**INTRODUCTION**

**Project Orientation**

This project aims to figuring out the algorithms that are used to transform a bitmap image into vector image, and trying to optimize the methods.

*What makes our project different from the other existing softwares?*

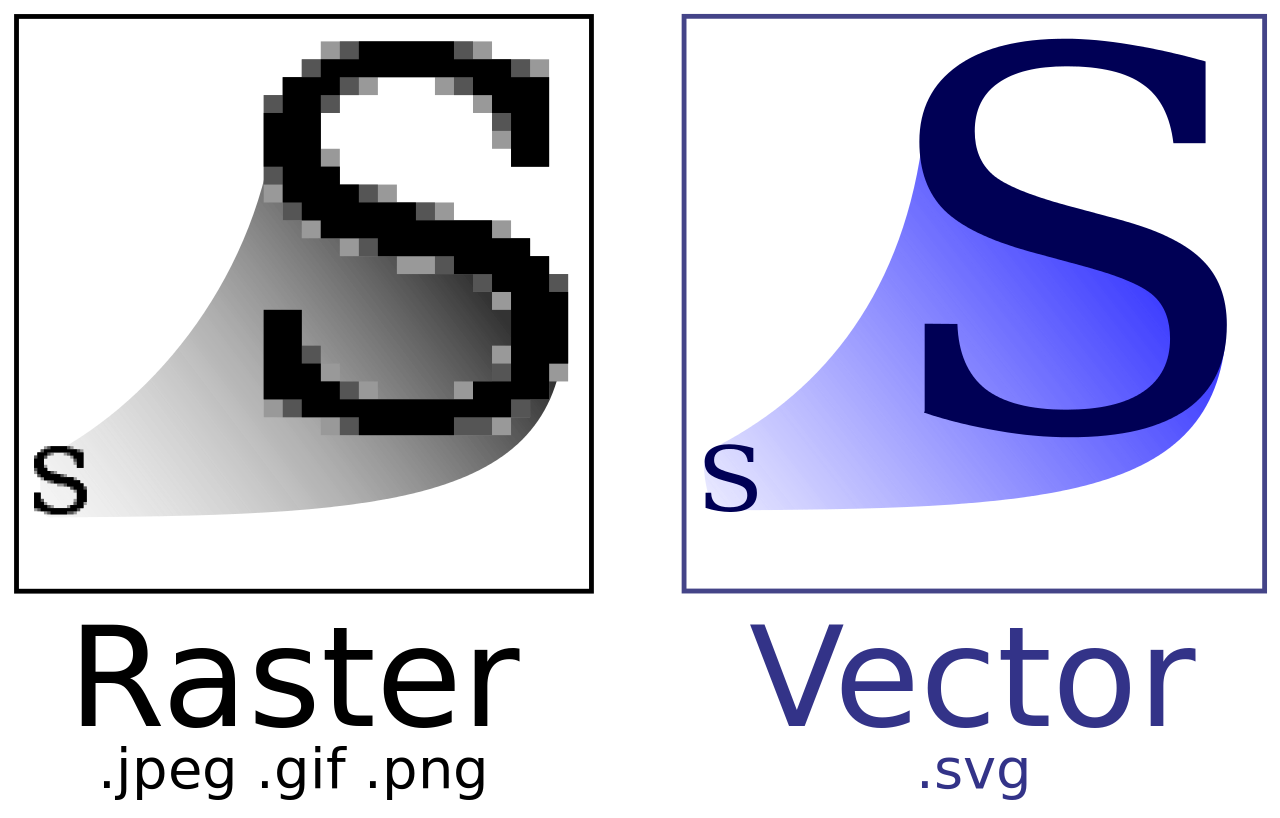
**Bitmap (Raster) Images**

Bitmap images are static. They are comprised of hundreds, and more often thousands, of tiny coloured dots called pixels. Each pixel has a fixed position within the makeup of the bitmap, and when these pixels are displayed together on a computer screen, they form an image. Bitmap images are, with a few exceptions, the standard for internet graphics, as well as Windows icons and backgrounds. Bitmaps are also the standard format for images captured by scanners and digital cameras. But, for some kind of purposes, people want to have a kind of images which dynamic, in the sense that we can enlarge the image without destruct it.

**Vector Images**

Vector image is made up of various “objects” such as lines and shapes. These objects are defined mathematically by a set of algorithms or formulas, which allow them to be redrawn over and over. Whereas the bitmap is static, the vector image is dynamic. It is not restricted by size or shape. It can easily be shrunk or enlarged, widened or compressed.

The comparation of bitmap and vector graphic images can be observed in this figure.



