A VECTOR-BASED METHOD FOR PROCEDURAL AMBIGRAM GENERATION

Scott VanRavenswaay
scottvr@gmail.com

ABSTRACT

Rotational ambigrams - typographic designs that read as the same or a different word when rotated 180 degrees - represent a significant artistic and geometric challenge. Manual creation of ambigrams is a specialized skill requiring extensive trial and error. Previous automated approaches have often focused on finding candidate words from a dictionary or have required a semi-automated, human-in-the-middle process using vector graphics editors. [1] This paper presents a novel, fully procedural method for generating ambigrammatic glyphs from pairs of characters using a purely vector-based approach. Our method avoids the complexities and artifacts of raster-space techniques by leveraging a sequence of affine transformations and Boolean path operations. We first align two source glyphs, derived from any TrueType font, by their geometric centroids. A smaller, scaled version of the resulting merged path is then generated. Finally, we apply a Boolean XOR operation between the original merged path and its scaled-down counterpart to create a clean, single-stroke "outline" glyph. This technique is computationally efficient, resolution-independent, and provides a tunable parameter for controlling the final stroke weight, offering a robust foundation for automated ambigram generation tools for designers and artists.

***K*eywords** Procedural Generation · Ambigrams · Vector Graphics · Computational Typography · Boolean Operations · PathOps · Fonts.

1 Introduction

The creation of ambigrams is a design discipline that combines artistic creativity with geometric constraints. A rotationally symmetric ambigram must form a legible character when viewed both upright and upside-down. This dual-state requirement makes their manual design a non-trivial task. While previous work has explored specific solutions, a generalized, procedural approach remains a compelling challenge.

This paper details a method for automatically generating an ambigrammatic glyph that fuses two source characters (e.g., 'a' and 'b'). Our primary contribution is a purely vector-based pipeline that constructs a hollow "outline" style ambigram from existing general-purpose fonts readily available on the user's computer or downloadable from the Internet. This method avoids the common pitfalls of rasterization-based approaches, such as aliasing artifacts and computational expense, by operating directly on the glyphs' mathematical path data.

2 Methodology

Our approach transforms two source glyph paths, P_1 and P_2, into a single, rotationally symmetric ambigram path, $P_{ambigram}$. The process consists of three main stages: alignment, merging, and outlining.

2.1 Initial Glyph Preparation

Given a character pair (c_1, c_2), we first extract their corresponding vector paths from a given TrueType font. The path for the second character, P_2, is immediately rotated by 180° to create P_2'.

2.2 Centroid Alignment and Merging

To create a coherent composite shape, the two paths must be aligned. We employ a centroid alignment technique. The geometric centroid $C = (c_x, c_y)$ of each path's bounding box is calculated. Each path is then translated by $-C$ to align its center with the origin $(0, 0)$.

These two individually centered paths are then merged into a single base shape, P_{base}, using a Boolean $union$ operation.

Listing 1: pseudocode for alignment and merge

```
path1_aligned = translate(path1, -centroid1)
path2_aligned = translate(path2_rotated, -centroid2)
P_base = union(path1_aligned, path2_aligned)
```

2.3 Outline Generation via XOR Operation

The final stylistic effect is achieved by creating an outline of P_{base}. This is accomplished by subtracting a smaller, scaled version of the path from itself.

First, a scaled-down path, P_{scaled}, is created by applying an affine transformation that scales P_{base} toward its own geometric center. The `scale_factor` (e.g., 0.88) is a key tunable parameter that directly controls the thickness of the final outline.

Finally, a Boolean XOR operation is performed between the base path and the scaled path. The XOR operation ($P_{base} \oplus P_{scaled}$) yields the regions present in one path or the other, but not both. This elegantly produces both the outer boundary and the correct inner boundaries for any holes in the glyph, resulting in a clean outline $P_{ambigram}$.

