Iluz, Shir; Vinker, Yael; Hertz, Amir; Berio, Daniel; Cohen-Or, Daniel and Shamir, Ariel. 2023. Word-As-Image for Semantic Typography. ACM Transactions on Graphics, 42(4), 151. ISSN 0730-0301 [Article]

https://research.gold.ac.uk/id/eprint/34053/

The version presented here may differ from the published, performed or presented work. Please go to the persistent GRO record above for more information.

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Goldsmiths, University of London via the following email address: gro@gold.ac.uk.

The item will be removed from the repository while any claim is being investigated. For more information, please contact the GRO team: gro@gold.ac.uk
Word-As-Image for Semantic Typography

Shir Iluz*
Tel-Aviv University, Israel

Yael Vinker*
Tel-Aviv University, Israel

Amir Hertz
Tel-Aviv University, Israel

Daniel Berio
Goldsmiths University, London

Daniel Cohen-Or
Tel-Aviv University, Israel

Ariel Shamir
Reichman University, Israel

Fig. 1. A few examples of our word-as-image illustrations in various fonts and for different textual concept. The semantically adjusted letters are created completely automatically using our method, and can then be used for further creative design as we illustrate here.

A word-as-image is a semantic typography technique where a word illustration presents a visualization of the meaning of the word, while also preserving its readability. We present a method to create word-as-image illustrations automatically. This task is highly challenging as it requires semantic understanding of the word and a creative idea of where and how to depict these semantics in a visually pleasing and legible manner. We rely on the remarkable ability of recent large pretrained language-vision models to distill textual concepts visually. We target simple, concise, black-and-white designs that convey the semantics clearly. We deliberately do not change the color or texture of the letters and do not use embellishments. Our method optimizes the outline of each letter to convey the desired concept, guided by a pretrained Stable Diffusion model. We incorporate additional loss terms to ensure the legibility of the text and the preservation of the style of the font. We show high quality and engaging results on numerous examples and compare to alternative techniques.

Code will be available at our project page.

1 INTRODUCTION

Semantic typography is the practice of using typography to visually reinforce the meaning of text. This can be achieved through the choice of typefaces, font sizes, font styles, and other typographic elements. A more elaborate and engaging technique for semantic typography is presented by word-as-image illustrations, where the semantics of a given word are illustrated using only the graphical elements of its letters. Such illustrations provide a visual representation of the meaning of the word, while also preserving the readability of the word as a whole.

The task of creating a word-as-image is highly challenging, as it requires the ability to understand and depict the visual characteristics of the given concept, and to convey them in a concise, aesthetic, and comprehensible manner without harming legibility. It requires a great deal of creativity and design skills to integrate the chosen visual concept into the letter’s shape [Lee 2011]. In Figure 2 we show some word-as-image examples created manually. For example, to create the “jazz” depiction, the designer had to first choose the visual concept that would best fit the semantics of the text (a saxophone), consider the desired font characteristics, and then choose the most suitable letter to be replaced. Finding the right visual element to illustrate a concept is ill-defined as there are countless ways to illustrate any given concept. In addition, one cannot simply copy a selected visual element onto the word – there is a need to find subtle modifications of the letters shape.

Because of these complexities, the task of automatic creation of word-as-image illustrations was practically impossible to achieve using computers until recently. In this paper, we define an algorithm for automatic creation of word-as-image illustrations based on recent advances in deep-learning and the availability of huge foundational models that combine language and visual understanding. Our resulting illustrations (see Figure 1) could be used for logo design, for signs, in greeting cards and invitations, and simply for fun. They can be used as-is, or as inspiration for further refinement of the design.

Existing methods in the field of text stylization often rely on raster textures [Yang et al. 2018], place a manually created style on top of the strokes segmentation [Berio et al. 2022], or deform the text into a pre-defined target shape [Zou et al. 2016] (see Figure 3). Only a few works [Tendulkar et al. 2019; Zhang et al. 2017] deal with semantic typography, and they often operate in the raster domain and use existing icons for replacement (see Figure 3E).

Our word-as-image illustrations concentrate on changing only the geometry of the letters to convey the meaning. We deliberately do not change color or texture and do not use embellishments. This

* Denotes equal contribution.
allows simple, concise, black-and-white designs that convey the semantics clearly. In addition, since we preserve the vector-based representation of the letters, this allows smooth rasterization in any size, as well as applying additional style manipulations to the illustration using colors and texture, if desired.

Given an input word, our method is applied separately for each letter, allowing the user to later choose the most likeable combination for replacement. We represent each letter as a closed vectorized shape, and optimize its parameters to reflect the meaning of the word, while still preserving its original style and design.

We rely on the prior of a pretrained Stable Diffusion model [Rombach et al. 2021] to connect between text and images, and utilize the Score Distillation Sampling approach [Poole et al. 2022] (see Section 3) to encourage the appearance of the letter to reflect the provided textual concept. Since the Stable Diffusion model is trained on raster images, we use a differentiable rasterizer [Li et al. 2020] that allows to backpropagate gradients from a raster-based loss to the shape’s parameters.

