

Differentiable Variable Fonts

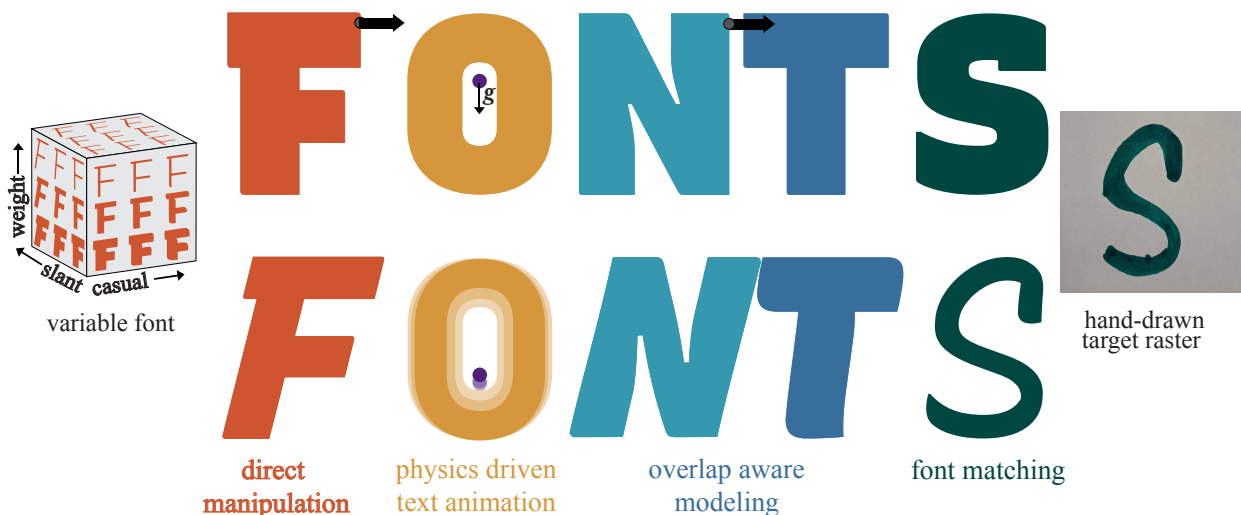
Kinjal Parikh^{1,2}, Danny M. Kaufman^{1,2}, David I.W. Levin^{1,3}, Alec Jacobson^{1,2}¹University of Toronto, ²Adobe Research, ³NVIDIA

Figure 1: We introduce differentiable variable fonts: a mathematical formulation of variable font interpolation made differentiable, enabling gradient-based optimization directly in variable font space. This combines the inherent design guarantees of fonts (legibility, readability, stylistic consistency) with direct differentiable control over text geometry. From left to right: ("F") direct manipulation that adapts glyph shapes while preserving design intent, ("O") physics-driven text animation, ("NT") overlap-aware modeling of glyph interactions, and ("S") font matching with a hand-drawn raster target.

Abstract

Typography is an essential component of visual communication. Editing and animating text appearance for graphic designs, title sequences, commercials, and logos remain highly skilled tasks requiring detailed, hands-on efforts from pro artists. Automating these challenging manual workflows requires balancing the competing goals of maintaining a text's legibility and aesthetics, while enabling creative expression. Variable fonts, recent parametric extensions to traditional fonts, offer the promise of new ways to ease and automate typographic design and animation. Variable fonts provide custom-constructed parameters along which fonts can be smoothly varied. These parameterizations could then potentially serve as high-value continuous design spaces, opening the door to modern automated design-optimization tools. However, currently variable fonts are underutilized in creative applications, exactly because artists so far still need to manually tune font parameters. Our work provides intuitive and automated font design and animation workflows with differentiable variable fonts. To do so we distill the current variable font specification to a compact mathematical formulation that differentially connects the highly non-linear, non-invertible mapping of variable font parameters to the underlying vector graphics representing the text. In turn, this enables us to construct a differentiable framework, with respect to variable font parameters, that allows us to perform gradient-based optimization of energies defined on vector graphics control points, and likewise, via differentiable SVG rasterization, on target rasterized images. We demonstrate the utility of this framework with a range of applications, including direct shape manipulation, overlap aware modeling, physics-based text animation, and automated font-design optimization. Our work now enables leveraging the carefully designed affordances of variable fonts with differentiability to use modern design-optimization technologies, and so opens new possibilities for easy, intuitive and expressive typographic design workflows.

CCS Concepts

• **Computing methodologies** → Parametric curve and surface models; • **Applied computing** → Media arts;

1. Introduction

Text is a fundamental element of visual communication, used across design, animation, and digital media. But editing and animating text remains a challenging task that demands significant time and manual effort from artists. Current workflows for manipulating typography rely on converting text, typed in a fixed font instance, into images, meshes, or SVG paths. However, this conversion from text to graphics representations is a one-way process that divorces the geometry from the font, making it hard to retain *legibility, readability, and consistent style* (See Fig. 3). This disconnect poses a challenge for developing tools for text manipulation and animation.

To address this disconnect between graphics tools and typography, we build on a recent font technology: variable fonts. A variable font is a single font file that stores a continuous range of design variants. Instead of offering a fixed style, a variable font exposes one or more parameters, called *axes*, that control stylistic variation. Each axis corresponds to a numerical value that can be adjusted independently and may correspond to conventional typographic properties such as weight or width, or to custom dimensions defined by the type designer (See Fig. 2). This continuous design space is powerful – as long as the text manipulation is constrained to a variable font, it will be legible and consistent in style. However, variable fonts are underutilized in creative applications because artists need to manually adjust each parameter in this often high-dimensional space.

We formalize the existing OpenType variable font interpolation mechanism in a compact mathematical model, revealing the highly non-linear, non-invertible mapping of variable font parameters to the underlying vector graphics representing the text (Section 3). Based on our formulation, we implement variable font interpolation in a way that exposes gradients with respect to a font's axes, enabling gradient-based optimization directly in the variable font space (Section 4). This makes it possible to cast a broad range of graphics editing tasks as energy minimization problems defined directly over variable font parameters.

We showcase the utility of our differentiable framework through four applications spanning interactive design, animation, and analysis. First, we introduce an inverse-kinematics-style interface for direct glyph manipulation, enabling designers to edit shapes via geometric targets rather than abstract axis values (Section 5.1). Second we support overlap-aware modeling that automatically resolves collisions (Section 5.2), and third a physics-inspired simulation pipeline that animates texts with forces and constraints, while preserving legibility and style. (Section 5.3). Fourth, we demonstrate an image-based font matching optimization, that finds variable font instances that best-capture free-form raster image input targets (Section 5.4).

