Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

The Computer-Based Generation of Fonts in the Style of Kandinsky

Kang Zhang (educator), Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083-0688, USA. Email: kzhang@utdallas.edu. ORCID: 0000-0003-3802-7535.

Jinhui Yu (educator), State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou, 310027, P. R. China. Email: jhyu@cad.zju.edu.cn.

Manuscript received 16 April 2019.

Abstract
This paper presents a general framework for programmed automatic generation of artistic fonts. By parameterizing various font attributes, such as color and aspect ratio, we are able to generate artistically styled fonts in almost unlimited variations to suit any types of design requirements. We demonstrate our experiments on generating fonts of Kandinsky abstract style, built on a collection of the artist's styled patterns. Our approach generates fonts composed of vector strokes and is thus highly scalable, limited only by the computer hardware.

Keywords: artistic font, generative art, Kandinsky

1 Introduction

As a pioneer of abstract paintings, Wassily Kandinsky has influenced generations of abstract artists and designers. During his Bauhaus years from 1922 to 1933, Kandinsky was interested in the composition and interaction of geometrical forms, and also published his well-known theoretical art book [0]. Among many design achievements during this period, his abstract styles of paintings have been modeled for modern designs by various new technological means.

An ability of modeling and generating abstract arts of aesthetic and/or well-known styles automatically in a digital form has a profound implication in art and design. First, it would facilitate more systematic and scientific study and thus enable insightful understanding of such styles. Second, generative art integrating different forms and styles becomes possible, such as static, animated, flashed displays and so on. Third, it enables digital creativity in generating new and possibly synthetic styles. Finally, it eases the commercial application of abstract art in forms, such as font generation, logo design, advertisements, packaging, and decoration at a negligible cost [0].

Limited attempts have been made for using heuristic rules and patterns to encode well-known artistic styles and visual elements and then generating digital forms of specific styles of paintings. Recent advances in deep learning using neural networks and the applications in transferring artistic styles [0][0] have produced encouraging results, mostly at a textual or low (non-semantic) level, thus particularly suited for styles like impressionism.

Based on Kandinsky's art theory [0][0] and our analysis of various visual elements used in his paintings in his Bauhaus period, our first project was to generate various artistic forms in Kandinsky style. Our initial work on automatic generation of Kandinsky abstraction images

Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

included very few styled patterns and generated uniform backgrounds and no sophisticated combination of geometric objects [0]. We then conducted an in-depth and comprehensive experiment on generating several styles of Kandinsky's paintings mainly in his Bauhaus period [0]. That project did not include any font design and generation.

This paper presents our recent work on automatically generating vector fonts (rather than bitmap fonts) in a customizable font-generation framework. This generative framework with parametrization of various visual attributes has the following advantages:

- Vector fonts can be generated at almost any size and any resolution, limited only by the computer's own memory and screen sizes.
- Most of the font attributes, such as color, height vs width aspect ratio, tilting angle (e.g. italics), and letter-spacing are all customizable.
- Fonts may be generated within any graphic design space to include desirable interaction, such as transparent or semi-transparent overlapping.

To summarize, fonts are essentially programmable for many desirable design effects.

The rest of this paper is organized as follows. Section 2 first proposes a general framework for generating customizable font sets, with detailed examples of programming Kandinsky's styles and various customized fonts generated in the programmed styles. Section 3 discusses the scalability of our approach and potential implications of such a generative framework, followed by Section 4 that reviews related work. Finally, Section 5 concludes the paper and mentions possible future directions.

2 Customizable Styled Font Generation

This section first provides an overview and explanation of our generation framework and then example patterns styled on Kandinsky's geometric abstract paintings and how they are encoded. The section ends with the results of several generated fonts of various design effects.

