Finding Approximate Ambigrams and Making them Exact

J. Loviscach

Fachhochschule Bielefeld (University of Applied Sciences), Germany

Abstract

Rotational ambigrams are arrangements of letters that can also be read upside down. Existing approaches to automatically create such ambigrams employ highly artificially-looking and difficult-to-read typefaces. In contrast to that, the ambigram generator introduced here is based on a vector graphics editor that ensures perfect symmetry. A major component is an algorithm to smoothly fuse different vector shapes. As not all words lend themselves to be converted into legible ambigrams, an optional preparatory step is included. In this step, a dictionary is searched for words that are shaped almost symmetrically so that meaningful input is provided to the editing stage.

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Graphics Utilities—Graphics editors

1. Introduction

Ambigrams—a term already used by Hofstadter [Hof85, p. 276] in the Eighties—have gained much popularity in current popular culture through Dan Brown's novel "Angels and Demons" [Bro03] and its film adaptation. This paper addresses a major type of rotational ambigrams, namely lettering that forms a point-symmetric shape. For a tiny number of words such as "SOS" and "pod" this is easy to achieve with little or even no change to the shapes. Typically, however, tweaking the shapes of given word to turn it into an ambigram requires an artist's eye and much experimentation, see for instance the typographic works by John Langdon, who was one major source of inspiration for Dan Brown's novel.

Currently available software that helps to automatically create ambigrams is based on special typefaces. Their letters look like other—upright—letters when read upside down. The legibility, however, is poor in most cases. In addition, the uncommon forms of the letters direct the reader's attention to the circumstance that the text forms an ambigram. A good ambigram looks like normal text on first sight and reveals its symmetry only on closer inspection.

This work presents a two-pronged approach to create ambigrams with common typefaces:

1. A dictionary is used to find promising candidate words whose shapes are already close to symmetric. Even though this step is optional, replacing a word by a synonym that has more intrinsic symmetry benefits the result strongly. In addition, the best candidates for ambigrams in the rank lists produced by the search include many surprise findings and thus support a serendipitous approach.
2. An vector graphics editor with built-in enforced symmetry is used to tweak the shapes, see Figure 1. This editor possesses standard features but employs a freehand curve drawn by the user to cut the shapes. The remaining parts of the shapes are copied, rotated by 180 degrees and smoothly connected to the original ones.

To the author's best knowledge, this paper is the first to pick up the topic of ambigrams in computer graphics research. Further contributions to the state of the art are the application of a dictionary-based search to vector graphics shapes and the smooth fusion of vector graphics shapes.

This paper is structured as follows: Section 2 discusses related work. Section 3 introduces the method for finding words that are almost ambigrams. The ambigram editor—the main part of the prototype application—is described in Section 4. The details about the smooth fusing process are given in Section 5. Section 6 concludes this paper and points out options for future development.

2. Related work

Several approaches to create ambigrams have already been implemented. Probably the earliest one is Ambimatic [Hol09]. It employs a specific, highly stylized and poorly legible typeface that contains all combinations of

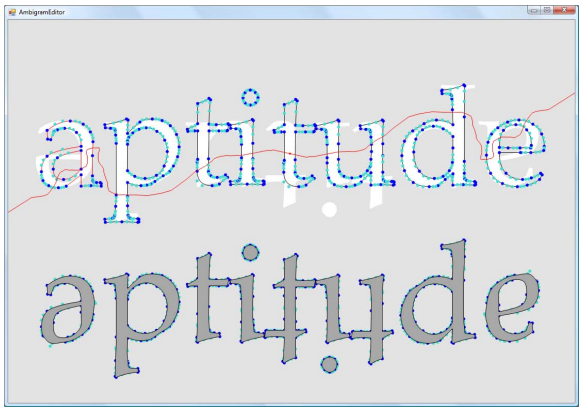

Figure 1: *The ambigram editor offers standard editing functions but enforces symmetry. It cuts the shapes along a path drawn by the user, mirrors them and connects the mirrored ones to the original ones with adjustably soft transitions. The input is displayed in the top half, the output in the lower half.*

one character (to be read in normal orientation) fused with another character (to be read upside down). Glyphusion [Hun10], a newer approach that is part of several other products, is based on a blackletter typeface in which a single character may turn into one or two characters when read upside down. To achieve this effect, the different characters are only distiguished by strokes that are tiny in comparison to the vertical blackletter base strokes. This reduces the legibility, which is also true for the optional "Script" typeface, which rather looks like bold slanted sans-serif.