2. Related Work

2.1. Parametric Font Representations

The idea of fonts as continuous design spaces dates back to early systems like Knuth's METAFONT [Knu79], and Adobe's Multiple Master fonts [Ado97], which explored parameterized or inter-

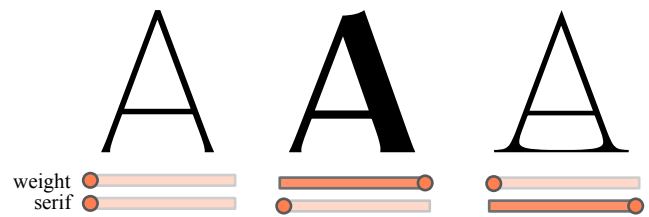


Figure 2: A variable font with weight and serif as variable axes.

polated glyph design. These efforts laid the groundwork for modern variable fonts [Mic18], which extend the OpenType standard to support continuous variation along designer-defined axes.

Alongside industrial developments, academic research has explored parametric representations of fonts—ranging from early work that treated high-level glyph features like stems and serifs [SR98] or reusable components [HH01] as parameters, to more recent approaches that learn continuous font spaces from examples to support interpolation [CK14]. While these methods can be effective for font generation, they often offer limited user control over the design space — an important requirement in applications like logo design, title sequences, and motion graphics, where maintaining stylistic consistency is critical for brand identity. Variable fonts provide a practical alternative: they define a constrained, interpretable space crafted by type designers, allowing controlled manipulation without sacrificing design intent. They are also supported across all major browsers and enable efficient optimization, making them well-suited for deployment in real-world design tools. Rather than proposing a new font representation, our contribution lies in providing a concise mathematical formalization of the existing OpenType variable font mechanism and making it differentiable, enabling gradient-based optimization.

2.2. Font Manipulation

Digital fonts represent each glyph's geometry as a collection of Bézier curves. In standard design workflows, text is converted to vector paths and edited using general-purpose tools such as Adobe Illustrator, Figma, Inkscape, or Affinity Designer. This detaches the text from its typographic structure, reducing it to unstructured vector geometry. While this provides flexibility, making precise edits requires multiple iterations of careful curve manipulation to preserve legibility and aesthetic appeal. The process is time-consuming and demands expert judgment. Likewise, automatic shape manipulation techniques from geometry processing operate on vector graphics or meshes without regard for typographic structure, and can similarly compromise legibility when applied to text. Moreover, many such techniques aim to preserve area, length, or angles under deformation [IMH05; SA07; WXW*06; CPSS10; WG10; SBBG11], making them ill-suited for common typographic modifications, such as increasing stroke weight or applying slant, which alter area and internal angles in ways these methods are not designed to accommodate. (See Fig. 3)

Attempts to preserve meaningful structure during 2D or 3D

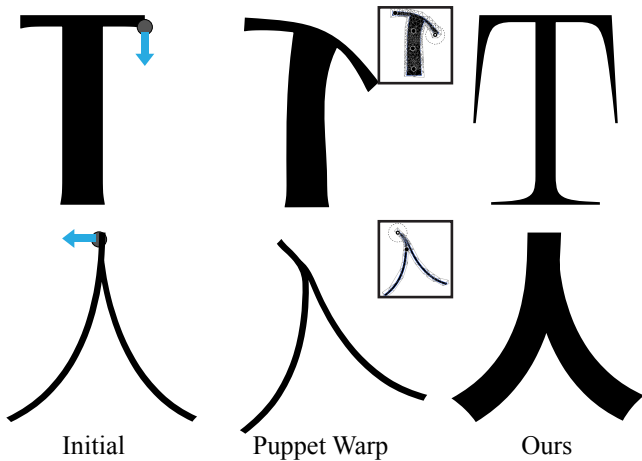


Figure 3: General-purpose direct manipulation often compromises legibility and stylistic consistency. We compare with Puppet Warp [LJG14], default settings with automatically generated pins (see inset). For character *T*, our method maintains style by cleanly extending the serif when the indicated point is dragged downward, whereas Puppet Warp produces distortions. For the Chinese character 人 (person), Puppet Warp deforms the shape into 入 (enter), a different character. Our method preserves readability, legibility, stylistic consistency.

shape editing have a long history across various application domains. Some methods rely on user-defined constraints [Gle92; HLW93; Sut64], while others attempt to automatically detect and preserve salient structures [CLDD09; BL15; AVR*22]. In contrast, variable fonts are carefully constructed by type designers to ensure that every point in the design space yields a legible and stylistically coherent instance. By allowing the end user to select an appropriate variable font, we can take advantage of this expert-authored structure, with the assurance that edits confined to the font’s defined variation space will preserve legibility.

2.3. Kinetic Typography

Kinetic typography, or text animation, is a well-established motion graphics technique used to capture audience attention [Bor04; KM08]. The Kinetic Typography Engine [LFH02] laid the groundwork for modern animation software (e.g., Adobe After Effects and TypeMonkey), where designers manually keyframe properties such as position, scale, rotation, and opacity. For novice users, presets offer one-click animations, though these are often rigid and provide limited creative flexibility.

More recently, image-based approaches have been explored [LMO*24; PBSJ24; MLTX19]. Wakey-Wakey [XZY*23] performs example-based motion transfer by optimizing control points of text vector outlines. However, such methods either neglect core principles of typography, legibility and stylistic consistency, or rely on heuristic regularization, which still cannot guarantee their preservation. In contrast, our work allows animating text directly

in the space of variable fonts, where legibility and stylistic coherence are preserved by construction. To drive motion, we employ variational methods for physics based animation.

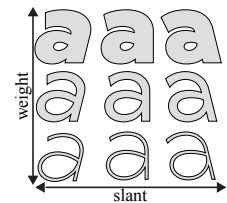
2.4. Font Matching

Font recognition is the task of identifying a typeface given an image of a text fragment. It has been studied extensively, with approaches ranging from nearest-class-mean classifiers with learned features [CYJ*14] to deep learning-based methods [WYJ*15a; WYJ*15b; WLTX18]. These methods aim to select the closest typeface from a large library of static fonts. In contrast, our objective is to recover the best-matching instance within a variable font, operating in a continuous parameter space rather than over discrete classes. To this end, we combine differentiable rasterization [LLMR20] with our differentiable variable font framework to directly optimize font variation axes against a simple image loss without requiring any learning.

