vectorization approach. Since each region is distinct and independent from one another, multiple regions can be vectorized at the same time. Thus, speeding up the vectorization process.

Due to the nature of semi-structured imagery where each region will only contain a single colour, a scanline-based approach can be used in this stage. The scanline algorithm will be based off of the scanline algorithm used for boundary pixel detection in the work of Xiong, X., et. al..

For each line $l_i$, where $0 \leq i < h \mid h$ is the height of input image, in the raster input, each pixel $p_i$ will be assigned to a pixel set $r_{ij}$, where $j$ is the index to a set in $l_i$. Each pixel set will contain horizontally adjacent pixels that have the same or near-same colours. We include pixels whose colours are within a certain threshold $k$ from the colour of the pixels that is most prominent in the set. This threshold is

necessary due to the fact that certain parts of a regions may contain an anti-aliased pixel. Each $l_i$ will have a pixel set vector $r_i$ containing all pixel sets of $l_i$:

$$r_i = (r_{i0}, r_{i1}, \ldots, r_{i(j-1)}, r_{ij})$$

Consequently, a region vector $r$ will contain all pixel set vectors.

$$r = (r_0, \ldots, r_i)$$

Note that there is an opportunity to utilize parallelism, as shown in the work of Xiong, X., et. al. [27], during this step due to the independent nature of every line in the raster input.

Once we obtain all the pixel sets for each line, we iterate through $r$, $(h-2)$ times. For each iteration, we process $r_i$ and $r_{i+1}$. If there are any $r_{ij_\alpha}$ whose pixels are vertically adjacent and have the same colour (or within $k$) to another pixel set $r_{(i+1)j_\beta}$, then those two pixel sets are merged into one. By the end of the iterations, we have obtained a set of regions that we can individually vectorize.

## 5.2 Curve Approximation

The core step of the proposed approach is curve approximation. This stage fits curves to the region boundaries of the raster input. This stage is based on the work by Laube, P., et. al. on curve approximation on point sequences using deep learning [18].

### Point Sequence Generation

The work of Laube, P., et. al. takes a point sequence as input. As such, for this proposed approach, we must generate point sequences from the region we will be

vectorizing. For every region we ought to vectorize, we treat the center of boundary pixels and the detected corners of each region as a point sequence. However, we must also take into account the fact that certain portions of a region may be a corner where the curves in such segment would have $C_0$ continuity. As such, the point sequence we generate must take into account corners. This would implore us to take note of the following cases during point sequence generation:

1. For regions with no corners, a random pixel will be selected as both start and end point of the point sequence. The expectation is that there will be no difference in the resulting curve from choosing a different start and end point.

2. For regions with a single corner, the corner point will be selected as the start and end point of the point sequence. This is to ensure that the resulting vector output will have a corner at that point.

3. For regions with two or more corners and assuming $n$ is the number of corners, the point sequence will be divided at those corners into separate point sequences. This will result in $(n + 1)$ new point sequences. Each new point sequence will have their start and end points be the corner points they are adjacent to.

Each point sequence will then be passed to the next stage to be parametrized and have a curve approximated for.

## Point Sequence Curve Approximation

The point sequences obtained from the previous step are now to be fitted with curves, specifically B-splines. As provided by the framework by Laube, P., et. al., two neural networks will be used in this stage and some preprocessing will be performed on

the point sequences. The two neural networks are a point parametrization network (PPN), which approximates parametic values to point sequences, and a knot selection network (KSN), which predicts new knot values for knot vector refinement.

**Sequence Segmentation**

The input point sequence must first be split. This is to ensure that real data and training data match in terms of complexity. Let us define a function $\hat{k}(p)$ that measures the complexity of a point sequence $p$, where $k_i$ is the curvature at point $p_i$, given its total curvature:

$$\hat{k}(p) = \sum_{i=0}^{m-1} \frac{(|k_i| + |k_{i+1}|)||p_{i+1} - p_i||_2}{2}$$

A point sequence $p$ is split into point sequence segments $p^s, s = 1, \ldots, r$ at the median, if $\hat{k(p)} > \hat{k_t}$ for a threshold $\hat{k_t}$. This $\hat{k_t}$ will be set, as per the original authors have done, to the 98th percentile of $\hat{k}(\cdot)$ of the training set. This process is performed $r - 1$ times, until each $p^s$ satisfies $\hat{k(p)} > \hat{k_t}$.

