DualVector: Unsupervised Vector Font Synthesis with Dual-Part Representation Supplementary Material

Ying-Tian Liu[1] Zhifei Zhang[2] Yuan-Chen Guo[1] Matthew Fisher[2]
Zhaowen Wang[2] Song-Hai Zhang[1*]
[1] BNRist, Department of Computer Science and Technology, Tsinghua University [2] Adobe Research
{liuyingt20@mails., guoyc19@mails., shz@}tsinghua.edu.cn
{zzhang, matfishe, zhawang}@adobe.com

A. Implementation Details

A.1. Network Architecture and Evaluation

In our implementation of DualVector, we set the number of dual parts and curves as $N = 6$ and $M = 4$. The architecture of the image encoder $\mathcal{E}, \mathcal{E}_f$ and the decoder $\mathcal{D}_I$ is shown in Tab. 1, which is similar architecture to that presented in VQGAN [1]. The output image $\mathcal{I}$ is with a resolution of 256×256. The path decoder $\mathcal{D}_P$ is implemented as an MLP in the size of $[256, 256, 256, 8MN]$. The dimension for the intermediate variables z, f_i, μ, σ, T is set to 256. In the font generation task, SA contains 4 stacked self-attention layers each with 4 attention heads and the FFN contains a linear layer in the size of $[512, 256]$. We train the model for font reconstruction using Adam optimizer [2] with a learning rate of 1×10^{-3} with cosine annealing decay [4]. For the fine-tuning and refinement step, we adopt a learning rate of $2.5 \times 10^{-4}, 0.5$, respectively. In the contour refinement step, the canvas of the SVG glyph is set to have a side length of 256. For the font reconstruction task, we set λ's as $\lambda_P = 0.5, \lambda_I = 1, \lambda_u = 1$ and the UDF warm-up is adopted in the first 8 epochs, lasting ~ 160 epochs in all. For the font generation task, we first train the encoding parts, i.e. $\mathcal{E}_f$, SA, FFN and the embedding producing T, with the latent guidance and KL loss for 32 epochs to achieve a good initialization. We set $\lambda_{latent} = 20, \lambda_{kl} = 1.25 \times 10^{-4}$ during this process. After that, we unfreeze the whole network and train it end-to-end with $\lambda_P = 1, \lambda_I = 0.5, \lambda_{kl} = 1.25 \times 10^{-4}$ until convergence. Due to the limited speed of DeepVecFont [6] and our method in generating fonts, we only evaluate the metrics on the first 200 fonts of the 1425 fonts in our all experiments for all methods.

A.2. Contour Refinement

To be clear, the contour $\partial\mathcal{O}$ contains multiple closed paths. Each path may consist of a different number of segments that are either straight line segments or Bézier curve segments. The SVG canvas is of size 256×256 in this stage. The contour refinement for each glyph lasts for 200 steps. We set $\lambda_{reg} = 10^{-6}$ for refinement.

Pruning Prior to the refinement, we first prune the paths $\{C_1, C_2, ..., C_K\}$ on the contour. Let $S(\cdot)$ denote the area. We remove C_i that satisfies $S(C_i) < 50$.

Simplification There are three different simplification strategies. (a) For every 50 iteration steps, we replace segments that are too short (with length < 3) with the average of their start and end points. (b) After that, we replace the Bézier curves $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ with $\vec{\boldsymbol{ac}}$ if $\angle \boldsymbol{abc} > 171°$. (c) After 150 iterations, we join the adjacent segments if they could be replaced by a single segment without losing many details. To be concrete, we join two adjacent line segments if they are at an angle greater than 175°. For adjacent Bézier curves, we first convert them to the implicit quadratic curve form. The differences between the coefficients of the implicit curves can roughly reflect their proximity. If $||e_1 - e_2||_2$ is less than 0.02 where e_1, e_2 are their normalized coefficients respectively, we consider them close enough and join them.