To preserve the shape of the original letter and ensure legibility of the word, we utilize two additional loss functions. The first loss regulates the shape modification by constraining the deformation to be as-conformal-as-possible over a triangulation of the letter’s shape. The second loss preserves the local tone and structure of the letter by comparing the low-pass filter of the resulting rasterized shape. The second loss preserves the local tone and structure of the letter by comparing the low-pass filter of the resulting rasterized shape to the original one.

We compare to several baselines, and present many results using various typefaces and a large number of concepts. Our word-as-image illustrations convey the intended concept while maintaining legibility and preserving the appearance of the font, demonstrating visual creativity.

2 RELATED WORK

Text Stylization. One approach to text stylization is artistic text style transfer, where the style from a given source image is migrated into the desired text (such as in Figure 3A). To tackle this task, existing works incorporate patch-based texture synthesis [Fish et al. 2020; Yang et al. 2017] as well as variants of GANs [Azadi et al. 2018; Jiang et al. 2019; Mao et al. 2022; Wang et al. 2019; Yang et al. 2022]. These works operate within the raster domain, a format that is undesirable for typographers since fonts must be scalable. In contrast, we operate on the parametric outlines of the letters, and our glyph manipulation is guided by the semantic meaning of the word, rather than a pre-defined style image.

A number of works [Ha and Eck 2018; Lopes et al. 2019; Wang and Lian 2021] tackle the task of font generation and stylization in the vector domain. Commonly, a latent feature space of font’s outlines is constructed, represented as outline samples [Balashova et al. 2019; Campbell and Kautz 2014] or parametric curve segments [Ha and Eck 2018; Lopes et al. 2019; Wang and Lian 2021]. These approaches are often limited to mild deviations from the input data. Other methods rely on templates [Lian et al. 2018; Suveranont and Igarashi 2010] or on user guided [Phan et al. 2015] and automatic [Berio et al. 2022] stroke segmentation to produce letter stylization (such as in Figure 3B). However, they rely on a manually defined style, while we rely on the expressiveness of Stable Diffusion to guide the modification of the letters’ shape, to convey the meaning of the provided word. In the task of calligram generation [Xu and Kaplan 2007; Zou et al. 2016] the entire word is deformed into a given target shape. This task prioritises shape over the readability of the word (see Figure 3D), and is inherently different from ours, as we use the semantics of the word to derive the deformation of individual letters.

Most related to our goal, are works that perform semantic stylization of text. Tendulkar et al. [2019] replace letters in a given word with clip-art icons describing a given theme (see Figure 3E). To choose the most suitable icon for replacement, an autoencoder is used to measure the distance between the letter and icons from the desired class. Similarly, Zhang et al. [2017] replace stroke-like parts of one or more letters with instances of clip art to generate ornamental stylizations. An example is shown in Figure 3C. These approaches operate in the raster domain, and replace letters with existing icons, which limits them to a predefined set of classes present in the dataset. Our method, however, operates in the vector domain, and incorporates the expressiveness of large pretrained image-language models to create a new illustration that conveys the desired concept.

Large Language-Vision Models. With the recent advancement of language-vision models [Radford et al. 2021] and diffusion models [Nichol et al. 2021; Ramesh et al. 2022; Rombach et al. 2021], the field of image generation and editing has undergone unprecedented evolution. Having been trained on millions of images and text pairs, these models have proven effective for performing challenging vision related tasks such as image segmentation [Amit et al. 2021], domain adaptation [Song et al. 2022], image editing [Avrahami et al. 2022; Hertz et al. 2022; Tumanyan et al. 2022a], personalization [Gal et al. 2019], and many other tasks.
et al. 2022, 2023; Ruiz et al. 2022], and explainability [Chefer et al. 2021]. Despite being trained on raster images, their strong visual and semantic priors have also been shown to be successfully applied to other domains, such as motion [Tevet et al. 2022], meshes [Michel et al. 2021], point cloud [Zhang et al. 2021], and vector graphics. CLIPDraw [Frans et al. 2021] uses a differentiable rasterizer [Li et al. 2020] to optimize a set of colorful curves w.r.t. a given text prompt, guided by CLIP’s image-text similarity metric. Tian and Ha [2021] use evolutionary algorithms combined with CLIP guidance to create abstract visual concepts based on text. Other works [Vinker et al. 2022a,b] utilize the image encoder of CLIP to generate abstract vector sketches from images.

Diffusion models have been used for the task of text guided image-to-image translation [Choi et al. 2021; Tumanyan et al. 2022b]. In SDEdit [Meng et al. 2022], an adequate amount of noise is added to a reference image, such that its overall structure is preserved, and then the image is denoised in a reverse process with a guiding text. Pretrained diffusion models have also been used to generate 3D objects [Metzer et al. 2022; Poole et al. 2022], or vector art [Jain et al. 2022] conditioned on text.

In our work, we also utilize the strong visual and semantic prior induced by a pretrained Stable Diffusion model [Rombach et al. 2021], however, for the task of semantic typography. For that purpose we add new components to the optimization process to preserve the font’s style and text legibility.