**Sub/Supersampling and Normalization**

To be able to approximate parametric and knot values using the PPN and KSN, the number of points per segment $p^s$ must equal the input size $l$ of the aforementioned networks. As such, all segments $p^s$ are either subsampled or supersampled.

If a segment $p^s$ has a number of points greater than $l$, then $p^s$ is subsampled. This process involves drawing points in $p^s$ such that the drawn indices $i$ are equally distributed and include the first and last point. If, on the other hand, the number of points in $p^s$ is less than $l$, then *temporary* points are linearly interpolated between consecutive points $p_i^s$ and $p_{i+1}^s$. This interpolation is performed until the number of

points equal $l$.

The sampled segments are then normalized to $\bar{p}^s$, which consists of the points

$$\bar{p}_i^s = \frac{p_i^s - min(p^s)}{max(p^s) - min(p^s)}$$

where $min(p^s)$ and $max(p^s)$ are the minimum and maximum coordinates of $p^s$ respectively.

**Parametrization of Point Segments**

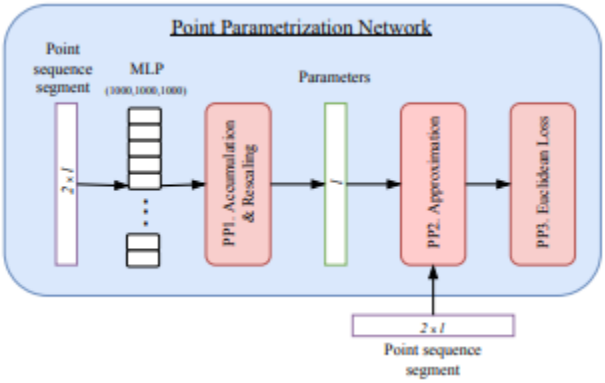


Figure 5.1: The network architecture of the Point Parametrization Network (PPN)
.

The PPN will be responsible for parametrization of point segments. For every $\bar{p}^s$, the PPN generates a parametrization $\bar{t}^s \subset [0, 1]$, which will be rescaled to $[u_{s-1}, u_s]$ and adapted to sampling of $\bar{p}^s$.

In supersampled segments, the parameters $t_i^s$ of temporary points are simply removed from $\bar{t}^s$. In a subsampled $p^s$, for every point $p_i$ that was removed from the segment, a parameter $\bar{t}_i$ is inserted to $\bar{t}^s$.

$$t_i = t_\alpha^s + (t_\beta^s)\frac{chordlen(p_\alpha^s, p_i)}{chordlen(p_\alpha^s, p_\beta^s)}$$

*chordlen* is the length of the polygon defined by a point sequence. In the sub-sampled segment, with parameters $t^s_\alpha$ and $t^s_\beta$, $p^s_\alpha$ and $p^s_\beta$ are the closest neighbours of $p_i$.

The initialization of the parametric step requires an initial knot vector. We first define $u_0 = 0$ and $u_n = 1$. For each segment (except the last one), one knot $u_i$ is added.

$$u_i = u_{i-1)} + \frac{chordlen(p^s)}{chordlen(p)}, i = 1, \ldots, r-1$$

This yields a start and end knot for every point sequence segment.

**The PPN Architecture** The PPN, as stated earlier, takes in an input of segments $p$, which can be written as $p = (x_0, \ldots, x_{l-1}, y_0, \ldots, y_{l-1})$. The parameter domain is defined as $u_0 = t_0 = 0 and u_n = t_{l-1} = 1$. For a sequence of points $p$, a parameter vector $t = (t_i)_i$, is defined as $t_i = t_{i-1} + \Delta_{i-1}$. The task of the PPN is to predict missing values $\Delta = (\Delta_0, \ldots, \Delta_{l-2})$ with

$$\Delta_s ubi > 0, i = 0, \ldots, l-2$$

such that $t_0 < t_1$ and $t_{l-2} < t_{l-1}$. We apply a multilayer perceptron (MLP) to the input data $p$, yielding as output a distribution for parametrization $\Delta^{mlp} = (\Delta^{mlp}_0, \ldots, \Delta^{mlp}_{l-2})$ of size $l-1$.

