Pixelating Vector Line Art

Tiffany C. Inglis (Advisor: Craig S. Kaplan)
University of Waterloo

Abstract

Pixel artists rasterize vector paths by hand to minimize artifacts at low resolutions and emphasize the aesthetics of visible pixels. We describe Superpixelator, an algorithm that automates this process by rasterizing vector line art in a low-resolution pixel art style. Our method successfully eliminates most rasterization artifacts in order to draw smoother curves, and an optimization-based approach is used to preserve various path properties. A user study compares our algorithm to commercial software and human subjects for its ability to rasterize effectively at low resolutions. A professional pixel artist reports our results as on par with hand-drawn pixel art.

1 Problem and Motivation

Digital images are represented in one of two ways: as raster graphics or as vector graphics. Raster graphics use rectangular grids of coloured pixels to represent images, whereas vector graphics consist of paths defined mathematically by their control points. Technically speaking, all digital images must be eventually converted to raster graphics in order to be displayable as pixels on screen. The process of converting from vector graphics to raster graphics is called *rasterization*.

In a raster image, the pixels are of a fixed size and shape. Any geometric transformation (other than rotations by multiples of 90° or reflections in horizontal and vertical axes) applied to the image will produce an approximate result that is of lower quality than the original. Figure 1a shows a raster ellipse that becomes disconnected after rotation. If the same ellipse is rotated as a vector object *before* rasterization is applied (see Figure 1b), then the quality does not suffer. Many artists choose to work in vector form because they can delay the rasterization step to avoids unnecessary image deterioration, and the editing process is more flexible because they are working with higher-level path primitives rather than low-level pixels. Moreover, rasterizing vector graphics to different resolutions produces much better-looking results than scaling raster graphics to different sizes.

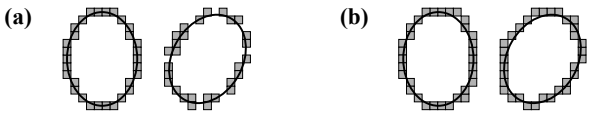


Figure 1: *(a) Rotating an ellipse in raster form. (b) Rotating an ellipse in vector form (before rasterization).*

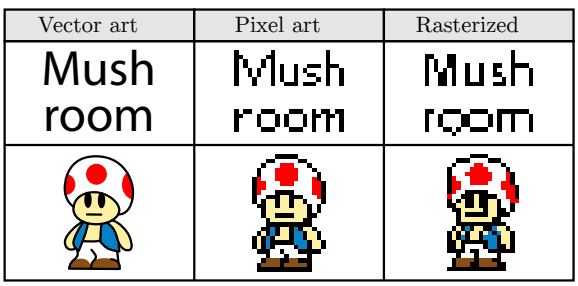


Figure 2: *Pixel art conveys information more effectively at low resolutions than naïvely rasterized vector art.*

Rasterization is a widely studied problem that forms part of the foundation of computer graphics. Numerous algorithms have been developed in the past few decades, with an emphasis of improving efficiency. All these rasterization algorithms assume pixels to be small enough to produce a continuous signal that can be processed seamlessly by the human visual system. However, the assumption that pixels are infinitesimal is not true and often we encounter limitations on the pixel level. In typography, for example, fonts are frequently rendered as pixel-thin paths. Naïvely applying rasterization to font outlines will introduce various artifacts, drastically reducing readability (compare the text renderings in Figure 2).

Pixel art is another area in which pixel placement is extremely important and current rasterization algorithms cannot replace human effort adequately (see the image comparison in Figure 2). Pixel art is a style of digital art created and edited on the pixel level. Originally, pixel art was designed to satisfy constraints of old graphics hardware and contained very coarse abstract features. Nowadays, it is popular not only for its nostalgic appeal but also for the clean look it provides for a given resolution. Pixel artists are extremely attentive to details because even small pixel-level changes can significantly affect the aesthetic quality of an artwork. They never use automatic tools beyond flood fills so as not to blur the image or introduce unwanted artifacts. As a result, hand-drawn pixel art looks cleaner and more detailed than automatically rasterized vector art.