3. Variable Font Mathematics

Variable Fonts, as defined by the OpenType specification, are a generalization of earlier efforts like Adobe’s Multiple Master and Apple’s GX fonts. Their ability to easily interpolate between designer-specified glyphs belies complex underlying interpolation modes expressed through a combination of pseudocode and descriptive text, distributed across several sections of the OpenType documentation. In this section we distill this implementation-skewed schema to a compact mathematical formulation of variable font interpolation. This mathematical foundation enables the development of a differentiable variable font framework, which can then be applied to support new modes of interaction and control.

A font file contains a collection of *glyphs*, the geometric descriptions of characters, along with other information such as layout and rendering instructions. In modern fonts glyph geometry is stored as vector outlines. We exactly convert these outlines (including quadratic curves and line segments) to sequences of cubic Bézier segments, so the ordered 2D control-point positions fully determine the shape. Variable fonts extend this idea by letting glyph outlines vary continuously along one or more design axes (known as *variable axes*). The inset shows variations of the character ‘a’ from the variable font Afacad Flux, obtained by adjusting its ‘weight’ (wght) and ‘slant’ axes.



Each variable axis i is defined over a numeric domain with a minimum, maximum, and default value: $s_i^{\min} \leq s_i \leq s_i^{\max}$. An instance of the variable font is specified by a vector $\mathbf{s} = (s_1, s_2, \dots, s_n) \in \mathbb{R}^n$, where s_i is chosen (by the user via axis sliders in a UI) within the axis’s allowed range. Internally, these axis values are normalized to a standard coordinate system via a piecewise-linear transformation, which maps $\{s_i^{\min}, s_i^0, s_i^{\max}\}$ to $\{-1, 0, 1\}$.

$$w_i(s_i) = \begin{cases} \frac{s_i - s_i^0}{s_i^0 - s_i^{\min}} & \text{if } s_i \leq s_i^0 \\ \frac{s_i - s_i^0}{s_i^{\max} - s_i^0} & \text{otherwise } (s_i > s_i^0), \end{cases}$$

where s_i^0 specifies the values of s_i which map to 0. To achieve perceptual uniformity along s , the font designer may specify a non-uniform piecewise-linear remapping. This remapping must still map s_i^{\min} , s_i^0 , s_i^{\max} to $\{-1, 0, 1\}$ (see Fig. 4). Throughout the remainder of the paper, we treat $\mathbf{w} \in [-1, 1]^n$ as the canonical representation of a font instance, used in all interpolation and optimization computations.

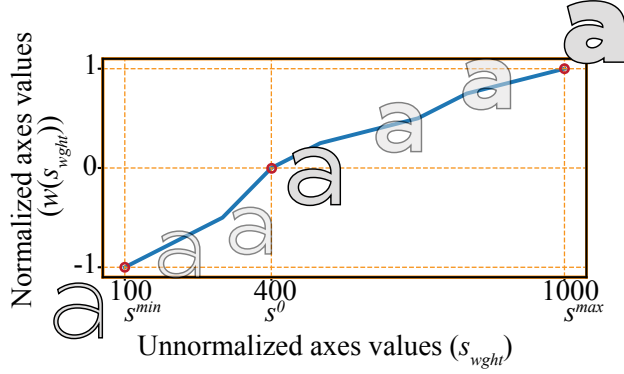


Figure 4: Mapping from user facing slider values to the normalized axes values

The font explicitly stores the control point locations for the default outline of each glyph, corresponding to $\mathbf{w} = \mathbf{0}$. We denote the vector of two-dimensional control points of glyph g in this default configuration as $\bar{\mathbf{p}}^g \in k \times 2$, where k is the total number of control points. To obtain glyph variations at $\mathbf{w} \neq \mathbf{0}$, the font provides *delta sets* (displacement vectors) that are appropriately scaled and added to $\bar{\mathbf{p}}^g$ (See Fig. 5(a)). We denote the delta set tensor for glyph g as $\Delta^g \in k \times 2 \times m$, where m is the number of delta sets, and $\Delta_{\dots, j}^g$ is the j -th delta set.

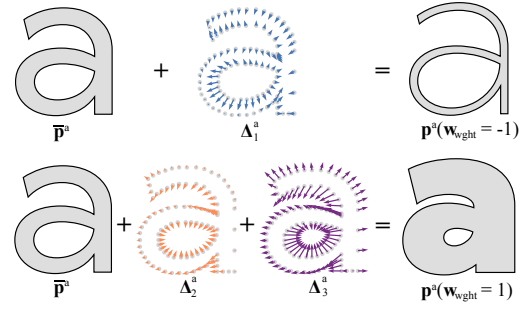
Each delta set is active only within a prescribed region of the variation space, and its influence is scaled by a weight function (*support function*). These weight functions are collected in a matrix $\Phi^{m \times n}$, where each row corresponds to a delta set and each column corresponds to a variation axis. For the j -th delta set, the weight function on axis i is piecewise linear, and is defined by three parameters: the start, peak, and end positions, given by the matrices $\mathbf{W}_{j,i}^{\text{start}}$, $\mathbf{W}_{j,i}^{\text{peak}}$, $\mathbf{W}_{j,i}^{\text{end}}$.

$$\phi_{j,i}(w_i) = \begin{cases} \frac{w_i - W_{j,i}^{\text{start}}}{W_{j,i}^{\text{peak}} - W_{j,i}^{\text{start}}} & \text{if } W_{j,i}^{\text{start}} \leq w_i < W_{j,i}^{\text{peak}} \\ \frac{W_{j,i}^{\text{stop}} - w_i}{W_{j,i}^{\text{stop}} - W_{j,i}^{\text{peak}}} & \text{if } W_{j,i}^{\text{peak}} \leq w_i < W_{j,i}^{\text{stop}} \\ 0 & \text{otherwise.} \end{cases}$$

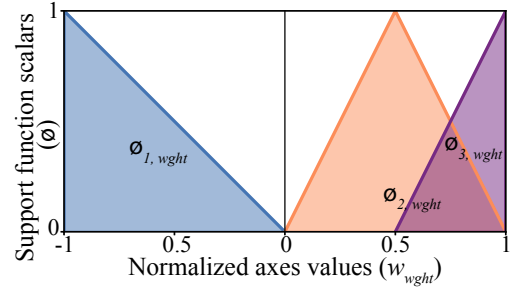
If no support function is specified for an axis, $\phi_{j,i}(w_i) = 1$ by default. See Fig. 5(b) for an illustration.