3 BACKGROUND

3.1 Fonts and Vector Representation

Modern typeface formats such as TrueType [Penney 1996] and PostScript [Inc. 1990] represent glyphs using a vectorized graphic representation of their outlines. Specifically, the outline contours are typically represented by a collection of lines and Bézier or B-Spline curves. This representation allows to scale the letters and rasterize them in any desired size similar to other vector representations. This property is preserved by our method as our output preserves the vectorized representations of the letters.

3.2 Latent Diffusion Models

Diffusion models are generative models that are trained to learn a data distribution by the gradual denoising of a variable sampled from a Gaussian distribution.

In our work, we use the publicly available text-to-image Stable Diffusion model [Rombach et al. 2021]. Stable Diffusion is a type of a latent diffusion model (LDM), where the diffusion process is done over the latent space of a pretrained model autoencoder. The encoder $E$ is tasked with mapping an input image $x$ into a latent vector $z$, and the decoder $D$ is trained to decode $z$ such that $D(z) \approx x$.

As a second stage, a denoising diffusion probabilistic model (DDPM) [Ho et al. 2020] is trained to generate codes within the learned latent space. At each step during training, a scalar $t \in \{1, 2, \ldots, T\}$ is uniformly sampled and used to define a noised latent code $z_t = \alpha_t z + \sigma_t \epsilon$, where $\epsilon \sim N(0, I)$ and $\alpha_t, \sigma_t$ are terms that control the noise schedule, and are functions of the diffusion process time $t$.

The denoising network $\theta_\| \epsilon_t \|$ which is based on a UNet architecture [Ronneberger et al. 2015], receives as input the noised code $z_t$, the timestep $t$ and an optional condition vector $c(y)$, and is tasked with predicting the added noise $\epsilon$. The LDM loss is defined by:

$$L_{LDM} = \mathbb{E}_{z \sim \varepsilon(E(x), y) \sim N(0, 1), t} \left[ \| \varepsilon - \theta_\| (z_t, t, c(y)) \|^2 \right].$$

In Stable Diffusion, for text-to-image generation, the condition vector is the text embedding produced by a pre-trained CLIP text encoder [Radford et al. 2021]. At inference time, a random latent code $z_T \sim N(0, I)$ is sampled, and iteratively denoised by the trained $\theta_\| \epsilon_t \|$ until producing a clean $z_0$ latent code, which is passed through the decoder $D$ to produce the image $x$.

3.3 Score Distillation

It is desirable to utilize the strong prior of pretrained large text-image models for the generation of modalities beyond rasterized images. In Stable Diffusion, text conditioning is performed via the cross-attention layers defined at different resolutions in the UNet network. Thus, it is not trivial to guide an optimization process using the conditioned diffusion model.

DreamFusion [Poole et al. 2022] proposed a way to use the diffusion loss to optimize the parameters of a NeRF model for text-to-3D generation. At each iteration, the radiance field is rendered from a random angle, forming the image $x$, which is then noised to form $x_t = \alpha_t x + \sigma_t \epsilon$. The noised image is then passed to the pretrained UNet model of Imagen [Saharia et al. 2022], that outputs the prediction of the noise $\epsilon$. The score distillation loss is defined by the gradients of the original diffusion loss:

$$\nabla_{\phi} L_{SDS} = \left[ w(t)(\epsilon_t(x_t, t, y) - \epsilon) \frac{\partial x}{\partial \phi} \right] \text{.} \tag{2}$$

where $y$ is the condition text prompt, $\phi$ are the NeRF’s parameters and $w(t)$ is a constant multiplier that depends on $\alpha_t$. During training, the gradients are back-propagated to the NeRF parameters to gradually change the 3D object to fit the text prompt. Note that the gradients of the UNet are skipped, and the gradients to modify the NeRF’s parameters are derived directly from the LDM loss.
We begin by defining the parametric representation of the letters $z = \mathbf{visual depiction of the letter. The user can then choose which letters}$ for differentiable rasterization. $\mathbf{different fonts and letters, and to facilitate the use of diffvg [Li et al.}$ for the input letter as $\mathbf{method is applied to every letter}$ of cubic Bézier curves, to have a consistent representation across $\mathbf{method is useful for our task of producing semantic glyphs, and we follow the technical steps proposed in Vec}$-

**3.4 VectorFusion**

Recently, VectorFusion [Jain et al. 2022] utilized the SDS loss for the task of text-to-SVG generation. The proposed generation pipeline involves two stages. Given a text prompt, first, an image is generated using Stable Diffusion (with an added suffix to the prompt), and is then vectorized automatically using LIVE [Ma et al. 2022]. This defines an initial set of parameters to be optimized in the second stage using the SDS loss. At each iteration, a differentiable rasterizer [Li et al. 2020] is used to produce a $600 \times 600$ image, which is then augmented as suggested in CLIPDraw [Frans et al. 2021] to get a $512 \times 512$ image $x_{aug}$. Then $x_{aug}$ is fed into the pretrained encoder $\mathcal{E}$ of Stable Diffusion to produce the corresponding latent code $z = \mathcal{E}(x_{aug})$. The SDS loss is then applied in this latent space, in a similar way to the one defined in DreamFusion:

$$\nabla_{\theta} L_{\text{SDS}} = \mathbb{E}_{t,x} \left[ w(t) \left( \frac{\partial \mathcal{E}_\theta(x_t + \sigma t e, y)}{\partial x_{aug}} - \epsilon \right) \frac{\partial z}{\partial x_{aug}} \frac{\partial x_{aug}}{\partial \theta} \right]$$

We find the SDS approach useful for our task of producing semantic glyphs, and we follow the technical steps proposed in VectorFusion (e.g. augmentations and the added suffix).