Subdivision After 50 iterations, we split Bézier curves with a length greater than 25.6 (10% of the canvas width) at their midpoints.

In Fig. 1, we show an example of the refinement process, including how the loss changes with the iteration step, the strategy adopted at certain times, and the intermediate results. The figure illustrates that our contour refinement step can improve the quality and representation efficiency of the contours.

Table 1. The architecture of the image encoder $\mathcal{E}, \mathcal{E}_f$ and the decoder $\mathcal{D}_I$, which is similar to that presented in [1]. The non-linear activation, Swish [5], is omitted. The input shape to the image encoder is $1 \times 128 \times 128$.

Module	Block/Layer	Output Shape
Encoder $\mathcal{E}, \mathcal{E}_f$	Conv2D	$16 \times 128 \times 128$
	Residual Block + Downsample Block	$16 \times 64 \times 64$
	Residual Block + Downsample Block	$32 \times 32 \times 32$
	Residual Block + Attention Block + Downsample Block	$32 \times 16 \times 16$
	Residual Block + Downsample Block	$64 \times 8 \times 8$
	Residual Block + Downsample Block	$64 \times 4 \times 4$
	Residual Block + Downsample Block	$128 \times 2 \times 2$
	Residual Block + Downsample Block	$256 \times 1 \times 1$
	Residual Block	$256 \times 1 \times 1$
Decoder $\mathcal{D}_I$	Conv2D	$256 \times 1 \times 1$
	Residual Block + Upsample Block	$256 \times 2 \times 2$
	Residual Block + Upsample Block	$128 \times 4 \times 4$
	Residual Block + Upsample Block	$128 \times 8 \times 8$
	Residual Block + Upsample Block	$64 \times 16 \times 16$
	Residual Block + Upsample Block	$64 \times 32 \times 32$
	Residual Block + Upsample Block	$32 \times 64 \times 64$
	Residual Block + Attention Block + Upsample Block	$32 \times 128 \times 128$
	Residual Block + Upsample Block	$16 \times 256 \times 256$
	Residual Block	$16 \times 256 \times 256$
	GroupNorm + Conv2D	$1 \times 256 \times 256$

B. Unsigned Distance Field Warm-Up

In Fig. 2, we illustrate how the UDF warm-up strategy could help the optimization of the dual part parameters. In this case, we optimize a randomly initialized dual part to a letter 'O' using different losses. The figure shows that using the pixel loss between the rendered image obtained by DiffVG [3] and the target or the loss $\mathcal{L}_P$ based on a differentiable occupancy field will yield similar results. This is because both losses can only produce gradients near the edges of the shape. When shapes fall into a hole, the losses tend to shrink the positive path and expand the negative path until they produce an entirely white solid shape. The UDF loss $\mathcal{L}_u$ can drive the shape to the approximately correct position so that it does not fall into local minima, producing results with minimal error. Therefore, we claim that the UDF warm-up will help with the initialization of dual parts in the early stage of training.

C. Font Sampling and Interpolation Results

We show more font sampling results in Fig. 4 in SVG format. Also, we provide font interpolation results in Fig. 5. The fonts at both ends of the interpolation are also generated by DualVector. We obtain intermediate results by performing linear interpolation in the latent space of the font style codes. We also provide in the attachment files some TrueType fonts which can be installed on the computer to display in various text editors. They are converted from DualVector's SVG results.

D. Metric for Vector Fonts

Evaluating the quality of synthetic fonts based on metrics in the vector domain is valuable. But designing a reasonable metric on vector graphics remains an open problem and is worth investigating. Here we devise a Chamfer-Distance-style metric between two vector glyphs. Suppose the two glyphs have contours U, V on the unit square canvas, we uniformly sample n points $\{u_i\}_{i=1}^n$ on U and calculate their average distance to V and vice versa. The "vector distance" between U, V is defined as

$$d_{VD}(U,V) = \frac{1}{n}\sum_{i=1}^{n} \min_{v \in V} ||u_i - v||_2 + \frac{1}{n}\sum_{i=1}^{n} \min_{u \in U} ||v_i - u||_2$$

We evaluate the metric ($n = 100$) for both font reconstruction and generation tasks (Tab. 2). DualVector achieves the best fit in terms of contouring.

