Computers & Graphics 56 (2016) 46–58

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on Expressive 2015

Advanced drawing beautification with ShipShape

Jakub Fišer [a,*], Paul Asente [b], Stephen Schiller [b], Daniel Sýkora [a]

[a] Czech Technical University in Prague, FEE, Czech Republic
[b] Adobe Research, United States

ARTICLE INFO

Article history:
Received 3 October 2015
Received in revised form
21 January 2016
Accepted 17 February 2016
Available online 3 March 2016

Keywords:
Drawing system
Input beautification
Vector graphics
Visual feedback

ABSTRACT

Sketching is one of the simplest ways to visualize ideas. Its key advantage is its easy availability and accessibility, as it require the user to have neither deep knowledge of a particular drawing program nor any advanced drawing skills. In practice, however, all these skills become necessary to improve the visual fidelity of the resulting drawing. In this paper, we present ShipShape—a general beautification assistant that allows users to maintain the simplicity and speed of freehand sketching while still taking into account implicit geometric relations to automatically rectify the output image. In contrast to previous approaches ShipShape works with general Bézier curves, enables undo/redo operations, is scale independent, and is fully integrated into Adobe Illustrator. We show various results to demonstrate the capabilities of the proposed method.

1. Introduction

Sketching with a mouse, tablet, or touch screen is an easy and understandable way to create digital content, as it closely mimics its real-world counterpart, pen and paper. Its low demands make it widely accessible to novices and inexperienced users. However, its imprecision means that it is usually only used as a preliminary draft or a concept sketch. Making a more polished drawing requires significantly more time and experience with the drawing application being used. Furthermore, when working with drawing or sketching software, users are often forced to switch between different drawing modes or tools or to memorize cumbersome shortcut combinations.

While we do not question the necessity or usefulness of complex tools to achieve non-trivial results, we argue that for certain scenarios, such as geometric diagram design or logo study creation, the *interactive beautification* [1] approach is more beneficial. Such workflows retain the intuitiveness of freehand input while benefiting from an underlying algorithm that automatically rectifies strokes based upon their geometric relations, giving them more formal appearance. With the quickly growing popularity of touch-enabled devices, the applicability of this approach expands greatly. However, whatever the potential of automatic beautification in a more general sketching context, most of the existing applications focus on highly structured drawings like technical sketches.

One of the biggest challenges in drawing beautification is resolving ambiguity of the user input, since the intention and its execution are often considerably dissimilar. Additionally, this issue becomes progressively more complex as the number of primitives present in the drawing increases.

In this paper, we present a system for beautifying freehand sketches that provides multiple suggestions in spirit of Igarashi et al. [1]. Strokes are processed incrementally (see Fig. 2) to prevent the combinatorial explosion of possible outputs. Unlike previous work, our approach supports polycurves composed of general cubic Bézier curves in addition to simple line segments and arcs. The system is scale-independent, and can easily be extended by new operations and inferred geometric constraints that are quickly evaluated and applied. The algorithm was integrated into Adobe Illustrator, including undo/redo capability. We present various examples to demonstrate its practical usability (Fig. 1).

2. Related work

The need to create diagrams and technical drawings that satisfy various geometric constraints led to the development of complex design tools such as CAD systems. However, these systems' complexity often limits their intuitiveness. Pavlidis and Van Wyk [2] were one of the first to try to alleviate this conflict by proposing a method for basic rectification of simple rectangular diagrams and flowcharts. However, their process became ambiguous and prone to errors when more complex drawings were considered, since the method needed to drop many constraints to keep the solution tractable.

* Corresponding author.
E-mail address: fiserja9@fel.cvut.cz (J. Fišer).

http://dx.doi.org/10.1016/j.cag.2016.02.003
0097-8493/

Fig. 1. Examples of drawings created using ShipShape. The final drawings (black) were created from the imprecise user input (gray) by beautifying one stroke at a time, using geometric properties such as symmetry and path identity. See Fig. 17 for more results.