Pixel art is supported by a large online community where individuals regularly share their work, critique the work of others, and create tutorials. A popular tutorial by Yu [2013] (summarized in Figure 3) covers the outlining, colouring, and shading steps involved in developing a finished pixel artwork. The outline of a pixel art image is referred to as the *line art*. We are particularly interested in creating clean line art from vector paths. By studying the work of pixel artists, and interacting with them directly, we can articulate the conventions they follow and ideally devise algorithms that embody those conventions.

Our research focuses on *pixelation* [Inglis and Kaplan 2012], a special class of rasterization algorithms designed to respect the aesthetic conventions of pixel art at low resolutions. Every pixel counts in this context, and a pixelation algorithm must therefore consider the placement of every pixel carefully, taking into account its effect on neighbouring pixels. Ultimately, pixelation should mimic the pixels that would be chosen by a human artist.

In this paper we present *Superpixelator*, a pixelation algorithm for converting vector line art to pixel line art. Our algorithm differs from traditional rasterization in that it uses an optimization-based approach to balance various competing aesthetic goals, producing results comparable to hand-drawn pixel art. To show that our algorithm is indeed a significant improvement over other rasterization algorithms and that it successfully meets pixel art standards, we conducted an extensive evaluation to collect both quantitative and qualitative data (see Section 4). The test cases include actual images and geometric primitives, in order to cover a wide range of possible vector inputs. The participants ranged from amateurs to expert pixel artists. A statistical analysis validates our claim that Superpixelator's results are more visually appealing compared to those of other rasterizers. The feedback from a professional pixel artist confirms that our algorithm performs on par with pixel artists.

Figure 3: *Selected steps from Yu's pixel art tutorial [2013], showing the evolution of a design from initial pixel line art to a finished work.*

2 Background and Related Work

As rasterization is a fundamental problem in computer graphics, much research has been done in this area. Bresenham's algorithm is commonly used to rasterize lines and circles [1965; 1977], and has been extended to handle ellipses and spline curves [Van Aken 1984; Anantakrishnan and Piegl 1992]. Recent advances in rasterization focus more on improving efficiency by simplifying calculations [Boyer and Bourdin 1999] or exploiting graphics hardware [Liu et al. 2011].

Font rasterization is a related field that specializes in converting text from vector form to raster form. At low resolutions, maintaining text readability is a difficult problem. Font hinting, used in TrueType fonts, helps rasterizers decide where to render pixels for troublesome areas [Stamm 1998; Hersch and Bétrisey 1991; Zongker et al. 2000]. Subpixel rendering algorithms [Elliot 1999], such as Microsoft's ClearType, take advantage of the colour subpixel layout in a liquid crystal display to increase the effective resolution available for antialiasing.

More recently, with the resurgence of retro pixel art games, the computer graphics community has taken an interest in pixel art. Kopf and Lischinki [2011] developed a depixelization algorithm that converts low-resolution pixel art into vector art by analyzing pixel-level structures. Gerstner et al. [2012] presented a method for abstracting high-resolution raster images such as photographs into lower-resolution pixel art outputs with limited colour palettes.

2.1 Artifacts in pixel line art

Unlike computer scientists, pixel artists take a different approach to the pixelation problem. Instead of developing algorithms, artists often define a set of aesthetically pleasing features to strive for and undesirable artifacts to avoid, then try to work within these constraints. We believe it is important to bridge the gap by thoroughly studying guidelines developed by artists in order to create an algorithm that generates artistic output.

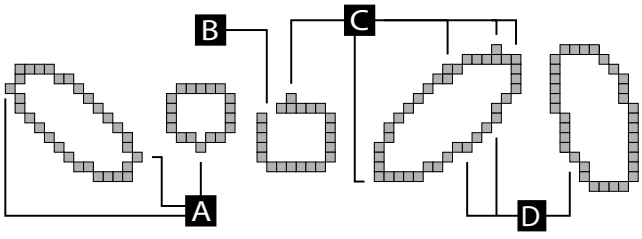


Figure 4: *Artifacts in pixel line art: blips (A), missing pixels (B), extra pixels (C), jaggies (D), and asymmetry (unlabelled).*

According to Yu's tutorial [2013], pixel line art should be drawn at one-pixel thickness while trying to avoid artifacts. Figure 4 shows several poorly rasterized ellipses exhibiting five types of artifacts: blips, missing pixels, extra pixels, jaggies, and asymmetry. We will provide concrete definitions for blips, jaggies, and asymmetry. As for missing pixels and extra pixels, they are simply violations of the one-pixel-thick requirement.