2.1 A Font Generation Framework

The key ideas and our methodology are depicted in the flowchart in Figure 1. Using the library built for automatic generation of Kandinsky styled images, we have encoded all the 26 uppercase, 26 lowercase, and 10 numeric letters into a font base, where each letter's key parameters, such as colors, could be set via the user input. This is illustrated by the top-right two boxes marked "Styled Patterns" (collection from many of the artist's paintings, as detailed below) and "Parameterizable Font Base", which form the foundation of the font generation framework.

In Figure 1, dashed lines represent user-provided customization and specification inputs. The user, typically designer, could specify colors differently for either individual letters or uniformly for the entire font set, and whether the colors should be randomized or not, if so, providing the random range in RGB values for each color (note that all the following example generated fonts are individually colored). The user's request would then be handled by the service marked "RGB

Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

Coloring". The user could also specify the size of the font set, the height vs width aspect ratio, and whether they should be italicized, to be handled by the services marked "Scaling" and "Sheering" respectively. "Scaling" may individually set the height and width and thus handles both the font size and aspect ratio. The default aspect ratio is the golden ratio, if not specified by the user. Finally, letter-spacing (sometimes known as *tracking* in typography), i.e. the uniform space between any two neighboring letters, could also be specified. To achieve a visually pleasing typesetting effect, one could require special spacing, or kerning, for specific combinations of letters, such as "AV".

The framework in Figure 1 conceptually is not limited to Kandinsky style. Any specific and/or artistic style and even a mixture of multiple styles could be encoded into patterns for generating interesting font sets.

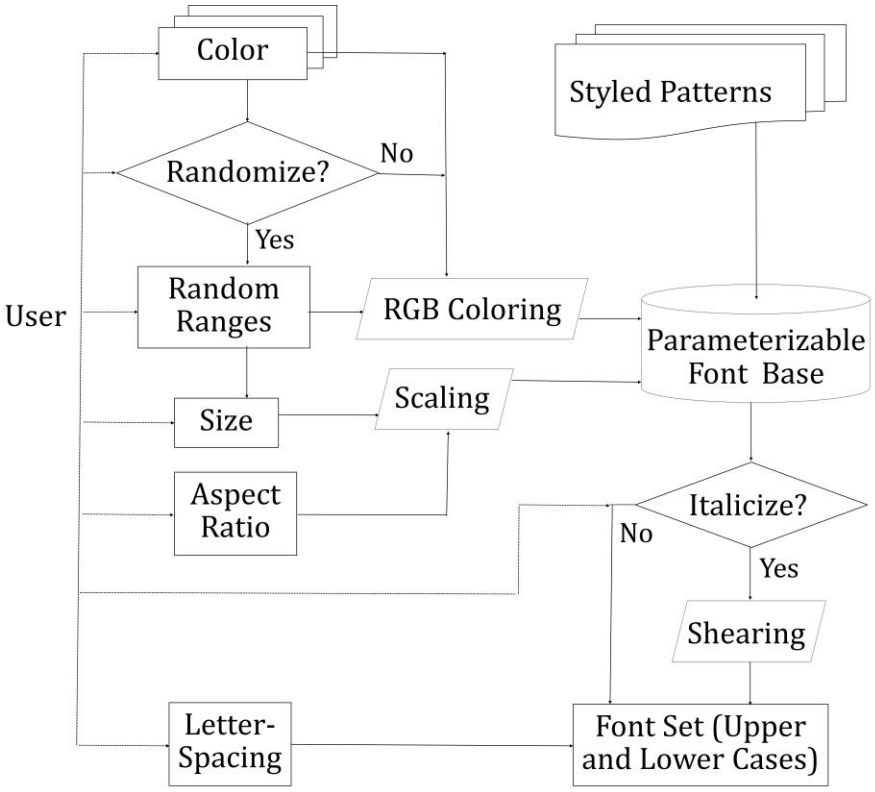

Figure 1: Customizable font generation framework. (© Kang Zhang)

2.2 Kandinsky Styled Patterns

We will call typical structures composed of geometrical forms found in the artist's paintings as *styled patterns.* Having modeled many of Kandinsky's abstract paintings during his Bauhaus period, including *Composition VIII (1923), Black and Violet (1923), On White II (1923), Several Circles (1926), Yellow, Red, Blue (1925),* and *Soft Hard (1927),* we have a large collection of styled patterns. With rich experience of encoding the artist's style, we are able to adapt and modify these patterns to suit English font strokes and structures.

Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

One of the most representative works during the Bauhaus period is *Composition VIII* [0], that includes many patterns suitable for font strokes. Figure 2 shows an automatically generated image modeled on *Composition VIII* after parameterization and randomization.

To demonstrate our approach in drawing styled patterns, we take an example pattern from *Soft Hard (1927),* shaped like a boomerang (a tool used by Australian aboriginal people). The shape shown in Figure 3 is generated by the Processing program segment in Figure 4. The program allows the attributes of position (i.e. X and Y), color (R, G, and B), rotation angle (Angle), and a rectangle of width (W) and height (H) to be assigned by another program at a higher level. In our case, this high-level program is a letter rendering program (e.g. for 'K') that needs to draw the boomerang shape as a stroke. The cubic Bézier curve function bezierVertex uses the rectangle's width W and height H to set the endpoints and control points, as illustrated in Figure 3. The curve function is called twice to draw the shape, filled with the specified color. The translate and rotate statements allow the pattern to be positioned and angled as specified.

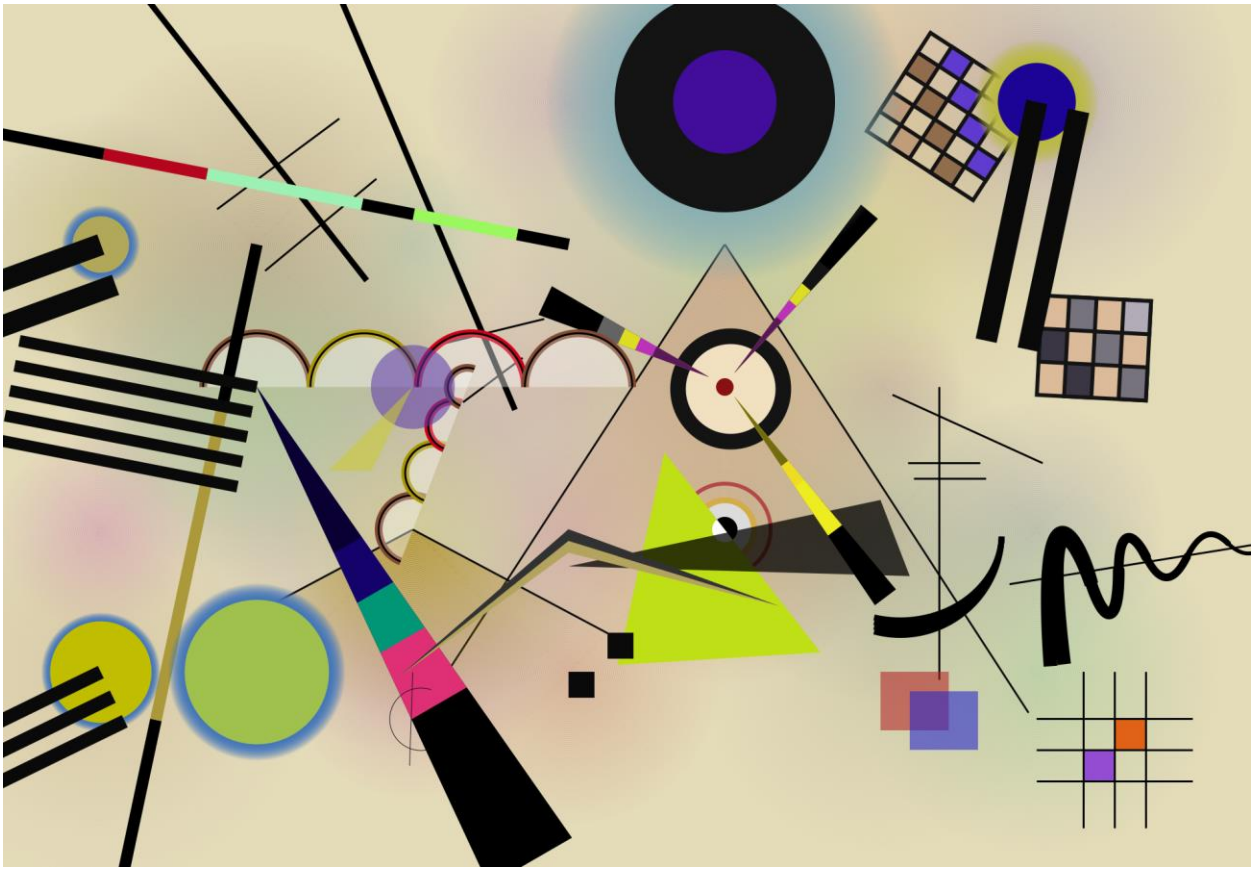

Figure 2: Randomly generated image modeling Kandinsky's *Composition VIII (1923)*. (© Kang Zhang)

Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

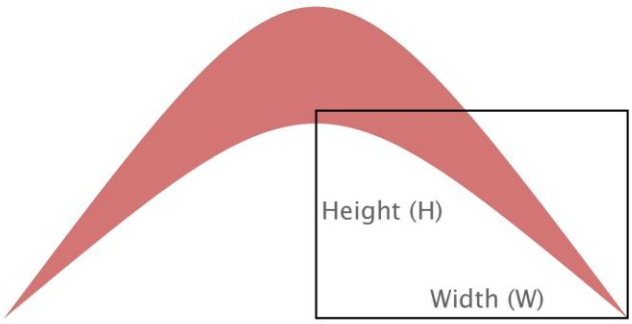

Figure 3: Boomerang shape in *Soft Hard (1927)*. (© Kang Zhang)

```
void boomerang(float R, float G, float B, int X, int Y, float Angle, float W, float H) {
      translate(X, Y);
      rotate(radians(Angle));
      noFill();
      strokeWeight(0); // minimal line width