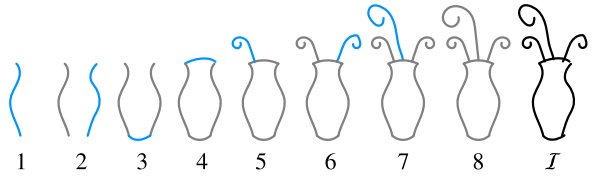

Fig. 2. Incremental beautification workflow. Every newly drawn stroke (blue) is beautified using previously created data (gray). The first stroke is left unchanged. As the drawing continues, more suitable geometric constraints emerge and are applied, such as path identity (2,6,7), reflection (2,6) or arc fitting (3,4). For comparison with the final beautified output (8), $\mathcal{I}$ shows the original input strokes. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

To alleviate this limitation, Igarashi et al. [1] proposed an interactive beautification system in which the user added strokes one by one and the system improved the solution incrementally while keeping the previously processed drawing unchanged. This solution kept the problem tractable even for very complex drawings. Moreover, the system also presented several beautified suggestions and let the user pick the final one. This brought more user control to the whole beautification process. Following a similar principle, other researchers developed systems for more specific scenarios such as the interactive creation of 3D drawings [3], block diagrams [4,5], forms [6], and mathematical equations [7].

However, a common limitation of the approaches mentioned above is that they treat the image as a set of line segments. To alleviate this drawback Paulson and Hammond [8] proposed a system called *PaleoSketch* that fit the user input to one of eight predefined geometric shapes, such as line, spiral or helix. In a similar vein, Murugappan et al. [9] and Cheema et al. [10] allowed line segments, circles and arcs.

Related to drawing beautification, there are also approaches to beautify curves independently, without considering more complex geometric relationships. Those approaches are orthogonal to our pipeline. They use either geometric curve fitting [11,12] or some example-based strategy [13,14]. Additionally, advanced methods for vectorizing and refining raster inputs have been proposed [15,16], which enable users to convert bitmap images into high quality vector output. However these do not exploit inter-stroke relationships. In our case we assume that the built-in curve beautification mechanism of Adobe Illustrator preprocesses the user's rough input strokes into smooth, fair paths.

This paper extends our previous work [17]. In Section 3.1 we discuss improvements to the arc and circle center rules, and introduce a generalized transformation adjustment framework. Section 3.4 describes a new method for curve alignment, and Section 3.5 describes the transformation adjustment mechanism in detail. Finally, Section 4 describes a new framework for handling curves with corners.

3. Our approach

A key motivation for our system is wanting to work with arbitrarily curved paths. This capability was not available in previous beautification systems. Although some can recognize a variety of curves including spirals and general 5th degree polynomials (*PaleoSketch* [8]), they recognize them only in isolation and do not allow to take other existing paths into consideration, which is important for interactive design.

Systems like that of Igarashi et al. [1] generate a set of potential constraints and then produce suggestions by satisfying subsets of these. A key challenge that prohibits simply generalizing these systems to support general curved paths is the number of degrees of freedom, which boosts the number of potential constraints that need to be evaluated. Moreover, unlike line or arc segments, many of a general path's properties, for example the exact coordinates of a point joining two smooth curves, do not have any meaning to the user. It would not be helpful to add constraints for this point. Finally, satisfying constraints on a subset of the defining properties might distort the path into something that barely resembles the original. Supporting generalized paths requires a different approach.

Our system is based on an extensible set of self-contained geometric rules, each built as a black box and independent of other rules. Every rule represents a single geometric property, such as having an endpoint snapped or being a reflected version of an existing path. The input to each rule is an input path consisting of an end-to-end connected series of Bézier curves, and the set of existing, resolved paths. The black box evaluates the likelihood that the path conforms to the geometric property, considering the resolved paths, and outputs zero or more modified versions of the path. Each modified version gets a score, representing the likelihood that the modification is correct.

For example, the same-line-length rule would, for input that is a line segment, create output versions that are the same lengths as existing line segments, along with scores that indicate how close the segment's initial length was to the modified length. Each rule also has some threshold that determines that the score for a modification is too low, and in that case it does not output the path.

The rules also mark properties of the path that have become fixed and therefore can no longer be modified by future rules. For example, the endpoint-snapping rule marks one or both endpoint coordinates of a path as fixed. The same-line-length and parallel-line rules do not attempt to modify a segment with two fixed endpoints.

Since the rules do not depend on each other, it is easy to add new rules to support additional geometric traits. Fig. 3 shows an illustrated list of rules supported in our system.

Chaining the rules can lead to complex modifications of the input stroke and is at the core of our framework. We treat the rule application as branching in a directed rooted tree of paths, where the root node corresponds to the unmodified input path. Each