Automatic symmetrization has been handled by introducing constraints into vector graphics editing software. For instance, Ryall et al. [RMS97] describe an editor for graph-type diagrams that can enforce—among others—symmetry constraints among the nodes. Igarashi et al. [IMKT97] propose to automatically detect—among others—approximate symmetries between stroke input and existing lines and then make these symmetries exact. As the application addressed in this work is concerned with the symmetrization of shapes *that already exist*, these approaches do not easily carry over.

There is much previous work in computational geometry on detecting approximate and partial symmetries. For instance, Mitra et al. [MGP06] build on the idea of the Hough transform by looking for clusters in the seven-dimensional parameter space of local symmetries (translation, reflection, rotation, uniform scaling). Li et al. [LLM07] use a different clustering approach to detect incomplete symmetries built from translation, reflection, and n-fold rotation of point sets. Even though the problem introduced in this paper is concerned with a *known* symmetry, namely point symmetry, methods such as these could help finding ambigrams with unexpected symmetries.

Z o pod dop pd oxo ped z peed deep pooed sos speeds VA SOS mu SSS SS S xxx xx x possessed s sexes sees esse noon e O non ose ene sums seas xerox suns passed VGA um one ere neon dup paned goll pard ore axe damp panned eme coos VOA pand paced HI oose allege erne nan Gog ecce slings mou umm fleg cees NIH cones fl senescence prod GB sus naan X sars ease are meu NH sass Pd cos gall azo dorp paved muumuu sans cores HRH comes

Figure 2: *An automatic process is employed to rank approximately 76,000 unique entries of a dictionary by their amount of point symmetry. The typeface of this example is "Aachen"; the best-ranked entries are shown.*

In a later work, Mitra et al. [MGP07] move past symmetry detection. They present a method to find approximate symmetries of polygonal 3D meshes, deform them to enhance the symmetry, or remesh them to create fully symmetric shapes. In principle, this approach could be applied to the task at hand, but would require a translation back and forth from cubic or quadratic curves to a mesh. This would reduce the graphical quality of the result.

3. Finding approximate ambigrams

The majority of words in the English language is not very promising when it comes to creating an ambigram. It is hard to see how words such as "Eurographics" could be rendered as point-symmetric shapes while retaining a basic amount of legibility. Hence, it is beneficial to start with looking for a word that lends itself well to form an ambigram. To this end, the software prototype steps through a dictionary, creates the representation for each word in a font specified in advance, checks for the degree of point symmetry and produces a rank list ranging from hits to near hits to weak hits, see Figure 2. This process is described in the following.

For testing purposes, a dictionary was compiled from a multitude of word lists found on the Internet. The largest list that has been incorporated stems from the American English spell check dictionary of OpenOffice.org. In total, the resulting dictionary contains nearly 76,000 unique words, a tiny fraction of which consists of acronyms or single letters.

To automate the ranking, an algorithmic definition of the degree of symmetry of a shape is required. To make the approach robust and simple, the words are rendered as bitmaps using standard built-in functions at a size of 30 points and a pixel depth of one bit, i. e. without antialiasing. Each bitmap is checked for symmetry under rotation by 180° degrees about the center of mass of the pixels that are "on."

A simple approach to measuring the deviation between the original shape and its (virtual) rotated copy would be to count how many pixels have different values for the two shapes, i. e. to determine the area of the symmetric set difference. This approach, however, would not take into account how far one shape extends beyond the other. A small remote mass of pixels in the difference set may not be hidden by

shape changes as easily as a large near mass. Hence, the counting approach is too simple.

A well-known measure of the geometric size of the deviation between the original shape and its (virtual) rotated copy is the Hausdorff distance [HKR93]. The Hausdorff distance, however, would be overly strict, as a single outlier can ruin the distance result of an otherwise perfect shape.

To blend these methods of measuring the deviation from perfect symmetry, the proposed method proceeds as follows: The bitmap image is converted into a discrete distance field with nine sets of distances ranging from zero pixels (that is, the point is contained in the shape) to eight or more pixels. To compute this distance field, the bitmap is convolved with a matrix of 15×15 pixels containing a cone. As typical typefaces are rather thin, this simple process runs adequately fast: The matrix is only, so to speak, stamped into the bitmap around the relatively few pixels that are set to "on."

Then, the software iterates over all "on" pixels of the virtual rotated copy and accumulates the distances of the original at the same positions. On first sight, this looks as though only gaps in the original shape below the rotated shape will be taken into account. However, protrusions of the original shape out of the rotated shape contribute as well because they appear as gaps at the rotated position.