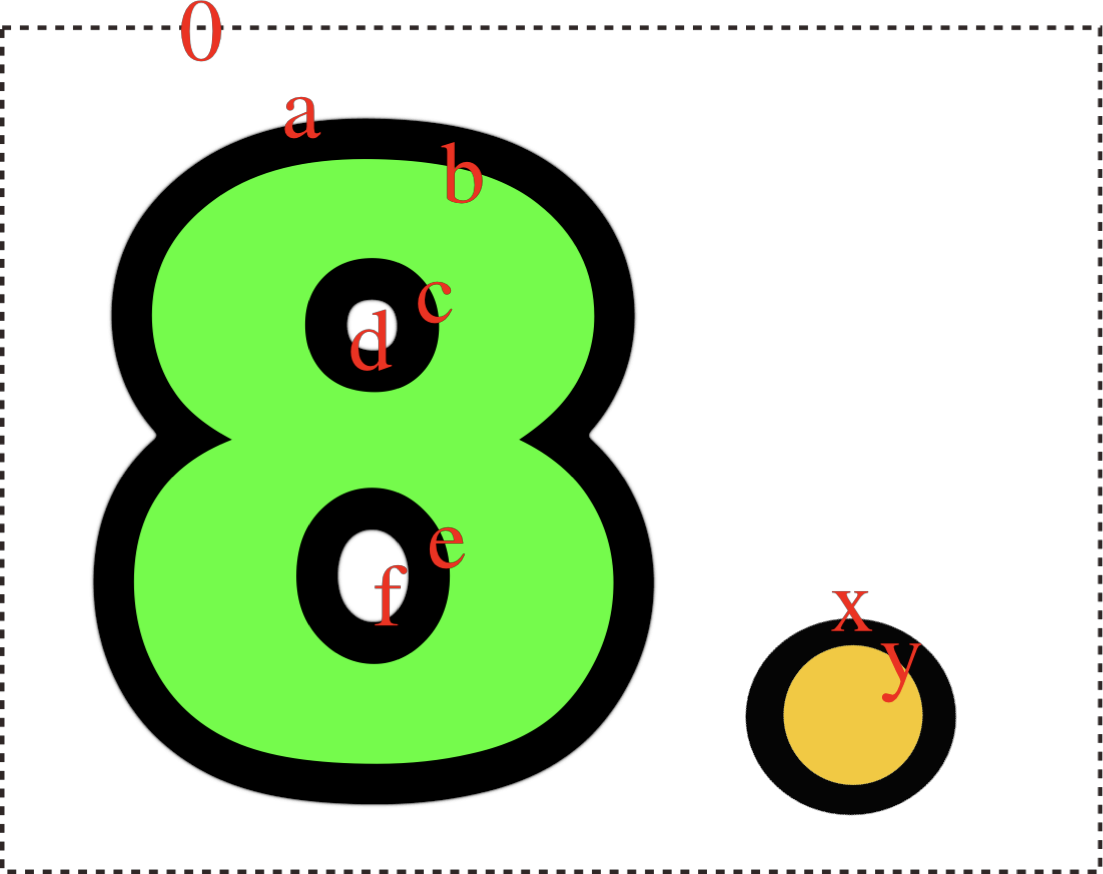
**General Framework**

The software takes a bitmap image as an input, and will result a vector graphic image (in SVG format). In general, the algorithms used in the image convertion coverts “contour detection”, “color transition”, “polygonalization”, and “corner detection”. For further details, kindly check below.

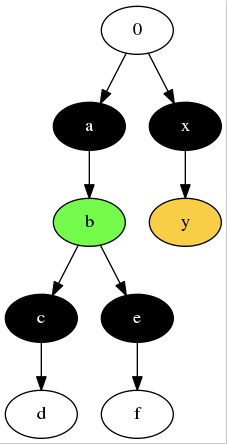
**WORK PACKAGES**

1. **CONTOUR DETECTION**

Contour detection is a process of identifying and locating sharp discontinuities in an image. This process takes a bitmap image as an input and results a list of bitmap contours. In some cases, we work with an image containing some shapes which are inside the other shapes. Therefore, before going through the contour detection, the first thing we need to do is extracting the contour hierarchy. This hierarchy represents the relationship between contours in the image. For more details, let’s observe this example.