**4 METHOD**

Given a word $W$ represented as a string with $n$ letters ($l_1, \ldots, l_n$), our method is applied to every letter $l_i$ separately to produce a semantic visual depiction of the letter. The user can then choose which letters to replace and which to keep in their original form.

**4.1 Letter Representation**

We begin by defining the parametric representation of the letters in $W$. We use the FreeType font library [FreeType 2009] to extract the outline of each letter. We then translate each outline into a set of cubic Bézier curves, to have a consistent representation across different fonts and letters, and to facilitate the use of diffvg [Li et al. 2020] for differentiable rasterization.

**4.2 Optimization**

The pipeline of our method is provided in Figure 5. Since we are optimizing each letter $l_i$ separately, for brevity, we will omit the letter index $i$ in the following text and define the set of control points for the input letter as $P$.

Given $P$ and the desired textual concept $c$ (both marked in purple in Figure 5), our goal is to produce a new set of control points, $\hat{P}$, depending on the letter’s complexity and the style of the font, the extracted outlines are defined by a different number of control points. We have found that the initial number of control points affects the final appearance significantly: as the number of control points increases, there is more freedom for visual changes to occur. Therefore, we additionally apply a subdivision procedure to letters containing a small number of control points. We define a desired number of control points for each letter of the alphabet (shared across different fonts), and then iteratively subdivide the Bézier segments until reaching this target number. At each iteration, we compute the maximum arc length among all Bézier curve segment endpoints. The blue dots are the initial Bézier curve segment endpoints. The orange dots are the remaining control points respectively before and after subdivision.

![Fig. 6. Illustration of the letter’s outline and control points before (left) and after (right) the subdivision process. The orange dots are the initial Bézier curve segment endpoints. The blue dots are the remaining control points respectively before and after subdivision.](image)
defining an adjusted letter \( \hat{l} \) that conveys the given concept, while maintaining the overall structure and characteristics of the initial letter \( l \).

We initialize the learned set of control points \( \hat{P} \) with \( P \), and pass it through a differentiable rasterizer \( R \) [Li et al. 2020] (marked in blue), which outputs the rasterized letter \( R(\hat{P}) \). The rasterized letter is then randomly augmented and passed into a pretrained Stable Diffusion [Rombach et al. 2021] model, conditioned on the CLIP’s embedding of the given text \( c \). The SDS loss \( \nabla_\theta L_{SDS} \) is then used as described in Section 3 to encourage \( R(\hat{P}) \) to convey the given text prompt.

To preserve the shape of each individual letter and ensure the legibility of the word as a whole, we use two additional loss functions to guide the optimization process. The first loss limits the overall shape change by defining as-conformal-as-possible constraint on the shape deformation. The second loss preserves the overall shape and style of the font by constraining the tone (i.e. amount of dark vs. light areas in local parts of the shape) of the modified letter not to diverge too much from the original letter (see Section 4.3).

The gradients obtained from all the losses are then backpropagated, to update the parameters \( \hat{P} \). We repeat this process for 500 steps, which takes ~5 minutes to produce a single letter illustration on RTX2080 GPU.

4.3 Loss Functions

Our primary objective of encouraging the resulting shape to convey the intended semantic concept, is utilized by \( \nabla_\theta L_{SDS} \) loss (described in Section 3). We observe that using \( \nabla_\theta L_{SDS} \) solely can cause large deviations from the initial letter appearance, which is undesired. Hence, our additional goal is to maintain the shape and legibility of the letter \( R(\hat{P}) \), as well as to keep the original font’s characteristics. For that purpose we use two additional losses.

**As-Conformal-As-Possible Deformation Loss.** To prevent the final letter shape from diverging too much from the initial shape, we triangulate the inner part of the letter and constrain the deformation of the letter to be as conformal as possible (ACAP) [Hormann and Greiner 2000]. We use constrained Delaunay triangulation [Barber and Huhdanpaa 1995; Deaunay et al. 1994] on the set of control points defining the glyph. It is known that Delaunay triangulation can be used to produce the skeleton of an outline [Prasad 1997; Zou et al. 2001], so the ACAP loss also implicitly captures a skeletal representation of the letter form.

\[ L_{acap}(P, \hat{P}) = \frac{1}{k} \sum_{j=1}^{k} \sum_{i=1}^{m_j} (\alpha_j^i - \hat{\alpha}_j^i)^2 \]  

where \( k = |P| \) and \( \hat{\alpha} \) are the angles induced by \( D(\hat{P}) \).