E. Compact and Topologically Correct Vector Fonts

DualVector learns the shape decomposition from only glyph images and achieves compact vector font synthesis. But DeepVecFont [6] may produce results with wrong topologies even with vector supervision. The difference is

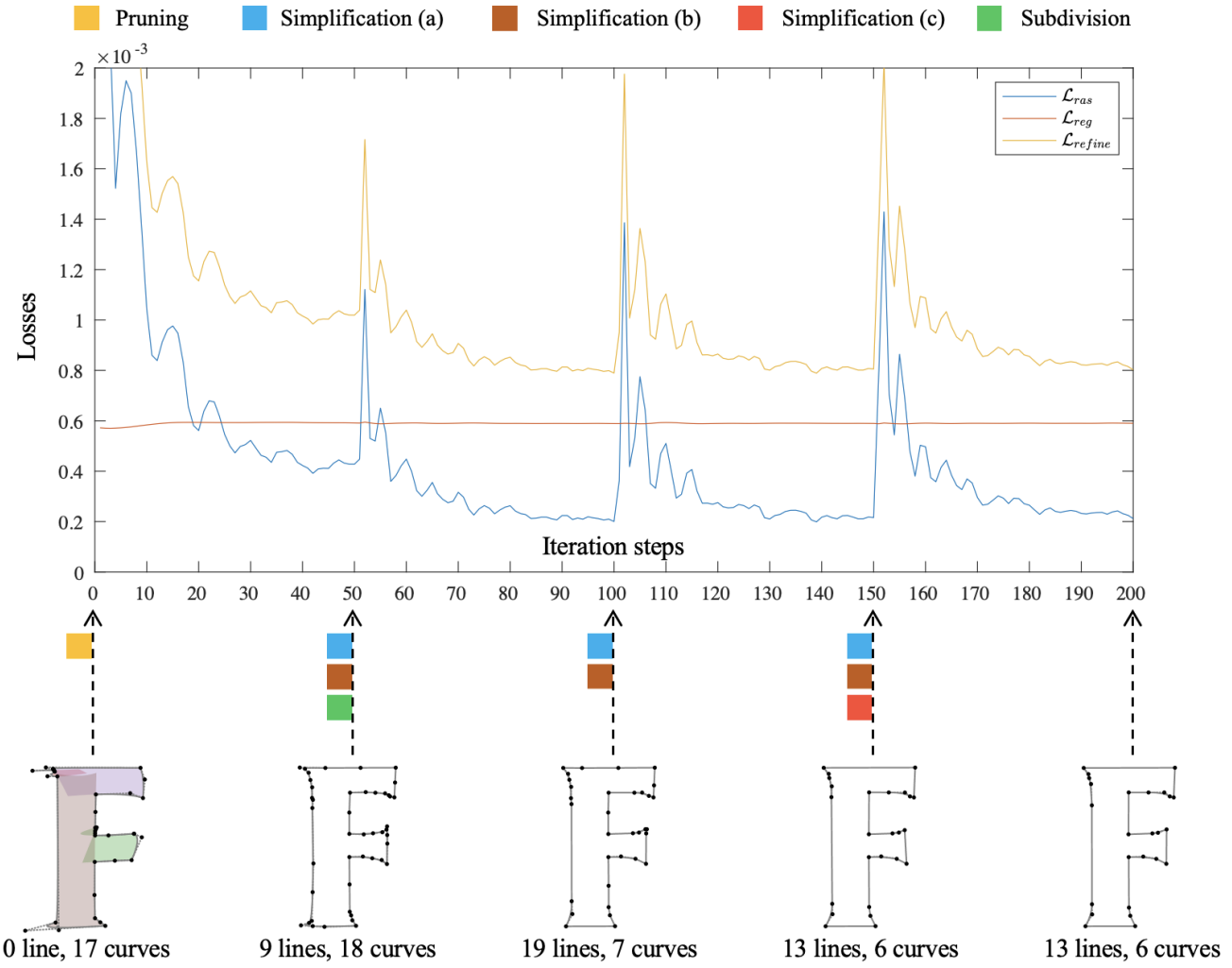

Figure 1. An example of the refinement process. We plot the loss $\mathcal{L}_{refine}$ as a function of iteration steps. The operations are marked near the timeline with different colors. We also show the contour along with the endpoints and control points throughout the refinement. We record the number of lines and curves of the contour under each intermediate result to illustrate how the composition of the contour changes as the refinement goes on.

Table 2. Evaluation of the "vector distance" metric for the font reconstruction and generation tasks.

Method	d_{VD}(recon)↓	d_{VD}(generation)↓
Ours	**0.0082**	**0.0346**
DeepVecFont	-	0.0491
Multi-Implicits	0.0170	0.0395
Im2Vec	0.0473	0.0658

that DeepVecFont [6] predicts the glyph contour directly, including the type and parameters of SVG commands, which are both predicted by the network. If the SVG command prediction is wrong, it will easily produce unwanted or self-intersecting curves. DualVector, on the other hand, separates shape prediction and SVG production, decreasing the difficulty. It outputs the parameters of dual parts and calculates the contour later. The contour calculation and refinement can avoid self-intersection and decide the type of SVG elements for each segment, respectively. We show a typical example in Fig. 3. DeepVecFont [6] relies on the robustness of the network to ensure that the topology is reasonable. Our method eliminates the possibility of self-intersection by a deterministic contour calculation. In this way, DualVector produces a more compact vector glyph.