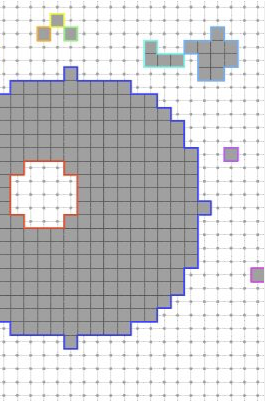


We have contours **0** (which is the image contour), the **a** contour (contour of the outer edge of the number 8), the **x** contour (the outer edge of the circle), the **y** contour which is the end of the color region started by the **x** contour (**x** and **a** are the end of the color regions started by the **0** contour), etc.. From this image you can see that the contours and color regions have a hierarchical structure:



This structure is the first thing that needs to be extracted. Basically what we need are the edges of this graph, every edge represents a contour, for example the edge 0->a represents the outer contour of the number 8. The edge 0->x represents the outer contour of the yellow dot on the right of the eight.

The contours can be represented as arrays of indices (i,j) of the pixel coordinates of the contour, for example, for a simple image:



Now, the image is not in binary pixel values, but it has smooth transitions. Here, contour detection methods will be used to find these contours.

1. **COLOR TRANSITION**

This part of the project is about representing the color transitions in a mathematical way. The input for this step is the color region of the bitmap picture, and then it will be represented as some kind of color transition in the SVG format. The colors produced by a gradient vary continuously with position, producing smooth color transitions. For example, things that can be represented in the SVG format are linear and radial gradients.

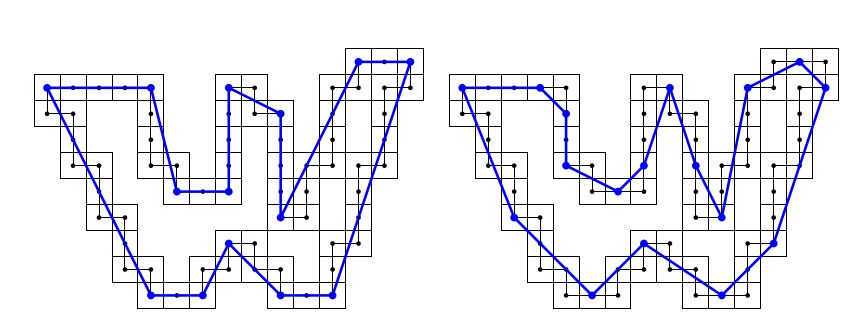
A linear color gradient is specified by two points, and a color at each point. The colors along the line through those points are calculated using [linear interpolation](https://en.wikipedia.org/wiki/Linear_interpolation), then extended perpendicular to that line. Meanwhile, a radial gradient is specified as a [circle](https://en.wikipedia.org/wiki/Circle) that has one color at the edge and another at the center. Colors are calculated by linear interpolation based on distance from the center. This can be used to approximate the [diffuse reflection](https://en.wikipedia.org/wiki/Diffuse_reflection) of light from a point source by a [sphere](https://en.wikipedia.org/wiki/Sphere). The example of linear and radial gradient can be observed in the following pictures.

|  |  |
| --- | --- |
|  |  |
| Linear Gradient | Radial Gradient |

Here we don’t have to restrict ourselves to color transitions available in the SVG format, we can just find a mathematical description of the color transitions, which will allow us to scale the image to a higher resolution.

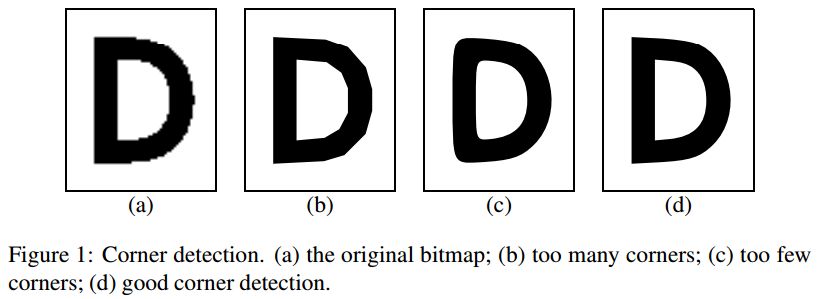
1. **POLYGONALIZATION**

The input for this step is the pixel outlines, which will be generated to some polygons, that simplify the outline, because the outline might contain too much redundant information, for example many positions in a straight line, the polygon would kind of reduce this information to its essence. This polygon calculation is optimizing some kind of error function from the polygon to the pixel outline. For example, these polygons are ones of the possible results for the same pixel contour.



1. **CURVE FITTING**

After finishing polygonalization step, the next step is to fit the curves. Beforehand, we need to do “Corner Detection” to detect the corners and every possible continuous curve. Let’s observe the following pictures.



(a) the original bitmap; (b) too many corners; (c) too few corners; (d) good corner detection

The second thing we need to do is defined a criteria of the “goodness” of the fit, so that we can obtain the optimal result. This will be some kind of distance function from the curve to the part of the polygon that it fits.