**Tone Preservation Loss.** To preserve the style of the font as well as the structure of the letter we add a local-tone preservation loss term. This term constrains the tone (amount of black vs. white in all regions of the shape) of the adjusted letter not to deviate too much from tone of the original font’s letter. Towards this end, we apply a low pass filter (LPF) to the rasterized letter (before and after deformation) and compute the L2 distance between the resulting blurred letters:

\[ L_{tone}(\hat{P}, \hat{P}) = \frac{1}{k} \sum_{j=1}^{k} \left\| \text{LPF}(R(\hat{P})) - \text{LPF}(R(\hat{P})) \right\|_2^2 \]

An example of the blurred letters is shown in Figure 8, as can be seen, we use a high value of standard deviation \( \sigma \) in the blurring kernel to blur out small details such as the ears of bear.

Our final objective is then defined by the weighted average of the three terms:

\[ \min_{\hat{P}} \nabla_\theta L_{SDS}(R(\hat{P}), c) + \alpha \cdot L_{acap}(P, \hat{P}) + \beta \cdot L_{tone}(R(\hat{P}), R(\hat{P})) \]

where \( \alpha = 0.5 \) and \( \beta \) depends on the step \( t \) as described next.

4.4 Weighting

Choosing the relative weights of the three losses presented above is crucial to the appearance of the final letter. While the \( \nabla_\theta L_{SDS} \) loss encourages the shape to deviate from its original appearance to better fit the semantic concept, the two terms \( L_{tone} \) and \( L_{acap} \) are responsible for maintaining the original shape. Hence, we have two competing parts in the formula, and would like to find a balance between them to maintain the legibility of the letter while allowing the desired semantic shape to change.

We find that \( L_{tone} \) can be very dominant. In some cases, if it is used from the beginning, no semantic deformation is performed.
Therefore, we adjust the weight of $L_{\text{tone}}$ to kick-in only after some semantic deformation has occurred. We define $\beta_t$ as follows:

$$\beta_t = a \cdot \exp \left( - \frac{(t - b)^2}{2c^2} \right)$$

with $a = 100, b = 300, c = 30$. We analyse the affect of various weighting in Section 5.3.

5 RESULTS

The robustness of our approach means it should be capable of handling a wide range of input concepts as well as supporting different font designs. Figures 1, 4, 33, 17, and more results in the supplemental material demonstrate that our approach can handle inputs from many different categories and various fonts, and that the generated results are legible and creative. Figure 9 demonstrate how the illustrations created by our method for the same word follow the characteristics of different fonts. Although the perceived aesthetics of a word-as-image illustration can be subjective, we define three objectives for an effective result: (1) it should visually capture the given semantic concept, (2) it should maintain readability, and (3) it should preserve the original font’s characteristics.

We evaluate the performance of our method on a randomly selected set of inputs. We select five common concept classes - animals, fruits, plants, sports, and professions. Using ChatGPT, we sample ten random instances for each class, resulting in 50 words in total. Next, we select four fonts that have distinct visual characteristics, namely Quicksand, Bell MT, Noteworthy-Bold, and HobeauxRoccoceaux-Sherman. For each word, we randomly sampled one of the four fonts, and applied our method to each letter. For each word with $n$ letters we can generate $2^n$ possible word-as-images, which are all possible combinations of replacements of illustrated letters. A selected subset of these results is presented in Figure 33. The results of all letters and words are presented in the supplementary material.

As can be seen, the resulting word-as-image illustrations successfully convey the given semantic concept in most cases while still remaining legible. In addition, our method successfully captures the font characteristics. For example, in Figure 33, the replacements for the “DRESS” and “LION” are thin and fit well with the rest of the word. In addition, observe the serifs of the letter A used for the fin of the shark in the “SHARK” example. We further use human evaluation to validate this as described below.

5.1 Quantitative

We conduct a perceptual study to quantitatively assess the three objectives of our resulting word-as-images. We randomly select two instances from each of the resulting word-as-image illustrations for the five classes described above, and visually select one letter from each word, resulting in 10 letters in total. In each question we show an isolated letter illustration, without the context of the word. To evaluate the ability of our method to visually depict the desired concept, we present four label options from the same class, and ask participants to choose the one that describes the letter illustration best. To evaluate the legibility of the results, we ask participants to choose the most suitable letter from a random list of four letters. To assess the preservation of the font style, we present the four fonts and ask participants to choose the most suitable font for the illustration. We gathered answers from 40 participants, and the results are shown in Table 1. As can be seen, the level of concept recognizability and letter legibility are very high, and the 51% of style matching of the letter illustration to the original font is well above random, which is 25%. We also test our algorithm without the two additional structure and style preserving losses ($L_{\text{cap}}$ and $L_{\text{tone}}$) on the same words and letters (“Only SDS” in the table).

<table>
<thead>
<tr>
<th>Method</th>
<th>Semantics</th>
<th>Legibility</th>
<th>Font</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>0.8</td>
<td>0.9</td>
<td>0.51</td>
</tr>
<tr>
<td>Only SDS</td>
<td>0.88</td>
<td>0.53</td>
<td>0.33</td>
</tr>
</tbody>
</table>

Table 1. Perceptual study results. The level of concept recognizability and letter legibility are very high, and style matching of the font is well above random. The “Only SDS” results are created by removing our structure and style preserving losses.