Blips are single pixels that stick out of smooth paths. A blip occurs when a vector path lightly grazes a column or row of pixels in the underlying pixel grid, causing the pixelated path to contain a single pixel in that column or row. Figure 5 (left) shows a spiral path rasterized with four blips.

Jaggies is a term used by the pixel art community to describe places where a pixelated path looks jagged. They occur when the pixels are not arranged in a way that accurately reflects the curvature of the vector path. Consider the vector path in Figure 6a with positive curvature. Ideally, its pixelation should also have this property, but first we must adapt the notion of curvature to pixelated paths.

A pixelated path is a series of *pixel spans* (i.e., contiguous rows or columns of pixels) joined together at their diagonal corners. It is essentially a polygonal approximation of the original vector path formed by joining the corners of these pixel spans, as shown in Figures 6b and 6c. Each polygonal approximation can be written as a sequence of the line segment slopes. For example, the two pixelated paths in Figure 6b and c are given respectively by $\{1/4, 1/3, 1/2, 1, 1, 1, 2, 3\}$ and $\{1/4, 1/2, 1, 1/2, 1, 1, 2, 3\}$. We call these *slope sequences* since they describe how the slope changes in a pixelated path. The *curvature* of a pixelated path is then defined as the rate of change of its slope sequence.

In our example, since the vector path has positive curvature, we want its corresponding pixelation to have positive curvature. If it does not, then the point at which the curvature changes from positive to negative is referred to as a *jaggie*. The first pixelated path has positive curvature everywhere because its slope sequence is increasing, and therefore it has no jaggies. The second pixelated path, on the hand, contains one jaggie.

Symmetry refers to rotational and reflection symmetry. All ellipses, for example, have 180° rotational symmetry; if they are axis-aligned, then they also have both horizontal and vertical reflection symmetry. When rasterizing a symmetric vector path, it is crucial to preserve symmetry in the resulting pixelated path. Figure 4 shows examples of four asymmetrically rasterized ellipses.

In the next section, we will describe how our algorithm attempts to eliminate these artifacts in order to produce more visually appealing pixelations. Note that in some cases it may be impossible to remove all artifacts. Instead, we optimize over a set of candidates to find the pixelation that achieves the most satisfactory trade-off between competing aesthetic goals.

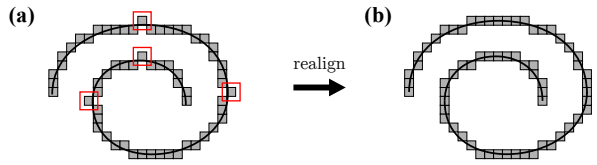


Figure 5: *Realign a vector path by shifting its local extrema to pixel centres to remove blips.*

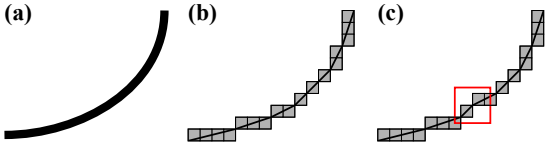


Figure 6: *(a, b) A positive curvature path should be represented by pixel spans that form an increasing sequence in terms of slope. (c) Jaggies are places where the sequence is decreasing.*

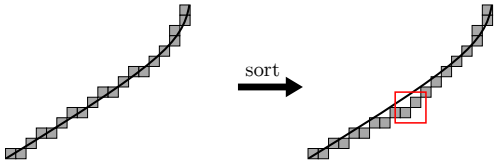


Figure 7: *Sorting a pixelated path may result in a significant deviation from the vector path it is trying to approximate.*

3 Approach and Uniqueness

Our algorithm, Superpixelator, converts vector line art into pixel line art. It builds upon Bresenham's algorithm by adding more steps to suppress various types of pixel art artifacts. We choose Bresenham's algorithm because it efficiently rasterizes paths to one-pixel thickness, so that we only have to worry about how to eliminate the remaining artifacts, namely blips, jaggies, and asymmetry.