      stroke(R, G, B);   // stroke color
      fill(R, G, B, 200);  // fill color at transparency level of 200
      beginShape();
      vertex(-W, H);
      bezierVertex(0, -H, 0, -H, W, H);  // draw top curve
      bezierVertex(0, -H/4, 0, -H/4, -W, H); // draw bottom curve
      endShape();
      rotate(radians(-Angle));
      translate(-X, -Y);
}
```

Figure 4: Program segment for rendering a boomerang shape. (© Kang Zhang)

2.3. Example Font Effects

By parameterizing various visual attribute, we are able to generate almost unlimited variations of fonts. A sample set of automatically generated 26 letters in both uppercase and lowercase is shown in Figure 5. Figure 6 shows example texts, including a regular type, an italic type, a vertically sheered type, and an example logo design. The sheering angle (or slope) is set at -20 degrees along X-axis for the italic example and -20 degrees along Y-axis for the vertical sheering example. The angle could be set at any value as desired. It may be noted that parts of a letter have the same color on different appearances of the same letter, such as the black line in the middle of the stroke in ‘n’, while other parts change colors. Some strokes are fixed at specific colors to closely resemble Kandinsky’s style. In fact, any part of a letter could be assigned with any color, fixed or within a random range of RGB values for any design needs.

Figure 7 demonstrates example letters of varied width vs height aspect ratios and varied sizes while maintaining the same resolution. We may also set the background as transparent so that only font strokes are visible, as shown in program generated “On White II” (1923) variation in Figure 8. The abstract image was generated before rendering the text “On White II” at the appropriate position on the image.

Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

Figure 5: A sample set of generated 26 uppercase and 26 lowercase letters. (© Kang Zhang)

Figure 6: Examples of generated texts: (a) regular, (b) italic, (c) vertical sheering, (d) a logo design. (© Kang Zhang)

Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

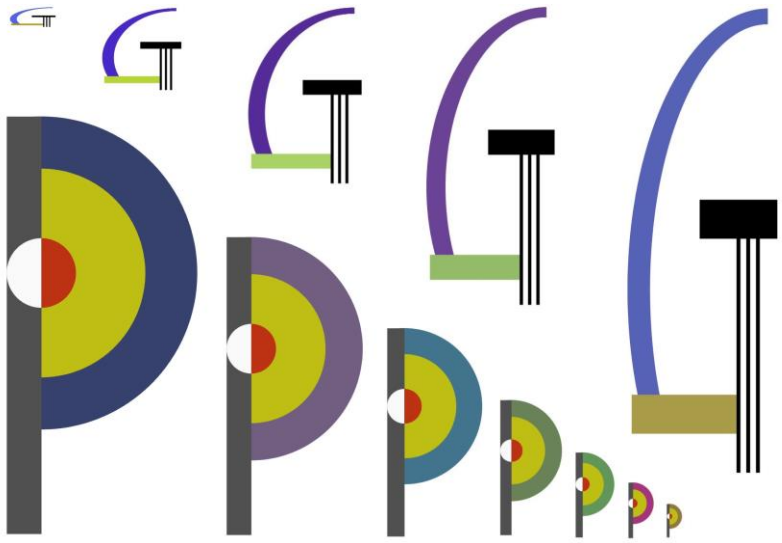

Figure 7: Scalable font size and aspect ratio. (© Kang Zhang)

Figure 8: Text "On White II" with the transparent background generated on the program-generated Kandinsky's *On White II (1923)* image. (© Kang Zhang)

Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

Simple animated effects are demonstrated in a short video clip uploaded as a supplementary material. The video records the output console during our program execution, that animates the changing shapes and colors of two individual strokes (a triangle and a boomerang shape) forming the capital letter 'K'. More complex effects, possibly including interactions among different patterns and between generated fonts and other elements of a design space, could also be implemented as needed.

The generated font and text image could finally be saved as a file of any of the .png, .tiff, and .jpg type, whose size is determined by the canvas dimension set in the set-up part of the Processing program. The size of the image file (and thus the font size) could be as large as that supported by the Processing's internal memory.

3 Scalability and Implications

The generation program is highly scalable with the size of the generated fonts, using Processing's "scale" statement. The only limitation to the image size is the machine's memory space set within the Processing language and the screen resolution. We run our program on an Apple iMac with 8 GB memory, an Intel Core i5 processor at 3.2 GHz, and a Graphics AMD Radeon R9 M390 2048MB. Setting the Processing language's memory at 6GB, we have been able to generate fonts on the canvas of 20,000 x 15,000 pixels. Such an image in the .tiff format requires a storage space of 900 MB.

We have experimented with the generation speed by running the program to generate a single uppercase letter and again to generate all 26 uppercase letters in a continuous sequence (overlaying each other simply to test the speed). Each letter is sized 13,000 x 9,000 pixels on the canvas of 20,000 x 15,000 pixels, the generation process took about 25 seconds in both cases. Similarly, it took about 22 seconds to generate a single lowercase letter or all 26 lowercase letters of the same size. This implies that the majority of time spent is not on rendering fonts, but on the setup process and memory access. Our program has not been optimized for either speed or memory usage.

Our approach has apparently immediate application in the graphic design field. Furthermore, since various font attributes could be changed at run-time, various dynamic effects, such as animation and flashing color, could be achieved. The example effects demonstrated in the supplementary video are easily achieved by a few lines of code in our programmed font generation framework, but impossible to achieve with any manually designed fonts. Dynamic effects suit visual marketing [0] extremely well, particularly recently emerging field of computational advertising. Even though the patterns encoded are all styled on Kandinsky's geometric paintings during his Bauhaus years, the presented approach is equally applicable to generating fonts of other artistic styles as long as the styled patterns are available and font structures are pre-coded.