The contribution of a delta set is obtained by multiplying its weight functions across all active axes. The final glyph shape is then given by the nonlinear interpolation

$$\mathbf{p}^g(\mathbf{w}) = \bar{\mathbf{p}}^g + \Delta^g \cdot \gamma(\mathbf{w}) \quad (1)$$



(a)



(b)

Figure 5: Glyph variations are produced by adding delta sets to the default glyph $\bar{\mathbf{p}}$, with each delta set scaled by a nonlinear support function of \mathbf{w} . (a) Example variations of the character ‘a’ from the variable font *Afacad Flux* at $w_{\text{wght}} = -1$ and $w_{\text{wght}} = 0.999$ (other axes at default), showing the contributing delta sets. (b) One-dimensional view of the support functions ϕ along the weight axis, indicating the regions of influence of the delta sets in (a).

$$\gamma(\mathbf{w}) = \prod \phi(\mathbf{w}), \quad (2)$$

where \prod is a row-wise serial product and \cdot is tensor contraction with the third dimension of Δ^g . We illustrate this process in Fig. 6 and Fig. 7.

Layout Interpolation. In addition to glyph outlines, fonts specify layout quantities that control spacing and alignment in text composition. Layout is a complex process handled by text shaping engines, involving features such as kerning, ligatures, contextual alternates, and script-specific behaviors. Modeling the entire layout pipeline is beyond the scope of this work. For our purposes, we implement a minimal subset of layout logic sufficient to support word-level text composition. Specifically, we model the left and right side bearings. Side bearings define the space between a glyph’s outline and its bounding box, and together with neighboring glyphs determine inter-glyph spacing (See Fig. 8). These correspond to additional 2D points that define a glyph’s horizontal extent and its spacing relative to neighbors. The two points representing the LSB and RSB are interpolated using the same equation as the glyph’s control points (Eq. (1)). The final position of the k -th control point

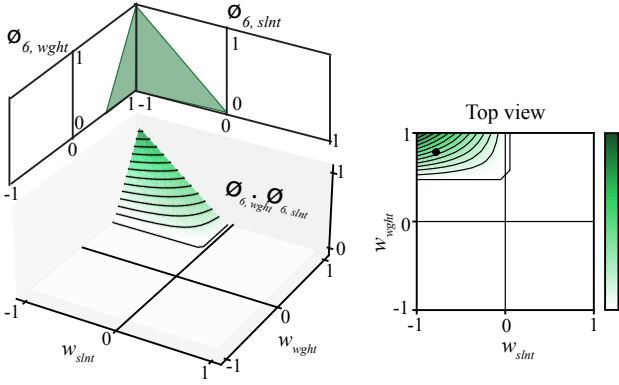


Figure 6: The scaling functions of delta sets are highly non-linear and non-invertible as they are defined as products of the support functions across all axes.

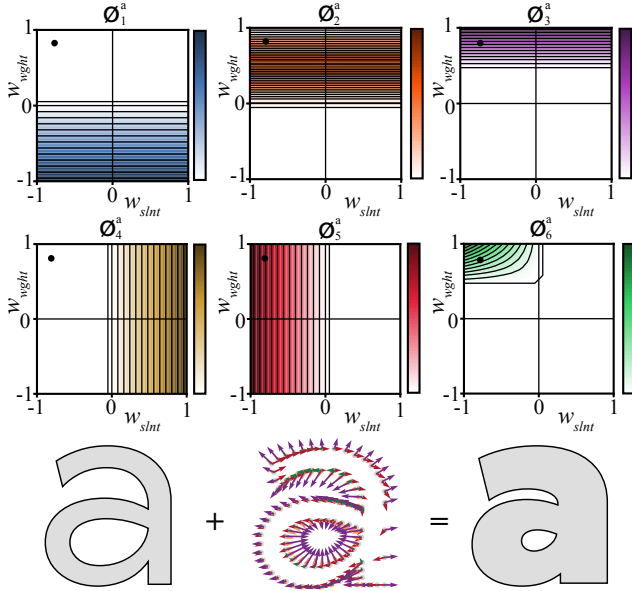


Figure 7: Illustration of delta sets and their influence on glyph variation. (Top) Scaling factors for six of the delta sets of the font *Afacad Flux*, plotted over the two variation axes, weight (w_{wght}) and slant (w_{slnt}). The black dot marks the chosen instance \mathbf{w} . (Bottom) The default glyph (left) is deformed by scaled delta displacements (middle) to produce the final glyph at the target instance (right).

of the j -th glyph in a word of length l is given by

$$\vec{\mathbf{p}}^l(\Theta) = \sum_{z=0}^{l-1} (\vec{\mathbf{p}}^z(\Theta_z) - \overleftarrow{\mathbf{p}}^z(\Theta_z)) - \overleftarrow{\mathbf{p}}^l(\Theta_l) + \mathbf{p}^l(\Theta_l) \quad (3)$$

where $\Theta = [\mathbf{w}^1 \mathbf{w}^2 \dots \mathbf{w}^l]$ are the parameters for each glyph in the

© 2026 The Author(s).

Computer Graphics Forum published by Eurographics and John Wiley & Sons Ltd.

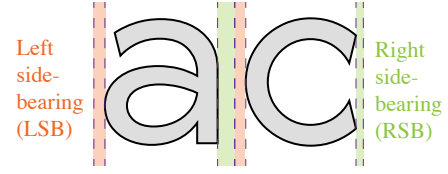


Figure 8: Role of Left and Right side bearings in laying out a word.

word and $\overleftarrow{\mathbf{p}}^g$, $\overrightarrow{\mathbf{p}}^g$ denotes the left and right sidebearing points of g respectively. Note that in this formula, we abuse notation slightly by assuming integer glyph position will index the correct glyph in the font.

4. Optimization of Variable Fonts

Building on the mathematical formulation in Section 3, we now describe a general approach for optimizing variable fonts using automatic differentiation.

We make the mapping from axis weights Θ to control positions \mathbf{p} in the interpolation pipeline (Eq. (3)) differentiable by implementing it in PyTorch.

A natural concern is that the mapping from axis weights Θ to control point positions \mathbf{p} is piecewise non-linear and may contain derivative discontinuities at region boundaries. However, these occur only on a measure-zero subset of the parameter space, and correspondingly we did not encounter any difficulties, in practice, across all examples presented in this paper.