In the absence of a relevant baseline for comparison, we define baselines based on large popular text-to-image models. Specifically, we use (1) SD Stable Diffusion [Rombach et al. 2021], (2) SDEdit [Meng et al. 2022], (3) DallE2 [Ramesh et al. 2022] illustrating the word, (4) DallE2+letter illustrating only the letter, and (5) CLIPDraw [Frans et al. 2021]. We applied the methods above (details can be found in supplemental material) to three representative words – “bird”, “dress”, and “tulip”, with the fonts Bell MT, Quicksand, and Noteworthy-Bold, respectively. The results can be seen in Figure 10.

In some cases Stable Diffusion (SD) did not manage to produce text at all (such as for the bird) and when text is produced, it is often not legible. The results obtained by SDEdit preserve the font’s characteristics and the letter’s legibility, but often fail to reflect the desired concept, such as in the case of the bird and the dress.
Additionally, it operates in the raster domain and tends to add details on top of the letter, while our method operates directly on the vector representation of the letters with the objective of modifying their shape. DallE2 manages to reflect the visual concept, however it often fails to produce legible text. When applied with a dedicated prompt to produce the word-as-image of only one letter (fifth column), it manages to produce a legible letter, but there is less control over the output – it is impossible to specify the desired font or to control the size, position, and shape of the generated letter. Therefore, it is not clear how to combine these output illustrations into the entire word to create a word-as-image.

CLIPDraw produces reasonable results conveying the semantics of the input word. However, the results are non-smooth and the characteristics of the font are not preserved (for example observe how the letter ‘E’ differs from the input letter). We further examine CLIPDraw with our shape preservation losses in the next Section.

5.3 Ablation

Figure 11 illustrates the impact of the letter’s initial number of control points. When less control points are used ($P_o$ is the original number of control points), we may get insufficient variations, such as for the gorilla. However, this can also result in more abstract depictions, such as the ballerina. As we add control points, we get more graphic results, with the tradeoff that it often deviates from the original letter. In Figure 15 we show the results of using only the $\nabla P \mathcal{L}_{\text{LSDS}}$ loss. As can be seen, in that case the illustrations strongly convey the semantic concept, however at the cost of legibility. In Figure 16 we analyze the effect of the weight $\alpha$ applied to $\mathcal{L}_{\text{acap}}$. Ranging from 1 to 0. When $\mathcal{L}_{\text{acap}}$ is too dominant, the results may not enough reflect the semantic concept, while the opposite case harms legibility. Figure 13 illustrates a change in the $\sigma$ parameter of the low pass filter. When $\sigma = 1$ almost no blur is applied, resulting in a shape constraint that is too strong.

In Figure 14 we show the results of replacing the $\nabla P \mathcal{L}_{\text{LSDS}}$ loss with a CLIP based loss, while using our proposed shape preservation terms. Although the results obtained with CLIP often depict the desired visual concept, we find that using Stable Diffusion leads to smoother illustrations, that capture a wider range of semantic concepts.

By using the hyperparameters described in the paper, we are able to achieve a reasonable balance between semantics and legibility. The parameters were determined manually based on visual assessments, but can be adjusted as needed based on the user’s personal taste and goals.

6 CONCLUSIONS

We presented a method for the automatic creation of vector-format word-as-image illustrations. Our method can handle a large variety of semantic concepts and use any font, while preserving the legibility of the text and the font’s style.

There are limitations to our method. First, our method works letter by letter, and therefore, it cannot deform the shape of the entire word. In the future we can try to optimize the shape of several letters. Second, the approach works best on concrete visual concepts, and may fail with more abstract ones. This can be alleviated by optimizing the shape of letters using different concepts than the word itself. Third, the layout of letters can also be automated for example, using methods such as [Wang et al. 2022].

Our word-as-image illustrations demonstrate visual creativity and open the possibility for the use of large vision-language models for semantic typography, possibly also adding human-in-the-loop to arrive at more synergistic design methods of ML models and humans.
"Bear"  |  B | B B B B B
"Singer" | N | N N N N N
"Giraffe" | R | R R R R R

Input  |  1  | 5  | 30  | 200  | Without L\textsubscript{tone}

Fig. 13. Altering the \( \sigma \) parameter of the low pass filter using in the \( L_{\text{tone}} \) loss. On the leftmost column are the original letters and concepts used, then from left to right are the results obtained when using \( \sigma \in \{1, 5, 30, 200\} \), and without \( L_{\text{tone}} \).

<table>
<thead>
<tr>
<th>Input Letter</th>
<th>CLIP loss</th>
<th>SDS loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>S</td>
<td>C</td>
</tr>
<tr>
<td>Q</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 14. Replacing the SDS loss with a CLIP-based loss.

"Bear"  | B | B B B B B
"Singer" | N | N N N N N
"Giraffe" | R | R R R R R

Input  |  1  | 0.75 | 0.5  | 0.25  | Without L\textsubscript{acap}

Fig. 15. The effect of using only the SDS loss: note how the third row simply looks like icon illustrations, while the second row still resembles legible letters.