The sum of the distances is divided by the total number of "on" pixels to provide an average distance. This final normalization ensures equal chances for long words in relation to short words and for all-capital acronyms in relation to words that contain only lowercase characters.

4. Editing with guaranteed symmetry

The central part of the solution consists of a vector graphics editor that produces shapes with enforced point symmetry. Its input consists of a word typed by the user typeset in a font selected by the user. The corresponding shape is built using standard calls of the Windows operating system to retrieve the outlines of characters. These are composed of line segments and quadratic Bézier curves, as this is all that is required for fonts in the TrueType format. To provide the standard editing functions with two tangential handles per Bézier segment, the prototype software, however, converts them to the more common cubic Bézier curves.

The software offers the standard editing capabilities such as selecting multiple handles or complete outlines and moving them using the mouse. Furthermore, the software produces rotational symmetry. Initially, the center of revolution is estimated from the bounding box of the total shape. At all times, the user can nudge this center point with the cursor keys. To help with this task and other tasks, a subdued image of the rotated shape is displayed in the background.

To define the symmetrization, the user has to draw a freehand polyline from the center point to the rim of the total shape. This polyline is automatically glued to a rotated copy of itself to become point-symmetric. All contours are cut along their intersections—if present—with this polyline. For the Bézier segments of the contours this requires solving a cubic equation (including degenerate cases) and splitting each segment that intersects the polyline. All parts and all complete contours *below* the polyline are removed. All parts and all complete contours *above* the polyline are copied, rotated and—where necessary—pasted to the contours above the polyline, see Figure 3.

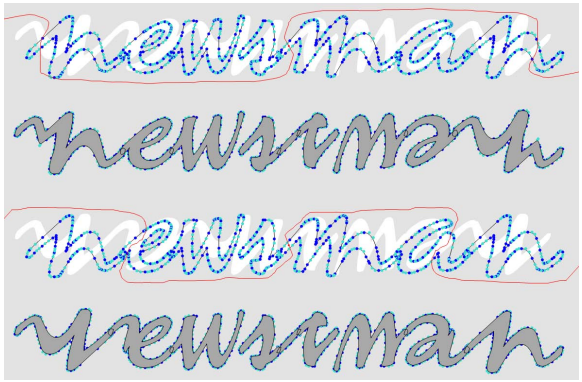

Figure 3: *The user draws a freehand polyline to define the cut (two examples shown). In many cases this leads to meaningful results without any adjustments to the handles of the contours.*

The possibly pointy connections produced by cutting and gluing are smoothed, which will be discussed in Section 5. All of this symmetrization happens in real time while the user edits the shapes or draws a new polyline to cut the shapes. Eventually, the result can be stored as an EMF vector graphics file.

5. Smooth fusion of vector shapes

The input to this step consists of a number of contours composed of lines and cubic Bézier curves, obtained by cutting and pasting. Where a contour meets the polyline used for cutting, a hard corner appears. The objective of the final step to be described now is to replace these corners by smooth Bézier transitions with adjustable softness, see Figure 4.

The basic idea of the smoothing process is to apply the following to each line that has been generated from the intersection between the polyline and a contour, see Figure 5: Starting from that line, move outward along the contour, in both directions, by an arc length that is specified globally by the user. Record the two resulting curve points and the speed vectors at these two places. Insert a Bézier segment that connects these two points. Pick tangent handles for this Bézier segment that continue in the directions of the derivatives. Choose the length of the tangent handles so as to optimize the curvature of the inserted piece. One special case has to

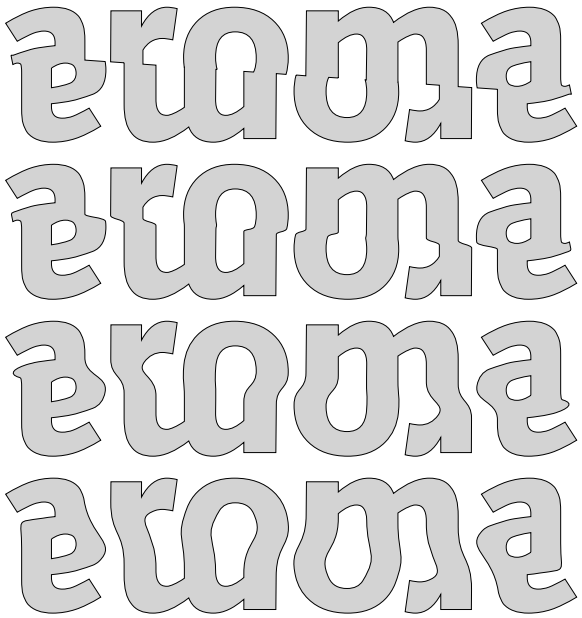

Figure 4: *The hard corners (topmost) produced by simple cutting and pasting are smoothed to a degree set by the user (second line to bottom).*