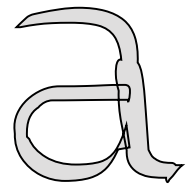
Our differentiable implementation enables continuous optimization of a wide variety of typography-related energy objectives directly over the variable axis weights. We formalize this as minimizing an energy function defined over axis weights:

$$\mathcal{E}(\Theta) = \|\mathcal{F}(\mathbf{p}(\Theta))\|^2 \quad (4)$$

Here, \mathcal{F} denotes a differentiable energy evaluated on the interpolated control points $\mathbf{p}(\Theta)$.

Choice of variable. The scaling factors of the delta sets ($\gamma(\Theta)$) are highly nonlinear functions due to the product of $\phi(\Theta)$ over all variable axes, and the piecewise-linear structure of $\phi(\Theta)$. It may be tempting to consider optimizing for the scaling factors directly, to bypass this nonlinearity. However, the m -dimensional space of scaling factors is not aligned with the design space of the font. Arbitrary configurations typically fall outside the axis-defined manifold and produce illegible glyphs (see inset). Moreover, projecting arbitrary scaling factors onto a consistent axis configuration is non-trivial. By contrast, optimizing directly in axis space guarantees that all interpolated instances remain within the design space prescribed by the font.

Axis limit constraints. The axis weights $\Theta \in [-1, 1]^{n-l}$ must remain within their valid bounds, since interpolation is undefined outside these limits. In practice, we clamp any out-of-range values before evaluating the interpolation to ensure it is well-defined. How-



ever, clamping alone creates regions with zero gradient. To address this, we project Θ back to the valid bounds after each update.

Optimization strategy. The optimization framework described above is compatible with a variety of standard gradient-based solvers. In our applications (Section 5), we employ Levenberg–Marquardt for energies defined over sampled outline points, and Adam for image-based energies defined on differentially rasterized glyphs.

Preprocessing. Realizing the variable font interpolation pipeline in practice requires substantial preprocessing. The data structures described in Sec. 3 are encoded in compressed lookup tables in the font file, often omitting values implicitly to reduce size—for example, through implied on-curve points, sparsely defined delta sets, default assumptions in region definitions, and inconsistent curve orientations. We preprocess these tables into uniform and unpacked representations that expose all geometry and layout data explicitly.

5. Applications

We now demonstrate the utility of our differentiable variable font optimization framework through several applications. These include interactive editing tools (Section 5.1, 5.2), physics driven kinetic typography (Section 5.3), and automatic font matching (Section 5.4). Each of these applications builds on the same underlying formulation but defines different energy objectives tailored to the task. We also report representative runtime statistics for these applications in Section 5.5.

5.1. Direct Manipulation

Current software support for variable fonts is limited to exposing font axes as sliders. This interface contrasts sharply with the vector-graphics workflows familiar to digital artists, where geometry is manipulated directly in 2D space. Although variable fonts are internally represented as vector graphics, the nonlinear mapping from axis weights to control point positions prevents such direct editing. Our differentiable formulation bridges this gap: by expressing editing operations as losses on control points and backpropagating them, we can optimize the axis weights to realize direct manipulation in the variable font domain.

To enable direct manipulation, we built an interactive system in which a user can select any point on the outline of a glyph and drag it to a new location. When a user clicks near the outline, the system identifies the closest curve segment and determines the corresponding Bézier parameter t . Once selected, the point can be dragged, with the system continuously updating the underlying axes weights to match the target position (See Fig. 9).

For direct manipulation, we formulate the energy as a least-squares objective that minimizes the difference between the dragged sample point on the outline and its target position. If $\mathbf{p}(\Theta, t)$ denotes the curve position at parameter t under axis weights Θ , and x^* is the user-specified target, the objective is

$$\mathcal{E}(\Theta) = \|\mathbf{p}(\Theta, t) - x^*\|^2 + \lambda \|\Theta - \Theta_{prev}\|^2,$$

where the second term regularizes the solution to remain close to the previous font instance. Additional editing constraints, such as

fixing points, enforcing shared x or y coordinates, or encouraging or maintaining collinearity, can be incorporated naturally as extra least-squares terms in the same formulation (See Fig. 10, video 2:35–2:57). We solve this optimization using the Levenberg–Marquardt algorithm.

User study. To assess whether direct manipulation could be useful for manipulating variable fonts, we conducted a pilot user study with six novice users (5 male, 1 female), all of whom were using variable fonts for the first time. Participants were asked to manipulate a glyph initialized with a variable font’s default parameters to match a target glyph shape sampled from the font’s design space. These tasks were performed using both variable font axis sliders and our direct manipulation interface, which allows users to select, drag, and fix points on the glyph outline.

As a preliminary usability assessment, participants completed the System Usability Scale (SUS) for both interfaces. The direct manipulation interface obtained a mean SUS score of 74.17, compared to 55.41 for the slider-based interface. In an additional comparison question, five out of six participants reported that the direct manipulation interface better captured their intent overall. Qualitative feedback suggested that direct manipulation was perceived as more intuitive and easier to learn, particularly for complex glyphs or fonts with many axes, while sliders were preferred for precise refinement and backtracking.

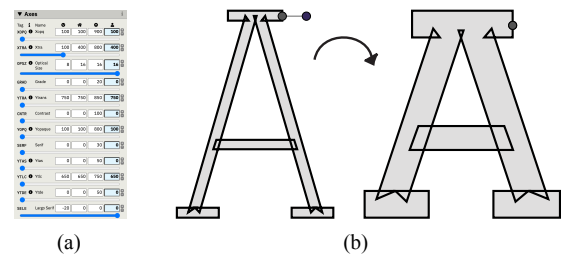


Figure 9: (a) The interface to variable fonts in existing systems exposes the variable font axes as sliders. (b) Our method enables manipulating the font instance by selecting a point on the text outline and dragging it. We optimize the variable font axes weights to match the user intent by leveraging our differentiable implementation. As opposed to typical slider-based workflow, our method enable a more direct and intuitive editing process.

5.2. Overlap Aware Modeling

When manipulating typography in graphic design, unintended overlaps can occur between glyphs or with background elements. A naive solution is to translate or resize the text, but this often compromises the overall design. Variable fonts provide additional degrees of freedom through their axis weights, but manually resolving collisions during iterative modeling can be tedious and time-consuming.