<table>
<thead>
<tr>
<th>Input Letter</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>Cat</td>
</tr>
<tr>
<td>Only SDS</td>
<td>Music</td>
</tr>
<tr>
<td></td>
<td>Robot</td>
</tr>
<tr>
<td></td>
<td>Cup</td>
</tr>
<tr>
<td></td>
<td>Hands</td>
</tr>
</tbody>
</table>

"Cat"   | "Music"  | "Robot"  | "Cup"  | "Hands"

Fig. 16. Altering the weight \( \alpha \) of the \( L_{\text{acap}} \) loss. On the leftmost column are the original letters and concepts used, then from left to right are the results obtained when using \( \alpha \in \{1, 0.75, 0.5, 0.25, 0\} \).

Fig. 12. Word-as-images produced by our method. This subset was chosen from the random set of words.
Fig. 17. Additional results produced by our method.
7 ACKNOWLEDGMENTS

We are grateful to Richard Hao Zhang for the early discussion of the text-as-image problem. Ali Mahdavi-Amiri and Oren Katzir for reviewing earlier versions of the manuscript and to Anran Qi for assisting in evaluating the Chinese words. This research was supported in part by the Israel Science Foundation (grants no. 2492/20 and 3441/21), Len Blavatnik and the Blavatnik family foundation, and the Tel Aviv University Innovation Laboratories (TI Labs).

REFERENCES


FreeType. 2009. FreeType library. https://freetype.org/

Kunpeng Song, Ligong Han, Bingchen Liu, Dimitris Metaxas, and Ahmed Elgammal. 2022. Diffusion Guided Domain Adaptation of Image Generators. https://doi.org/10.48550/ARXIV.2206.04655


RECEIVED...

REFERENCES


FreeType. 2009. FreeType library. https://freeType.org/


FreeType. 2009. FreeType library. https://freeType.org/


Yael Vinker, Yuval Alahaf, Daniel Cohen-Or, and Ariel Shamir. 2022a. CLIPascene: Scene Sketching with Different Types and Levels of Abstraction. https://doi.org/10.48550/ARXIV.2211.17256
SUPPLEMENTARY MATERIAL

A IMPLEMENTATION DETAILS

In this section we provide further implementation details. We intend to release the code to promote future research in this domain.

Our method is based on the pre-trained $\alpha l – 5$ Stable Diffusion model [Rombach et al. 2021], which we use through the diffusers [von Platen et al. 2022] Python package. We optimize only the control points’ coordinates (i.e. we do not modify the color, width, and other parameters of the shape). We use the Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.9, \epsilon = 10^{-8}$. We use learning rate warm-up from 0.1 to 0.8 over 100 iterations and exponential decay from 0.8 to 0.4 over the rest 400 iterations, 500 iteration in total. The optimization process requires at least 10GB memory and approximately 5 minutes to produce a single letter illustration on RTX2080 GPU.

Before we feed the rasterized 600x600 letter image into the Stable Diffusion model, we apply random augmentations as proposed in CLIPDraw [Frans et al. 2021]. Specifically, perspective transform with a distortion scale of 0.5, with probability 0.7, and a random 512x512 crop. We add the suffix “a [word]. minimal flat 2d vector. lineal color. trending on artstation.” to the target word $W$, before feeding it into the text encoder of a pretrained CLIP model.

B COMPARISONS

As described in Section 5.2 we define five baselines to compare with.

In this section we provide more details about the evaluation and more qualitative results. For (1) SD, we run Stable Diffusion [Rombach et al. 2021] with the default hyper parameters of 50 inference steps and a guidance scale of 7.5. We use the prompt “Word as image of the word [word]. [font] font. minimal flat 2d vector. lineal color. black and white style”.

For (2) SDEdit [Meng et al. 2022], we utilized the diffusers [von Platen et al. 2022] implementation, using the prompt “A [word], minimal flat 2d vector. lineal color. black and white style”, and the rasterized input letter as the reference image. We use the default values of 50 inference steps and a guidance scale of 7.5. We use a strength value of 0.85. The strength value determines the quantity of noise added to the input image – a value close to 0.0 results in higher degree of variation in the output, and vice versa.

We use the official website of OpenAI to run (3) DallE2 [Ramesh et al. 2022], using the prompt “Word as image of the word [word]. Where the letter [letter] looks like a [word]. [font] font. minimal flat 2d vector. lineal color. black and white style”. To encourage the manipulation of a specific letter, for (4) DallE2-letter we use the prompt ”The letter [letter] in the shape of a [word]. [font] font. minimal flat 2d vector. lineal color. black and white style”. For (5) CLIPDraw [Frans et al. 2021], we use the author’s official implementation with the recommended hyper-parameters. Instead of using randomly initialized strokes, we use our vectorized letter as input, along with the prompt “A [word]. [font] font. minimal flat 2d vector. lineal color. black and white style”. We provide more comparisons to the methods described above in Figure 20.

C PERCEPTUAL STUDY

In this section, we provide more details about the perceptual study described in Section 5.1. The randomly chosen objects, fonts, and letters are shown in Table 2. A few visual examples are shown in Figure 19.