The PPN further contains additional layers PP1, PP2, and PP3, which will be discussed next.

**PP1. Accumulation and Rescaling** The output $\Delta^{mlp}$ is used to compute a parameter vector $t^{mlp}$ with $t^{mlp}_0 = 0$ and

$$t_i^{mlp} = \sum_{j=0}^{i-1} \Delta_j^{mlp}, i = 1, \ldots, l-1$$

Since $t_{l-1}^{mlp}$ is usually not 1, rescaling $t^{mlp}$ yields the final parameter vector $t$ with

$$t_i = \frac{t_i^{mlp}}{max(t^{mlp})}$$

The MLP layer in the PPN uses a softplus activation function defined to be:

$$f(x) = ln(1 + e^x)$$

**PP2. Approximation** B-spline curve approximation is included directly into the PPN as a network layer. The input points $p$ and their parameters $t$ are used for an approximation with knot vector $u = (0, 0, 0, 0, 1, 1, 1, 1)$ for $k = 3$. The approximation layer's output $p^{app} = (p_0^{app}, \ldots, p_{l-1}^{app})$ is the approximating B-spline curve evaluated at $t$.

**PP3. Euclidean Loss** A loss function, which is a Euclidean loss function, is to be used in the PPN. The Euclidean Loss function is defined as

$$\frac{1}{l} \sum_{i=0}^{l-1} ||p_i - p_i^{app}||_2 \tag{5.2.1}$$

**Parametrization Refinement**

In some cases, the approximated parametrization of a $p^s$ may have errors. The approximation error of a $p^s$ is computed by using the Hausdorff distance to the input data $p$.

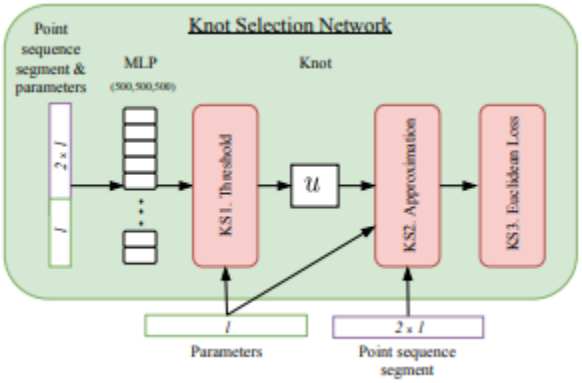


Figure 5.2: The network architecture of the Knot Selection Network (PPN)

.

The segments $p^s$ that have a large approximation error are computed a new knot using the KSN. For $\bar{p}^s$ and $\bar{t}^s$, the KSN generates a new estimated knot $\bar{u}_s \in [0, 1]$. The new knot $\bar{u}_s$ is mapped to the actual knot value range $[u_{s-1}, u_s]$ by

$$\tilde{u}_s = u_{s-1} + \bar{u}_s(u_s - u_{s-1})$$

Instead of $\tilde{u}_s$, the parameter value $t_i$ closest to $\tilde{u}_s$ is inserted into $u$. $u$ can be further refined until the desired curve approximation error threshold is satisfied.

**The KSN Architecture** This stage utilizes a KSN, as mentioned earlier. The KSN predicts a new knot $u$ to the interval $(0, 1)$ for a given segment $p$ and parameters $t$, of which were previously approximated by the PPN. This network uses an MLP, which transforms the input to a single output value $u^{mlp}$. The RELU function is used as the activation function for the MLP, except for the output layer where the Sigmoid function is used instead.

Similar to that of the PPN, the KSN has three additional layers: KS1, KS2, and KS3.