3.1 Removing blips via path realignment

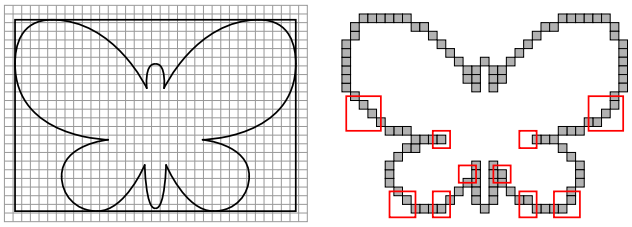
Blips only occur at the local extrema (i.e., the local minima and maxima in either the x- or y-direction) of a path, where it is possible for the vector path to lightly graze a row or column of pixels. Figure 5a shows a rasterized spiral path with blips at four of its local extrema. To remove blips, we must realign the vector path with respect to the pixel grid. Take each local extremum, shift it to the nearest pixel centre, and adjust the rest of the path accordingly. The realigned path and its corresponding pixelation, as shown in Figure 5b, no longer contains any blip artifacts.

3.2 Removing jaggies via partial sorting

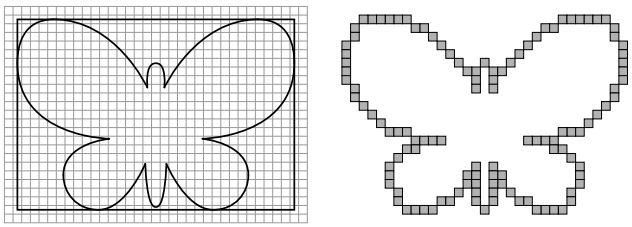
Jaggies are artifacts that occur when the curvature of a vector path is not accurately reflected in its pixelated counterpart, causing a jagged appearance. One way to remove them is to rearrange the pixels to correspond to a sorted slope sequence. For example, the slope sequence for the pixelated path in Figure 6c is $\{1/4, 1/2, 1, 1/2, 1, 1, 2, 3\}$; after sorting, the new slope sequence is $\{1/4, 1/2, 1/2, 1, 1, 1, 2, 3\}$, which corresponds to the jaggie-free pixelated path in Figure 6b.

The sorting method is guaranteed to remove all the jaggies, which often results in a smoother pixelated path. However, it is possible for the sorted pixelated path—particularly if it is long—to deviate sufficiently from the vector path that it no longer provides a good approximation. Figure 7 shows an example of a pixelated path that deviates significantly from the vector path after sorting, to the point where some pixels no longer intersect the vector path.

Our solution is to apply *partial sorting*, which takes into account both jaggedness and deviation. Jaggedness is the number of jaggies in a pixelated path and deviation measures the minimum distance between a vector path and its pixelation. For each pixelation, its cost is defined as a weighted sum of these two quantities, with the weights determined empirically in an attempt to maximize overall visual appeal. We then search among a small set of candidates to find the optimal pixelation that accurately approximates the vector path using minimal jaggies.



(a) *Before bounding box shifting.*



(b) *After bounding box shifting.*

Figure 8: *Bounding box shifting removes the asymmetry caused by path realignment. The red boxes indicate asymmety artifacts.*

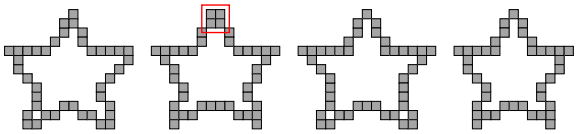


Figure 9: *Four non-ideal pixelations of a 14-pixel-wide star. From left to right, they are asymmetric, dull angle (due to the 2×2 pixel block), too wide, and too narrow.*

3.3 Preserving symmetry and other path properties via bounding box adjusting

We use path realignment to remove blips, which involves moving local extrema to the nearest pixel centres. Depending on the overall alignment of the path, the points may move asymmetrically, causing symmetric paths to be pixelated asymmetrically.