We provide additional results of our generated word-as-images. In Figures 21-32 we show results of selected words and unique fonts.

D ADDITIONAL RESULTS

Fig. 18. Some additional examples of word-as-image applied on Chinese characters. In Chinese, a whole word can be represented by one character. Here we show from left: bird, rabbit, cat and surfing (two last characters together). The complexity of characters imposes an additional challenge for our method. This could be alleviated in the future for example by dividing the characters to radicals and applying the method only on parts of the character.

Fig. 19. Examples of illustrations presented in the perceptual study. Each pair in the top part shows illustrations obtained using our proposed method (left) and using only SDS loss (right). On the bottom is an example of an illustration presented for the font recognition questions.
Fig. 20. Comparison to alternative methods based on large scale text-to-image models. On the left are the letters used as input (only for SDEdit, CLIPDraw, and ours), as well as the desired object of interest. The results from left to right obtained using Stable Diffusion [Rombach et al. 2021], SDEdit [Meng et al. 2022], DallE2 [Ramesh et al. 2022], DallE2 with a letter specific prompt, CLIPDraw [Frans et al. 2021], and our single-letter results.

In Figures 33-48 we show the results obtained for the random set of words.
Table 2. Randomly chosen objects, letters, and fonts for the perceptual study.

<table>
<thead>
<tr>
<th>Object</th>
<th>Letter</th>
<th>Font</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pineapple</td>
<td>P</td>
<td>Noteworthy-Bold</td>
</tr>
<tr>
<td>Orange</td>
<td>O</td>
<td>Quicksand</td>
</tr>
<tr>
<td>Rugby</td>
<td>Y</td>
<td>Noteworthy-Bold</td>
</tr>
<tr>
<td>Soccer</td>
<td>S</td>
<td>Noteworthy-Bold</td>
</tr>
<tr>
<td>Bear</td>
<td>B</td>
<td>Bell MT</td>
</tr>
<tr>
<td>Lion</td>
<td>O</td>
<td>Quicksand</td>
</tr>
<tr>
<td>Singer</td>
<td>N</td>
<td>Noteworthy-Bold</td>
</tr>
<tr>
<td>Pilot</td>
<td>P</td>
<td>Noteworthy-Bold</td>
</tr>
<tr>
<td>Coat</td>
<td>O</td>
<td>HobeauxRoccoceaux-Sherman</td>
</tr>
<tr>
<td>Shirt</td>
<td>S</td>
<td>Bell MT</td>
</tr>
</tbody>
</table>

Fig. 21. Word-as-image illustrations created by our method.
Fig. 22. Word-as-image illustrations created by our method.

Fig. 23. Word-as-image illustrations created by our method.

Fig. 24. Word-as-image illustrations created by our method.
Fig. 25. Word-as-image illustrations created by our method.

Fig. 26. Word-as-image illustrations created by our method.
Fig. 27. Word-as-image illustrations created by our method.

Fig. 28. Word-as-image illustrations created by our method.
Fig. 29. Word-as-image illustrations created by our method.

Fig. 30. Word-as-image illustrations created by our method.
Fig. 31. Word-as-image illustrations created by our method.

Fig. 32. Word-as-image illustrations created by our method.
Fig. 33. Word-as-image illustrations created by our method for randomly chosen words.

Fig. 34. Word-as-image illustrations created by our method for randomly chosen words.
Fig. 35. Word-as-image illustrations created by our method for randomly chosen words.

Fig. 36. Word-as-image illustrations created by our method for randomly chosen words.
Fig. 37. Word-as-image illustrations created by our method for randomly chosen words.

RUGBY SKIRT TOKYO PANTS
RUGBY SKIRT TOKYO PANTS
RUGBY SKIRT TOKYO PANTS
RUGBY SKIRT TOKYO PANTS
RUGBY SKIRT TOKYO PANTS
RUGBY SKIRT TOKYO PANTS
RUGBY SKIRT TOKYO PANTS
RUGBY SKIRT TOKYO PANTS

Fig. 38. Word-as-image illustrations created by our method for randomly chosen words.

TIGER SHOES PARIS DRESS
TIGER SHOES PARIS DRESS
TIGER SHOES PARIS DRESS
TIGER SHOES PARIS DRESS
TIGER SHOES PARIS DRESS
TIGER SHOES PARIS DRESS
TIGER SHOES PARIS DRESS
Fig. 39. Word-as-image illustrations created by our method for randomly chosen words.

Fig. 40. Word-as-image illustrations created by our method for randomly chosen words.
Fig. 41. Word-as-image illustrations created by our method for randomly chosen words.

Fig. 42. Word-as-image illustrations created by our method for randomly chosen words.
Fig. 43. Word-as-image illustrations created by our method for randomly chosen words.

Fig. 44. Word-as-image illustrations created by our method for randomly chosen words.
Fig. 45. Word-as-image illustrations created by our method for randomly chosen words.

Fig. 46. Word-as-image illustrations created by our method for randomly chosen words.
Fig. 47. Word-as-image illustrations created by our method for randomly chosen words.
Fig. 48. Word-as-image illustrations created by our method for randomly chosen words.