**KS1. Threshold Layer** The new knot $u$ computed has to satisfy the following conditions: $u \in (0, 1)$, $t \cap [0, u] \neq \theta$, and $t \cap [u, 1] \neq \theta$. Satisfying these conditions will will require us to use a threshold layer which maps $u^{mlp}$ to

$$u = \begin{cases} \epsilon & , \text{if } u^{mlp} \leq 0 \\ 1 - \epsilon & , \text{if } u^{mlp} \geq 1 \\ u^{mlp} & , \text{otherwise} \end{cases}$$

Introducing a small $\epsilon = 1e - 5$ makes sure that the knot multiplicity at the end knots stays equal to $k$.

**KS2. Approximation** Approximation in the KSN is generally similar to that of PPN. The only different is that of the knot vector. The knot vector in the KSN is defined to be $u = (0, 0, 0, 0, u, 1, 1, 1, 1)$. For backpropagation, the derivative of the B-Spline basis functions with respect to $u$ is required.

**KS3. Euclidean Loss** The loss function for the KSN is the same as that of the PPN. See 5.2.1.

### Network Training

The training of the PPN and KSN will be based from the work of Laube, P., et. al. [18]. The input size of the network will be $l = 100$.

The data set generated by the original authors consists of 150,000 curves. This data was synthesized from B-spline curves. Random control points $c_i$ were generated using a normal distribution $\mu$ and variance $\delta$ to define cubic ($k = 3$) B-spline curves with $(k + 1)$-fold end knots and no interior knots. The $y$-coordinates are given the configuration: $\delta = 2$ and $\mu = 10$. For the $x$-coordinates, $\delta = 1$ and $\mu = 10$ are used for

the first control point. All consecutive points have $\mu$ increased by $\Delta\mu = 1$. Curves with self-intersections are discarded, because the sequential order of their sampled points is not unique, and point sequences are usually split into subsets at the self-intersections. Smaller $\delta$ for the x-coordinates of control points reduces the number of curves with self-intersections. To closely match the target input as much as possible, we also include curves that have been manually fitted to raster images. These curves can be obtained from vector images available online.

For each curve, $l$ points $p = (p_0, \ldots, p_{l-1})$ are sampled. These curves then to have increasing x-coordinates from left to right. As such, index-flipped versions of the point sequences of the dataset are added, resulting in 300,000 point sequences. 20% of the sequences are used as test data in the training process.

The PPN is trained first since the KSN requires point parametrizations $t$, which is obtained from the PPN. After training, the PP2 and PP3 layers are discarded and PP1 becomes the output layer of the PPN. The parametric values $t$ are computed for the training dataset by applying the PPN and train the KSN on the combined input. After training, KS2 and KS3 are discarded, with KS1 becoming the network output layer. The MLPs of the PPN and KSN will consist of three hidden layers with sizes (1000, 1000, 1000) and (500, 500, 500) respectively. Dropout is applied to the MLP layers. The network is trained using the Adam optimizer.

## 5.3 Region Colouring and Merging

Once the curves have been approximated, the vectorization of the region will be filled with the colour prominent in the raster version of the region. The regions will be plotted unto their locations in the original raster input. Once all the regions have

been plotted, they will be grouped into a single vectorization. This will now be the vectorization output of the raster image.

# Chapter 6

# Describing How You Validated Your Approach.

# Chapter 7

# Stating Your Results and Drawing Insights From Them.

# Chapter 8

# Summarizing Your Thesis and Drawing Your Conclusions.

# Appendix A

# What should be in the Appendix

What goes in the appendices? Any material which impedes the smooth development of your presentation, but which is important to justify the results of a thesis. Generally it is material that is of too nitty-gritty a level of detail for inclusion in the main body of the thesis, but which should be available for perusal by the examiners to convince them sufficiently. Examples include program listings, immense tables of data, lengthy mathematical proofs or derivations, etc.

# Bibliography

[1] 6.22.scale image. https://docs.gimp.org/en/gimp-image-scale.html.

[2] Photoshop image size and resolution. https://helpx.adobe.com/photoshop/using/image-size-resolution.html{\\#}resampling.

[3] Potrace - inkscape wiki. http://wiki.inkscape.org/wiki/index.php/Potrace.