Listing 2: Pseudocode for scaling transform

```
bounds = P_base.bounds
center = get_center(bounds)
scale_factor = 0.88

# Transform moves path to origin, scales it, then moves it back

transform = T(center) * S(scale_factor) * T(-center)
P_scaled = apply_transform(P_base, transform)

P_ambigram = P_base ^ P_scaled
```

This method proved more robust than a simple `difference` operation, which failed to correctly render the outlines of interior holes in some cases.

3 Results

The described 'outline' strategy successfully generates single-stroke ambigrammatic glyphs for a wide variety of character pairs. The resulting paths are clean, fully vector, and can be saved as SVG files or composed into a final raster image. As the process avoids rasterization until the final rendering stage, it is fast and resolution-independent. Figure 1 shows some example outputs for the pair 'ab'.

Figure 1: 'ab' using (from left to right) "18cents", "Acme", "Alef Bold", "Alef Regular", "Alfredo" TrueType fonts.

Example ambigrams made with various fonts from the author's `Windows\Fonts` directory:

Figure 2: The string "deadbeef" as ambigram generated from Arial.ttf

Figure 3: The hexadecimal number $d34db33f$ using Borea.ttf

4 Discussion and Future Work

The primary strength of this method is its simplicity and robustness in that it can work with any TrueType Font and does not need any prior knowledge of the font it is to work with. Our vector-based method stands in contrast to recent work in the field that has successfully applied deep learning techniques, such as diffusion models, to generate ambigrams [2] [3]. While powerful, those methods require significant computational resources and training data, whereas our approach is a lightweight, geometric algorithm. By relying on a small set of vector operations, it avoids the complex failure modes encountered during our experiments with raster-space skeletonization and dilation, which included low-level library crashes and unrecoverable visual artifacts.

The main limitation is that the aesthetic quality is highly dependent on the geometric relationship between the two source characters. Not all pairs produce a legible result, necessitating a "human-in-the-loop" to judge the output, but considering the use cases are all for human consumption (visual arts, tattoos, etc), this is really just a matter of taste.

This work opens several avenues for future research:

- **Automated Legibility Scoring:** Implementing a **Hausdorff distance** or similar calculation to compare the generated glyph against the canonical source characters could provide a quantitative "legibility score." This would allow the system to automatically select the best-performing strategy or parameters for any given pair.
- **Advanced Alignment Strategies:** Moving beyond simple centroid alignment to an iterative **shape registration** approach could yield more natural and clever alignments by searching for an optimal fit that maximizes overlap or minimizes a distance metric.
- **Exploring Other Styles:** A "half-letters" strategy we designed, where the top halves of characters are clipped and merged, represents a promising direction for creating stylistically different ambigrams and warrants further development.

5 Conclusion

We have presented a robust and efficient method for the procedural generation of outline-style rotational ambigrams. By operating exclusively in vector space with boolean path operations, our technique provides a reliable foundation for tools aimed at assisting artists and designers in the complex task of ambigram creation. The potential for integrating automated quality scoring and more advanced alignment algorithms suggests a rich future for this area of computational typography.

Code and Examples

The source code is for fambigen is available at `https://github.com/scottvr/fambigen`

A Typeface gallery demonstrating a non-cherry-picked pass over all 406 TrueType Fonts in the `c:\windows\Fonts` directory on the author's personal computer can be seen at `https://killsignal.net/deadbeef`

References

[1] Jörn Loviscach. Finding approximate ambigrams and making them exact. In *Eurographics (Short Papers)*, pages 25–28, 2010.

[2] Boheng Zhao, Rana Hanocka, and Raymond A Yeh. Ambigen: Generating ambigrams from pre-trained diffusion model. *arXiv preprint arXiv:2312.02967*, 2023.

[3] Takahiro Shirakawa and Seiichi Uchida. Ambigram generation by a diffusion model. In *International Conference on Document Analysis and Recognition*, pages 314–330. Springer, 2023.