Figure 8a shows an example of a symmetric path pixelated asymmetrically. In Figure 8b, the same path is adjusted slightly so that the resulting pixelation is symmetric. The difference between these two paths is their overall alignment, which we can think of in terms of their bounding boxes. The way we preserve symmetry is as follows: first adjust the path by its bounding box so that the four corners of the box lie on pixel centres; then ensure that all steps leading up to the actual pixelation are performed symmetrically about the centre of the bounding box.

(a) *Computer-generated pixel art.* **(b)** *Samples of hand-drawn pixel art.*

Figure 10: *Results from Part 1 of the evaluation, which focuses on drawing low-resolution images.*

Besides symmetry, there are other path properties to consider preserving as well. Figure 9 shows four different pixelations of a 14-pixel-wide star. The first pixelation from the left is asymmetric. The second one is symmetric but has a 2×2 pixel block that causes the top acute angle to look dull. The third and fourth pixelations are both symmetric and preserve sharp angles, but unlike the first two, the third is too wide (15 pixels wide) while the fourth is too narrow (13 pixels wide).

We consider the following set of properties to be important when pixelating a vector path and try to preserve them: symmetry, width, height, position, angle sharpness, and aspect ratio. We define a cost based on these properties and use optimization to find the bounding box that minimizes the cost. We restrict our search to a small set of bounding boxes that differ from the initial bounding box by at most one pixel on all four sides. As a result, symmetry is usually preserved, save for exceptional cases in which the benefits of preserving other properties overweigh that of retaining symmetry.

3.4 Algorithm summary

In summary, our algorithm pixelates a vector path as follows:

1. Adjust its bounding box to preserve global path properties.
2. Realign the path by its local extrema to remove blips.
3. Rasterize the path with Bresenham's algorithm.
4. Apply partial sorting to remove jaggies while controlling the amount of deviation from the vector path.

Even though our algorithm uses optimization in two places, we restrict the search space to a small set of possible candidates and simplify the cost calculations as much as possible to keep the algorithm fast. We implemented Superpixelator as a Java application that rasterizes at about a tenth the speed of the native Java rasterizer, still fast enough to provide real-time feedback, particularly at typical pixel art resolutions. Our system allows the user to draw and edit a collection of vector paths, while seeing the pixelated paths update simultaneously.

4 Results and Contributions

For evaluation, we compare Superpixelator to rasterization algorithms used in commercial software, as well as drawings by human subjects (including both amateurs and expert pixel artists). The test cases contain both low-resolution images and geometric primitives in order to cover a wide range of possible inputs. The evaluation is split into two parts.

4.1 Evaluation of low-resolution images

The first part of our evaluation is a user study comparing our results to other computer-generated pixel art (rasterized by commercial software) and hand-drawn pixel art. Figure 10a shows a collection of vector input images along with rasterizations by Superpixelator, Adobe Illustrator, and Adobe Photoshop. The input images are designed to cover a variety of shapes, including lines of different slopes, curves, sharp angles, and features of various sizes.

To obtain hand-drawn pixel art, we conducted an online survey advertised through various social media including pixel art forums. Nearly 200 people participated, drawing pixel art based on eight vector images. For convenience and consistency, we developed a simple web application with which the participants drew their artwork and answered demographic questions such as their age, gender, artistic training, and pixel art experience. Figure 10b shows samples of pixel art created by the human subjects. The amount of variation between these images was much greater than anticipated, ranging from blind tracings to creative interpretations that barely resembled the original vector art.

We categorized the computer-generated images into three groups (Superpixelator, Illustrator, and Photoshop) and the hand-drawn images into four groups (based on artistic training and pixel art experience, which appeared to be the factors most highly correlated with the quality of pixel art produced), and combined them to create a pairwise comparison test. The pairwise comparison test consisted of 200 pairs of pixel art images. Each pair was drawn randomly from two different groups. The participant was then asked to choose either the more visually appealing image or the one more similar to the vector input.

To analyze the results, we took all the pairwise comparisons between each pair of groups, and performed a hypothesis test to determine if the ratings showed a significant preference of one group over another. The results gave us a ranking of the seven groups based on visual appeal and similarity to the vector inputs.

For both visual appeal and similarity, Superpixelator was ranked the highest, followed by the human subjects, Illustrator, and finally Photoshop. Among the human subjects, we were surprised to find that the group with the most artistic training and pixel art experience was not ranked the highest. After taking a closer look at their pixel art drawings, we discovered that the participants in this group tend to embellish their artwork (see Figure 10b for examples), making them poor representations of the input vector images.