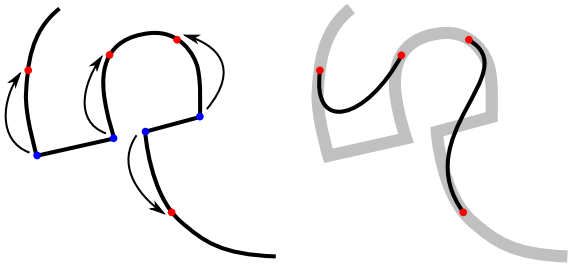

Figure 5: *The smoothing algorithm walks a given arc length outward from the intersections with the polyline (left). Then the points thus found are connected by a curve that picks up the tangent directions (right).*

be handled: Two intersections of the polyline with a contour may be so close in terms of arc length that the march outward as described before would lead to a collision. In this case, the new end point is placed on the mid point (in terms of arc length) on the contour between both intersections.

Choosing the right length of the tangent handles turned out to be difficult. A constrained minimization of the integral of the square of the second derivative of the curve produced overly flat connections. Hence, the software prototype uses a 1:1 blend of the length obtained by this optimization and the arc length walked along the contour.

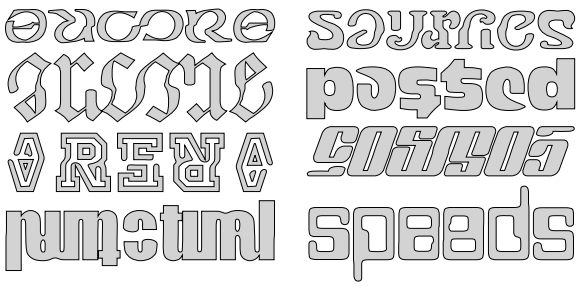

Figure 6: *Ambigrams created from some common typefaces.*

6. Conclusion and outlook

This paper presented a method to find approximate ambigrams and an interactive technique to create point-symmetric shapes from arbitrary vector graphics, see Figure 6. In particular the latter technique could have applications outside of the realm of ambigrams. For instance, one could provide soft versions of boolean operations in vector graphics editing software.

The dictionary search could be extended to provide suggestions for ambigram-friendly synonyms: “old” does not work well, but “antique” would be great. In addition, the system could recommend the best font for a word to become an ambigram. Another generalization could address ambigrams that appear as *different* words when rotated by 180 degrees.

References

[Bro03] BROWN D.: *Angels & Demons*. Atria Books, 2003. 1

[HKR93] HUTTENLOCHER D. P., KLANDERMAN G. A., RUCKLIDGE W. J.: Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*, 9 (1993). 3

[Hof85] HOFSTADTER D. R.: *Metamagical Themas: Questing for the Essence of Mind and Pattern*. Basic Books, 1985. 1

[Hol09] HOLST D.: The Ambimatic ambigram generator. http://www.ambigram.com/matic/, last accessed on 2010-02-23, 2009. 2

[Hun10] HUNTER M.: Glyphusion. http:http://glyphusion.com/, last accessed on 2010-02-23, 2010. 2

[IMKT97] IGARASHI T., MATSUOKA S., KAWACHIYA S., TANAKA H.: Interactive beautification: a technique for rapid geometric design. In *Proc. UIST ’97* (1997), pp. 105–114. 2

[LLM07] LI M., LANGBEIN F. C., MARTIN R. R.: Detecting approximate incomplete symmetries in discrete point sets. In *Proc. SPM ’07* (2007), pp. 335–340. 2

[MGP06] MITRA N. J., GUIBAS L. J., PAULY M.: Partial and approximate symmetry detection for 3D geometry. In *ACM SIGGRAPH 2006 Papers* (2006), pp. 560–568. 2

[MGP07] MITRA N. J., GUIBAS L. J., PAULY M.: Symmetrization. In *ACM SIGGRAPH 2007 Papers* (2007), p. 63. 2

[RMS97] RYALL K., MARKS J., SHIEBER S.: An interactive constraint-based system for drawing graphs. In *Proc. UIST ’97* (1997), pp. 97–104. 2