4 Related Work

Commercial online font generation systems, such as Font Meme (https://fontmeme.com/art-deco-fonts/), are usually uncustomizable, uniformly colored, and more importantly, unable to

Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

generate a specific artist's styles. Kandinsky styles of paintings have been modeled by some researchers. For example, Barnett and Barnett [0] analyzed Kandinsky's compositions, while Price [0] applied Kandinsky's aesthetics to Java programming. We previously analyzed the artist's visual elements and compositions in his abstract paintings, and generated many unique abstract images of the same style [0][0]. Others attempted to create Kandinsky fonts, such as those published online (http://luc.devroye.org/kandinsky.html and https://www.dafont.com/kandinsky.font). These fonts are not quite close to the artist's real style, in color or in shape. Not generated via computer programming, they do not offer any of the aforementioned advantages that our work offers. There has been no academic work on generating Kandinsky style of fonts in the literature.

Recent advances in artificial intelligence has renewed interests in generating or modeling art works. One such example is AARON, developed by Harold Cohen (http://www.kurzweilcyberart.com/). Xu et al. [0] also proposed an intelligent system based on constraint-based analogous-reasoning to automatically generate Chinese calligraphy after training a few training examples of existing calligraphic styles. These are all supervised learning approaches, relying on the available training images.

Campbell and Kautz [0] built a generative manifold of standard fonts such that very location on the manifold corresponds to a unique typeface. They obtain the locations via interpolation and extrapolation operations on existing fonts. The approach works well for standard fonts, not for any artistic style. Similarly, TYPE+CODE II [0] combines aspects of traditional hand-drawn calligraphy to generate aesthetic typography with contemporary elements using mathematical expressions and algorithms. Again, it is not modeled on any specific artistic style.

Goda et al. [0] used continuous structure of texture patches to model the artistic brush work of Chinese and Japanese calligraphy. It is limited to calligraphy which is usually of monochrome, apart from focusing only on the textural features of brushwork. The open source Processing projects also include a generative typography site http://generative-typografie.de. Various fancy transformed fonts could be generated here, but none of the fonts are modeled on any artistic style in both colors and patterns.

On font customization, Cheng [0] proposed a scalable method that allows a designer to use a CAD tool to specify the key points of font strokes, so that a set of font could be generated automatically. This method, however, only customizes the font outline, without any artistic styling. Another patent by Unruh [0] claims a font database and a set of rule for composite font generation. The database includes many bitmap files representing font characters that could not be customized or even modified.

5 Conclusion

This paper has presented our recent work on automatic generation of artistic style of font sets, built on our previous studies on Kandinsky's Bauhaus style of abstract paintings. The generated fonts could be readily used for graphic design and the framework and approach could be integrated within a graphic design system, such as a logo design system [0], for generation of harmonious graphic designs. Since fonts are program generated, dynamic features, such as

Leonardo Just Accepted MS.
https://doi.org/10.1162/leon_a_01888

flashing/changing colors and animation, could be easily achieved. One limitation of our approach is that although the customizable approach used is generic and scalable, visual patterns of other styles would need to be encoded before a fond base could be generated. The structure of each letter is also hard-coded and thus fixed.

More styled patterns could, however, be added and the styled encoding can also be further fine-tuned to generate patterns with more details. Patterns with mixed styles of multiple artists could also be encoded. We believe that there should be many opportunities to further extend the work. Our immediate future work is to use the same approach to generate fonts in Miro's surrealism style based on our research in modeling Miro [0]. Another future work is to use recently advanced deep learning tools to learn and generate different styled font sets. The techniques here are termed style transfer, rather than style generation, as there would be a training set to start with.

6 Acknowledgment

The work is partially supported by the National Natural Science Foundation of China under Grant No. 61772463.