[4] Titan xp graphics card with pascal architecture — nvidia geforce. https://www.nvidia.com/en-us/titan/titan-xp/.

[5] What is gpu-accelerated computing? - definition from techopedia. https://www.techopedia.com/definition/32876/gpu-accelerated-computing.

[6] Raster vs vector. https://vector-conversions.com/vectorizing/raster_vs_vector.html, n.d.

[7] Raster vs. vector graphics: What's the difference? https://pixellogo.com/blogs/pixellogo-blog/raster-vs-vector-graphics, n.d.

[8] Ilya Baran, Jaakko Lehtinen, and Jovan Popovic. Sketching clothoid splines using shortest paths. *Comput. Graph. Forum*, 29:655–664, 2010.

[9] P. J. Barendrecht, M. Luinstra, J. Hogervorst, and J. Kosinka. Locally refinable gradient meshes supporting branching and sharp colour transitions. *The Visual Computer*, pages 949–960, 2018.

[10] M. Bessmeltsev and J. Solomon. Vectorization of line drawings via polyvector fields. *CoRR*, abs/1801.01922, 2018.

[11] T. Birdal and E. Bala. A novel method for vectorization. *CoRR*, abs/1403.0728, 2014.

[12] Computerphile. Resizing images. `https://www.youtube.com/watch?v=AqscP7rc8{\\_}M`, 2016.

[13] P. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, September 2004.

[14] Juan Carlos Gonzales. Use of convolutional neural networks for image classification. `https://www.apsl.net/blog/2017/11/20/use-convolutional-neural-network-image-classification/`, 2017.

[15] S. Hoshyari, E. A. Dominici, A. Sheffer, N. Carr, Z. Wang, D. Ceylan, and I. Shen. Perception-driven semi-structured boundary vectorization. *ACM Transactions on Graphics*, 37(4):1–14, 2018.

[16] J. Jimenez and J. Navalon. Some experiments in image vectorization. *IBM Journal of Research & Development, volume = 26 no = 6 month =*.

[17] J. Kopf and D. Lischinski. Depixelizing pixel art. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)*, 30(4):99:1 – 99:8, 2011.

[18] P. Laube, M. Franz, and G. Umlauf. Deep learning parametrization for b-spline curve approximation. *CoRR*, abs/1807.08304, July 2018.

[19] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[20] Mathematics of Computer Graphics and Virtual Environments. B-splines, 2015.

[21] P. Parsania and P. Virparia. *International Journal of Innovative Research in Computer and Communication Engineering*, (12):7409.

[22] E. Simo-Serra, S. Iizuka, K. Sasaki, and H. Ishikawa. Learning to simplify: Fully convolutional networks for rough sketch cleanup. *ACM Trans. Graph.*, 35(4):121:1–121:11, July 2016.

[23] J. Sun, L. Liang, F. Wen, and H. Shum. Image vectorization using optimized gradient meshes. Association for Computing Machinery, Inc., July 2007.

[24] Techquickie. How do vector graphics work? `https://www.youtube.com/watch?v=W2xknX3k6FY`, 2016.

[25] Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. Schroers. A fully progressive approach to single-image super-resolution. June 2018.

[26] G. Xie, X. Sun, X. Tong, and D. Nowrouzezahrai. Hierarchical diffusion curves for accurate automatic image vectorization. *ACM Trans. Graph.*, 33(6):230:1–230:11, 2014.

[27] X. Xiong, J. Feng, and B. Zhou. Real-time image vectorization on gpu. *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2016)*, 1:143–150, 2016.

[28] A. Yang. A.i. gigapixel an inside story topaz labs. `https://topazlabs.com/a-i-gigapixel-story/`, 2018.

[29] C. Yang, C. Ma, and M. Yang. Single-image super-resolution: A benchmark. pages 372–386. Springer International Publishing, 2014.

[30] M. Yang, H. Chao, C. Zhang, J. Guo, L. Yuan, and J. Sun. Effective clipart image vectorization through direct optimization of bezigons. *IEEE Transactions on Visualization and Computer Graphic*, Feb. 2016.