Our differentiable framework automates this process by updating the axis weights to resolve overlaps efficiently. (See Fig. 11). Collision detection is performed by densely sampling points along all glyph outlines and any background objects. For each colliding

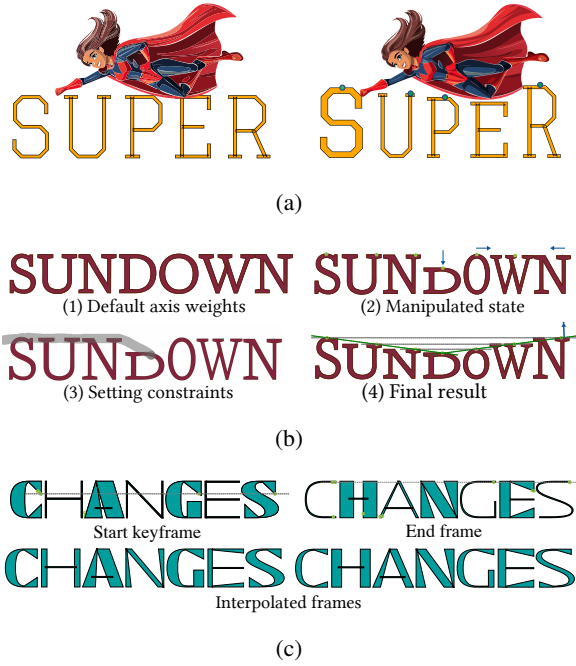


Figure 10: Examples of our 2D-control based font editing in various settings. (a) shows the result of a sequence of edits made by dragging multiple text outline samples to fit under a background object. (b) and (c) show result of constrained edits. In (b) outline samples are constrained to be collinear or to form a horizontal line for quickly creating the desired output. In (c) constrained edit is used to create keyframes for animation with consistent serif style.

point, we find the closest point on the other surface and its normal, defining a unilateral collision plane. Let \mathbf{t} contain the curve indices and Bézier parameters for all points that need to be projected, and let $\mathbf{p}(\Theta, \mathbf{t})$ give the corresponding stacked positions under the current axis weights. Similarly, let \mathbf{b} and \mathbf{N} be the stacked closest points and normals. We define the collision energy as a differentiable

$$\mathcal{E}_{\text{collision}}(\Theta) = \|\max(0, \mathbf{N}^T(\mathbf{p}(\Theta, \mathbf{t}) - \mathbf{b}))\|^2 \quad (5)$$

which penalizes points that penetrate other surfaces while leaving non-penetrating points unconstrained. We iteratively minimize this energy with Levenberg–Marquardt to find axis weights that resolve these collisions.

5.3. Physics Driven Kinetic Typography

Kinetic typography refers to animated text where motion is used to convey emphasis, mood, or narrative. It is widely used in graphic design, film, and advertising, but creating compelling motion for typography is difficult: designers must carefully balance visual dynamics with legibility and stylistic consistency. Variable fonts provide a natural design space for kinetic typography since their axes allow controlled shape variations without compromising aesthetics or readability.

With our differentiable framework, we can script physics-

© 2026 The Author(s).

Computer Graphics Forum published by Eurographics and John Wiley & Sons Ltd.

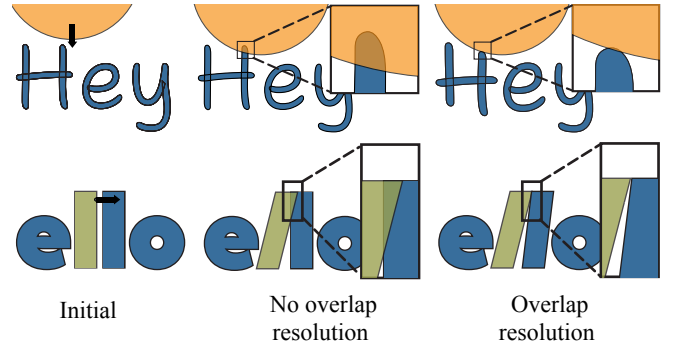


Figure 11: Text overlapping with background objects (top) or itself (bottom) is undesirable in many typography applications. Our method allows us to detect and resolve collisions in response to user edits. Editing the initial state (left) results in overlaps (center) which are avoided with collision response turned on (right).

inspired simulations directly in the space of variable font axes, enabling expressive animations without manually crafting trajectories frame by frame. We model momentum, elasticity, and contact in our simulations (Fig. 12):

- Elasticity is expressed as $\mathcal{E}_{\text{elastic}}(\Theta) = \|\Theta - \Theta_{\text{rest}}\|^2$, which encourages axes weights to remain close to a designated rest configuration.
- Momentum is handled by computing a momentum-predicted state Θ_m from the current velocity, and introducing a kinetic energy term $\mathcal{E}_{\text{kinetic}}(\Theta) = \|\Theta - \Theta_m\|^2$, which drives the solution toward that state.
- Collision handling follows the unilateral constraint formulation described in the Section 5.2, Eq. (5)

For time integration, we adopt an iterative constraint projection approach inspired by projective dynamics [BML*14]. Unlike standard projective dynamics, however, we cannot rely on a global linear solve since the collision constraint projection is non-linear. Instead, we iteratively minimize the combined elastic, kinetic, and collision energies using the Levenberg–Marquardt optimizer at each timestep.

5.4. Font Matching

A recurring task in graphic design and document editing is to reproduce the appearance of text found in an image. For example, a designer may wish to insert new text into a poster while matching the typography of existing content, or a digitization system may need to identify the closest font instance corresponding to scanned text. In all of these cases, the goal is not merely to recognize the font family, but to recover the precise appearance of the text in the observed image.

By combining our differentiable variable font approach with a differentiable SVG rasterizer, we can directly optimize image-based losses. Given a target image I_{target} and a rendering $I(\Theta)$ from

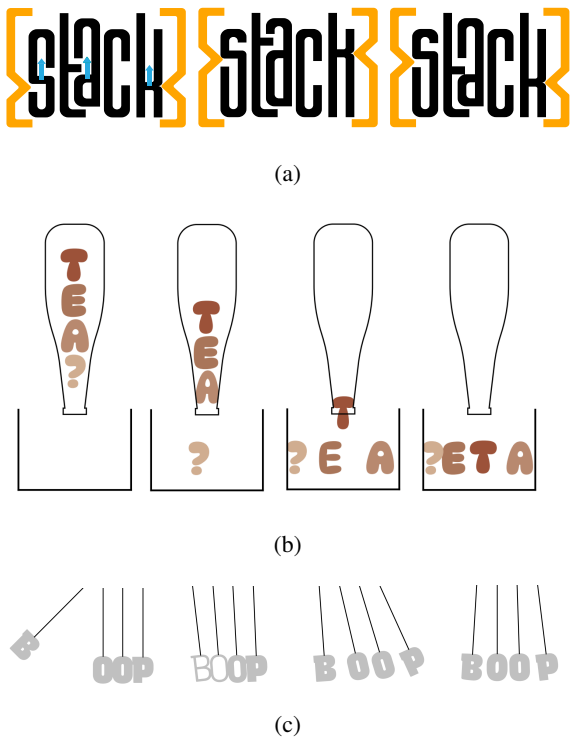


Figure 12: Examples of kinetic typography animations created with our approach, where text motion is driven by physics-based simulations.