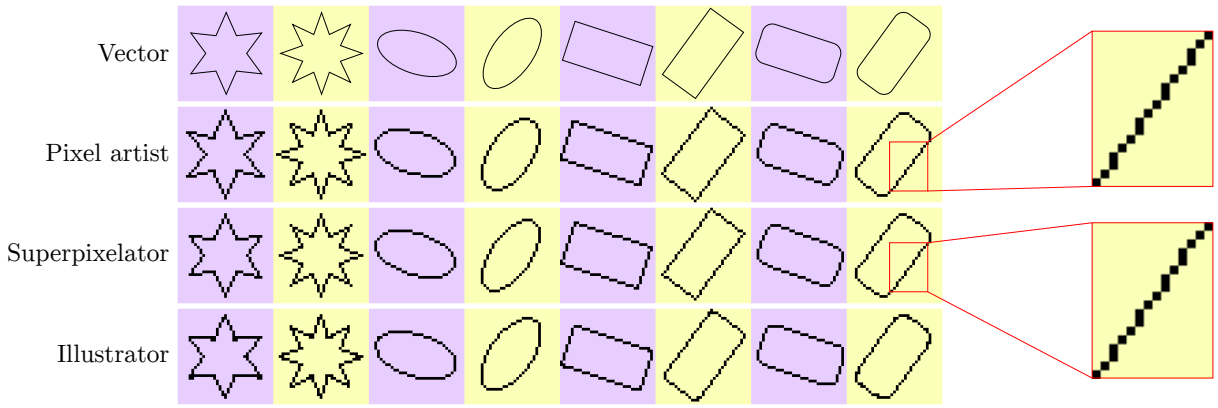


Figure 11: *Results from Part 2 of the evaluation, which focuses on drawing geometric primitives.*

Even though Superpixelator outperformed the human groups on average, there are individual participants (usually pixel artists) whose drawings received higher ratings. We analyzed the most highly rated images and learned that their success rests on making certain alterations based on contextual information. For example, the dinosaur's teeth are depicted with triangles to show their sharpness, and strands of hair are carefully separated to avoid pixel clusters.

4.2 Evaluation of geometric primitives

For the second part of the evaluation, we tested Superpixelator's ability to draw geometric primitives, including stars, ellipses, rectangles, and rounded rectangles. Being able to draw these shapes well is important because many image editors provide tools for drawing them, and these shapes are often used as building blocks for more complex drawings.

We worked closely with pixel artist Sven Ruthner, who provided both hand-drawn pixel art and qualitative feedback on our results. Ruthner is well-known in the pixel community, works as an administrator for the Pixelation forum, and has been doing pixel art professionally for around a decade. Our test consisted of a total of 24 shapes, eight of which are shown in Figure 11. These shapes were rasterized by Superpixelator and Illustrator (we omitted Photoshop because its results were identical to those of Illustrator), and converted manually to pixel art by Ruthner. Both automatic rasterizations took less than one second, compared to the artist who spent about one hour on the drawings.

The results are shown in Figure 11. Illustrator introduced many extra pixels and did not preserve any symmetry even though all the vector shapes have some form of symmetry. Ruthner, on the other hand, made sure all the shapes are drawn symmetrically. Superpixelator's results are very similar to the pixel artist's, with the most noticeable difference being the 6-pointed star; Ruthner drew it with horizontal symmetry, whereas Superpixelator sacrificed symmetry to maintain similarity to the vector shape.

According to Ruthner, Superpixelator's results are as good as the work of any pixel artist. He believes that our algorithm takes a correct approach to the problem and captures some of the shapes more faithfully than he did. One issue Ruthner noticed is the way we draw straight lines. When pixel artists draw straight lines, they tend to follow a regular pixel pattern to create a more structured look. The magnified region in Figure 11 shows a comparison between pixelated lines. Notice that in the artist's version, the slope sequence consists of a repeating $\{1, 1, 2\}$ pattern, whereas Superpixelator does not follow this pattern strictly.