References

1. 1W. Kandinsky, *Point and Line to Plane*, (originally published in 1926) translated by H. Dearstyne and H. Rebay (Dover Publications Inc., New York,1979).

2. 2J. Fogarty, J. Forlizzi, and S.E. Hudson, "Aesthetic Information Collages: Generating Decorative Displays that Contain Information", *Proceedings of 14th Annual ACM Symposium on User Interface Software and Technology (UIST'01)*, 141-150 (ACM, 2001).

3. 3L. A. Gatys, A. S. Ecker, and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks", *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'2016)*, 2414–2423 (IEEE, 2016).

4. 4X. Huang and S. Belongie, "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization", *Proc. 2017 IEEE International Conference on Computer Vision (ICCV'2017)*, Venice, Italy, 1501-1510 (IEEE, 22-29 October 2017).

5. 5W. Kandinsky, *Concerning the Spiritual in Art*, (originally published in 1914) translated with an introduction by M.T.H. Sadler (Dover Publications Inc., New York,1977).

6. 6K. Zhang, S. Harrell, and X. Ji, "Computational Aesthetics - On Complexity of Computer-Generated Paintings", *Leonardo Journal* **45**, No. 3, 243-248 (2012).

7. 7K. Zhang and J.H. Yu, "Generation of Kandinsky Art", *Leonardo* **49**, No.1, 48-55 (2016).

8. 8W. Kandinsky, *Composición VIII*. Solomon R. Guggenheim Museum, New York (1923).

9. 9B.H. Schmitt, and A. Simonson, *Marketing aesthetics: The Strategic Management of Brands, Identity and Image* (New York: The Free Press, 1997).

10. V.E. Barnett and P.H. Barnett, "The Originality of Kandinsky's Compositions", *The Visual Computer* **5**, No. 4, 203-213 (July, 1989).

11. C.B. Price, "From Kandinsky to Java - The Use of 20th Century Abstract Art in Learning Programming", *ITALICS* **6**, No.4, 35-50 (October 2007).

12. S. Xu, F.C.M. Lau, W.K. Cheung, Y. Pan, "Automatic Generation of Artistic Chinese Calligraphy", *IEEE Intelligent Systems* **20**, No.3, 32-39 (May-June 2005).

13. N.D.F. Campbell and J. Kautz, "Learning a Manifold of Fonts", *ACM Transactions on Graphics* **33**, No.4, Article No. 91 (July 2014).

14. Y. Ahn and G. Jin, "TYPE+CODE II: A Code-Driven Typography", *Leonardo* **49**, No. 2, 168-168 (2016).

15. Y. Goda, T. Nakamura, M. Kanoh, "Texture Transfer Based on Continuous Structure of Texture Patches for Design of Artistic Shodo Fonts", *ACM SIGGRAPH ASIA 2010*, Sketches, Article 18.

16. K.Y. Cheng, Stroke-based Font Generation, *US Patent US6157390A* (2000).

17. E. Unruh, Method and Apparatus for Composite Font Generation, *US Patent US6678688B1* (2004).

18. Y-N. Li, K. Zhang, and D.J. Li, "Rule-Based Automatic Generation of Logo Designs", *Leonardo* **50**, No.2, 177-181 (2017).

19. L. Xiong and K. Zhang, Generation of Miro's Surrealism, *Proceedings of 9th Symposium on Visual Information Communication and Interaction (VINCI'2016)*, Dallas, USA, 130-137 (ACM, 24-26 September 2016).

KANG ZHANG is a professor of computer science at the University of Texas at Dallas, ACM Distinguished Speaker and Fulbright Distinguished Chair. His research interests include visual languages, computational aesthetics, generative art and design.

JINHUI YU is a professor of computer science at the State Key Lab of CAD & CG, Zhejiang University, China. His research interests include image-based modeling, nonphotorealistic rendering, computer animation and computer art.