axes weights Θ , we minimize

$$\mathcal{E}_{\text{image}}(\Theta) = \|I(\Theta) - I_{\text{target}}\|^2$$

using gradient descent. In practice, we use differentiable vector graphics rasterization [LLMR20] and employ an Adam optimizer (with 100 iterations), which reliably converges to visually accurate matches without additional tuning (Fig. 13). Since glyph geometry often contains overlapping elements, correct rasterization is obtained by specifying the nonzero fill rule in the differentiable SVG rasterizer, without requiring any special handling during optimization.

5.5. Performance

We report representative runtime statistics for our applications to provide a qualitative sense of performance. All experiments were run using unoptimized PyTorch code on a MacBook Pro (M1 Pro, CPU only).

The average time per update call for interactive direct manipulation, with overlap-aware modeling enabled, is 0.024 s for Fig. 1(F), 0.049 s for Fig. 1(NT), and 0.033 s for Fig. 11 (“ello”).

Physics-driven kinetic typography is evaluated as an offline simulation. Fig. 1(O) requires 5.27 s for 1000 timesteps, while Fig. 12(stack) takes 9.09 s for 350 timesteps.

Font matching is an offline optimization task dominated by differentiable rasterization. Per-iteration runtimes are 0.37 s at reso-

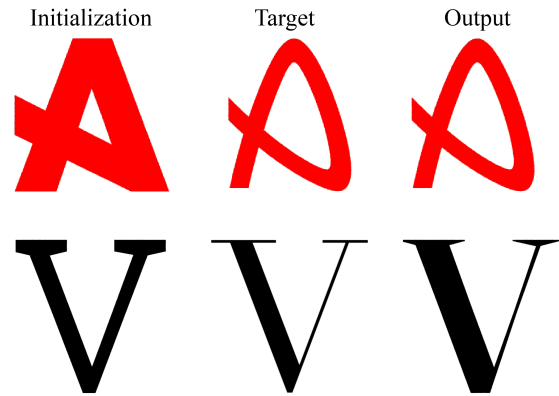


Figure 13: Font matching via differentiable optimization. (Top) A target glyph *A* from a variable font is reconstructed by optimizing the same font to recover the exact instance. (Bottom) A target glyph *V* from the non-variable font *Didot* is approximated using an unrelated variable font (*Amstelvar Roman*), demonstrating that our method can approximate the target even when the source and target fonts are unrelated.

lution 256×256 (Fig. 13(A)), 1.77 s at 576×512 (Fig. 13(V)), and 2.21 s at 640×640 (Fig. 1(S)).

6. Limitations and Future Work

While our framework introduces differentiability into variable font workflows, several important directions remain open.

Discrete changes. In addition to continuous variation along design axes, variable fonts can also encode discrete substitutions where the default glyph outline is replaced by a completely different geometry for an assigned part of the design space. Our current formulation does not capture these discontinuities, and extending differentiable models to handle such discrete structural changes would broaden the scope of applications.

Designing variable fonts. Creating variable fonts today is a labor-intensive process that requires designers to author detailed variation data for every glyph. A promising direction is to invert our framework, using differentiability to assist in generating these displacement sets automatically. Such tools could reduce the burden on type designers by extrapolating full families from a small set of manually designed exemplars.

Navigating variable fonts. Variable fonts differ in both the number and semantics of their axes, making it difficult for users to explore and select instances that suit their goals. Future work could investigate optimization-based navigation tools, helping users discover font instances that match stylistic, geometric, or functional criteria without exhaustive manual trial and error.

7. Conclusion

We introduce a compact mathematical formulation of variable fonts and a differentiable interpolation framework that exposes gradients

with respect to font parameters. This formulation enables a broad class of graphics tasks to be performed directly in variable font space via gradient-based optimization. Our approach opens up new uses in a wide variety of typography applications, spanning text editing, animation, and analysis. We believe this technique will catalyze new applications of variable fonts across design and animation, while also inspiring novel authoring tools that reduce the burden of designing them. To support this vision, we will release our implementation as open-source software, enabling both reproducibility and further exploration.

8. Acknowledgements

We thank Skef Iterum for feedback on our variable font formulation, Mengfei Liu for narrating the accompanying video, Zhecheng Wang, Chang Yue, Karran Pandey, Chenxi Liu, and Abhishek Madan for proofreading. Our research is funded in part by NSERC Discovery (RGPIN-2022-04680), the Ontario Early Research Award program, the Canada Research Chairs Program, a Sloan Research Fellowship, the DSI Catalyst Grant program and gifts by Adobe Inc.