5 Conclusion and future work

Superpixelator is a pixelation algorithm that rasterizes vector paths in the pixel art style. Blips are eliminated through path realignment. Path properties are preserved by adjusting the bounding box to an optimal configuration. Paths are drawn by applying partial sorting, which considers both jaggedness and deviation. A professional pixel artist prefers Superpixelator to other rasterization algorithms, and believes our algorithm correctly mimics what pixel artists do.

For future work, we would like to pixelate vector line drawings more effectively by considering relationships between various paths. Currently, Superpixelator treats paths individually, and when merging the resulting pixels, new artifacts can emerge (as demonstrated in Figure 12). Finding a method to rearrange the paths to avoid inter-path conflicts while preserving the overall topology of the drawing is an interesting challenge.

In icon design and sprite creation, it is often necessary to display the same image at different resolutions. Figure 13 shows an icon image drawn manually with different levels of detail to accommodate for multiple resolutions. As resolution decreases, less salient features are removed and the remaining features are redrawn in a more abstract fashion. It would be immensely useful to have an algorithm that automatically resizes an image with the correct level of detail and abstraction.

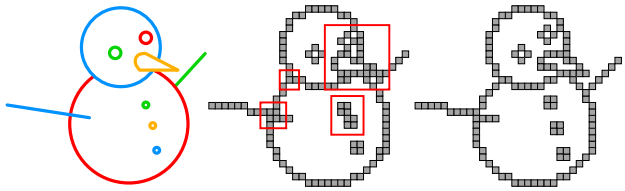


Figure 12: *(a) A vector image consisting of many paths. (b) Pixelating the paths individually creates inter-path conflicts. (c) By the rearranging the paths, these conflicts can be avoided.*

Figure 13: *An icon drawn with different levels of detail.*

References

ANANTAKRISHNAN, N., AND PIEGL, L. A. 1992. Integer de Casteljau algorithm for rasterizing NURBS curves. *Computer Graphics Forum 11*, 151–162.

BOYER, V., AND BOURDIN, J. 1999. Fastlines: a span by span method. *Computer Graphics Forum 18*, 3, 377–384.

BRESENHAM, J. E. 1965. Algorithm for computer control of a digital plotter. *ICM Systems Journal 4*, 25–30.

BRESENHAM, J. E. 1977. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM 20*, 100–106.

ELLIOT, C. H. B. 1999. Reducing pixel count without reducing image quality. *Information Display 15*, 12, 22–25.

GERSTNER, T., DECARLO, D., ALEXA, M., FINKELSTEIN, A., GINGOLD, Y., AND NEALEN, A. 2012. Pixelated image abstraction. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, NPAR '12, 29–36.

HERSCH, R. D., AND BÉTRISEY, C. 1991. Model-based matching and hinting of fonts. In *Proceedings of SIGGRAPH '91*, ACM Computer Graphics, vol. 25, 71–80.

INGLIS, T. C., AND KAPLAN, C. S. 2012. Pixelating vector line art. In *Proceedings of the 10th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '12, 21–28.

KOPF, J., AND LISCHINSKI, D. 2011. Depixelizing pixel art. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011) 30*, 4, 99:1 – 99:8.

LIU, Y. K., WANG, P. J., ZHAO, D. D., SPELIC, D., MONGUS, D., AND ZALIK, B. 2011. Pixel-level algorithm for drawing curves. In *Theory and Practice of Computer Graphics*, Eurographics Association, vol. 18, 33–40.

STAMM, B. 1998. Visual True Type: A graphical method for authoring font intelligence. In *Proceedings of the 7th International Conference on Electronic Publishing*, Springer-Verlag, London, UK, UK, EP '98/RIDT '98, 77–92.

VAN AKEN, J. R. 1984. Efficient ellipse-drawing algorithm. *IEEE Computer Graphics & Applications 4*, 9, 24–35.

YU, D., 2013. Pixel art tutorial. `makegames.tumblr.com/post/42648699708/pixel-art-tutorial`. [Online; accessed 14-Mar-2013].

ZONGKER, D. E., WADE, G., AND SALESIN, D. H. 2000. Example-based hinting of True Type fonts. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., SIGGRAPH '00, 411–416.