References

- [Ado97] ADOBE. *Designing Multiple Master Typefaces*. Technical Note 5087. Adobe Systems, 1997 2.
- [AVR*22] ARAÚJO, CHRYSIANO, VINING, NICHOLAS, ROSALES, ENRIQUE, et al. “As-locally-uniform-as-possible reshaping of vector clip-art”. *ACM Transactions on Graphics (TOG)* 41.4 (2022), 1–10 3.
- [BL15] BERNSTEIN, GILBERT LOUIS and LI, WILMOT. “Lillicon: Using transient widgets to create scale variations of icons”. *ACM Transactions on Graphics (TOG)* 34.4 (2015), 1–11 3.
- [BML*14] BOUAZIZ, SOFIEN, MARTIN, SEBASTIAN, LIU, TIAN, et al. “Projective dynamics: fusing constraint projections for fast simulation”. *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: [10.1145/2601097.2601116](https://doi.org/10.1145/2601097.2601116). URL: <https://doi.org/10.1145/2601097.2601116>.
- [Bor04] BORZYSKOWSKI, GEORGE. “Animated text: More than meets the eye”. *Beyond the comfort zone: Proceedings of the 21st ASCILITE Conference*, Perth. 2004, 141–144 3.
- [CK14] CAMPBELL, NEILL DF and KAUTZ, JAN. “Learning a manifold of fonts”. *ACM Transactions on Graphics (ToG)* 33.4 (2014), 1–11 2.
- [CLDD09] CABRAL, MARCIO, LEFEBVRE, SYLVAIN, DACHSBACHER, CARSTEN, and DRETTAKIS, GEORGE. “Structure-Preserving Reshape for Textured Architectural Scenes”. *Computer Graphics Forum*. Vol. 28. 2. Wiley Online Library. 2009, 469–480 3.
- [CPSS10] CHAO, ISAAC, PINKALL, ULRICH, SANAN, PATRICK, and SCHRÖDER, PETER. “A simple geometric model for elastic deformations”. *ACM Trans. Graph.* 29.4 (July 2010). ISSN: 0730-0301. DOI: [10.1145/1778765.1778775](https://doi.org/10.1145/1778765.1778775). URL: <https://doi.org/10.1145/1778765.1778775>.
- [CYJ*14] CHEN, GUANG, YANG, JIANCHAO, JIN, HAILIN, et al. “Large-scale visual font recognition”. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, 3598–3605 3.
- [Gle92] GLEICHER, MICHAEL. “Briar: A constraint-based drawing program”. *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1992, 661–662 3.
- [HH01] HU, CHANGYUAN and HERSCH, ROGER D. “Parameterizable fonts based on shape components”. *IEEE Computer Graphics and Applications* 21.3 (2001), 70–85 2.
- [HLW93] HSU, SIU CHI, LEE, IRENE HH, and WISEMAN, NEIL E. “Skeletal strokes”. *Proceedings of the 6th annual ACM symposium on User interface software and technology*. 1993, 197–206 3.
- [IMH05] IGARASHI, TAKEO, MOSCOVICH, TOMER, and HUGHES, JOHN F. “As-rigid-as-possible shape manipulation”. *ACM transactions on Graphics (TOG)* 24.3 (2005), 1134–1141 2.
- [KM08] KIDAWARA, YUTAKA and MINAKUCHI, MITSURU. “Kinetic typography for ambient displays”. *Proceedings of the 2nd international conference on Ubiquitous information management and communication*. ACM. 2008 3.
- [Knu79] KNUTH, DONALD E. *METAFONT: a system for alphabet design*. Tech. rep. Stanford, CA, USA, 1979 2.
- [LFH02] LEE, JOHNNY C, FORLIZZI, JODI, and HUDSON, SCOTT E. “The kinetic typography engine: an extensible system for animating expressive text”. *Proceedings of the 15th annual ACM symposium on User interface software and technology*. 2002, 81–90 3.
- [LJG14] LIU, SONGRUN, JACOBSON, ALEC, and GINGOLD, YOTAM. “Skinning cubic Bézier splines and Catmull-Clark subdivision surfaces”. *ACM Transactions on Graphics (TOG)* 33.6 (2014), 1–9 3.
- [LLMR20] LI, TZU-MAO, LUKÁČ, MICHAL, MICHAËL, GHARBI, and RAGAN-KELLEY, JONATHAN. “Differentiable Vector Graphics Rasterization for Editing and Learning”. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39.6 (2020), 193:1–193:15 3, 8.
- [LMO*24] LIU, ZICHEN, MENG, YIHAO, OUYANG, HAO, et al. “Dynamic typography: Bringing text to life via video diffusion prior”. *arXiv preprint arXiv:2404.11614* (2024) 3.
- [Mic18] MICROSOFT. *OpenType Font Variations Overview*. Microsoft. 2018. URL: <https://learn.microsoft.com/en-us/typography/opentype/spec/otvaroverview> 2.
- [MLTX19] MEN, YIFANG, LIAN, ZHOUHUI, TANG, YINGMIN, and XIAO, JIANGUO. “Dyntypo: Example-based dynamic text effects transfer”. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, 5870–5879 3.
- [PBSJ24] PARK, SEONMI, BAE, INHWAN, SHIN, SEUNGHYUN, and JEON, HAE-GON. “Kinetic typography diffusion model”. *European Conference on Computer Vision*. Springer. 2024, 166–185 3.
- [SA07] SORKINE, OLGA and ALEXA, MARC. “As-rigid-as-possible surface modeling”. *Symposium on Geometry processing*. Vol. 4. Citeseer. 2007, 109–116 2.
- [SBBG11] SOLOMON, JUSTIN, BEN-CHEN, MIRELA, BUTSCHER, ADRIAN, and GUIBAS, LEONIDAS. “As-killing-as-possible vector fields for planar deformation”. *Computer Graphics Forum*. Vol. 30. 5. Wiley Online Library. 2011, 1543–1552 2.
- [SR98] SHAMIR, ARIEL and RAPPOPORT, ARI. “Feature-based design of fonts using constraints”. *International Conference on Raster Imaging and Digital Typography*. Springer. 1998, 93–108 2.
- [Sut64] SUTHERLAND, IVAN E. “Sketch pad a man-machine graphical communication system”. *Proceedings of the SHARE design automation workshop*. 1964, 6–329 3.
- [WG10] WEBER, OFIR and GOTSMAN, CRAIG. “Controllable conformal maps for shape deformation and interpolation”. *ACM SIGGRAPH 2010 papers*. 2010, 1–11 2.
- [WLTx18] WANG, YIZHI, LIAN, ZHOUHUI, TANG, YINGMIN, and XIAO, JIANGUO. “Font Recognition in Natural Images via Transfer Learning”. *MultiMedia Modeling*. Ed. by SCHOEFFMANN, KLAUS, CHALIDABHONGSE, THANARAT H., NGO, CHONG WAH, et al. Cham: Springer International Publishing, 2018, 229–240. ISBN: 978-3-319-73603-7 3.
- [WXW*06] WENG, YANLIN, XU, WEIWEI, WU, YANCHEN, et al. “2D shape deformation using nonlinear least squares optimization”. *The visual computer* 22 (2006), 653–660 2.
- [WYJ*15a] WANG, ZHANGYANG, YANG, JIANCHAO, JIN, HAILIN, et al. “Deepfont: Identify your font from an image”. *Proceedings of the 23rd ACM international conference on Multimedia*. 2015, 451–459 3.

- [WYJ*15b] WANG, ZHANGYANG, YANG, JIANCHAO, JIN, HAILIN, et al. “Real-world font recognition using deep network and domain adaptation”. *arXiv preprint arXiv:1504.00028* (2015) 3.
- [XZY*23] XIE, LIWENHAN, ZHOU, ZHAOYU, YU, KERUN, et al. “Wakey-wakey: Animate text by mimicking characters in a gif”. *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 2023, 1–14 3.