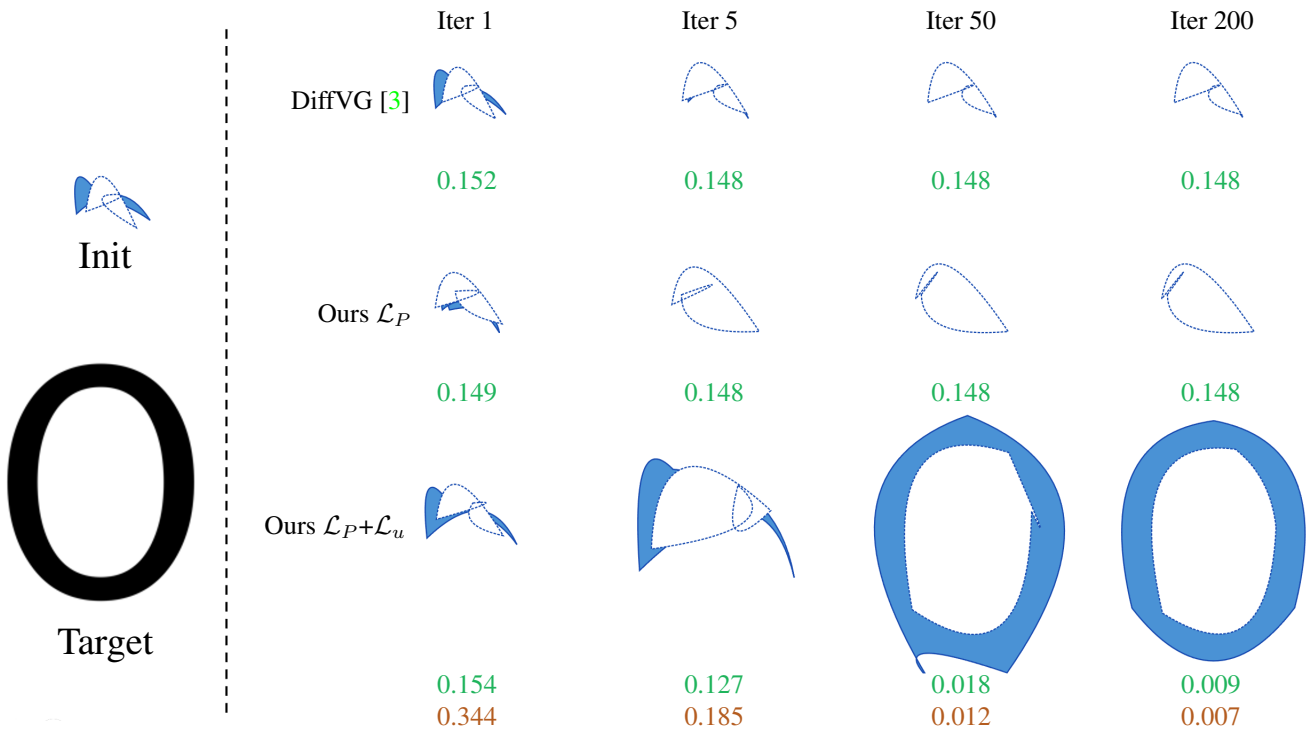

Figure 2. A comparison between the pixel loss of DiffVG [3], our occupancy loss in Eq.(10) in the main paper, and our occupancy loss with UDF warm-up. We optimize a randomly initialized dual part with 6 Bézier curves in the positive and negative parts with the above three losses. We plot the dual part at the 1st, 5th, 50th, and 200th iteration steps. The positive path is solid and the negative one is drawn with dashed lines. We also mark the mean square error between the rendered image and the target and the UDF warm-up loss $\mathcal{L}_u$ under each dual part.

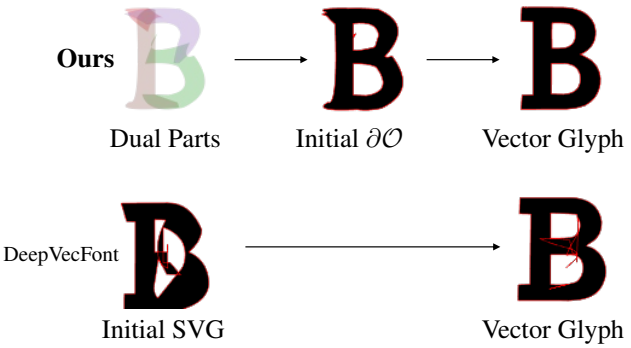

Figure 3. A comparison between our method and DeepVecFont [6] in the process of synthesizing vector fonts. We produce dual parts and then calculate and simplify contours, while DeepVecFont [6] predicts the contours directly and refines them later.

F. Dual Part Correspondence

In Fig. 6, we show the results of reconstructing several different styles of glyphs of the same letter with dual parts. We observe that the same dual part usually carries a single semantic meaning when representing the same letter in different styles, for example, the third part (green) focuses on reconstructing the lower part of its right vertical line when representing 'M'. The correspondence established by our method through dual parts is positive for downstream tasks such as interpolation or generation. To further illustrate the effect of such correspondence, we show the results of interpolating their dual parts between two styles of the same letter in Fig. 7. The correspondence between dual parts makes the interpolation very smooth, indicating the good properties of glyphs' latent space, which is why DualVector can generate high-quality glyphs.

References

[1] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 12873–12883. Computer Vision Foundation / IEEE, 2021. 1, 2

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, pages 1–15, 2015. 1

[3] Tzu-Mao Li, Michal Lukác, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization

for editing and learning. *ACM Trans. Graph.*, 39(6):193:1–193:15, 2020. 2, 4

[4] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *ICLR (Poster)*. OpenReview.net, 2017. 1

[5] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 7(1):5, 2017. 2

[6] Yizhi Wang and Zhouhui Lian. Deepvecfont: synthesizing high-quality vector fonts via dual-modality learning. *ACM Trans. Graph.*, 40(6):265:1–265:15, 2021. 1, 2, 3, 4

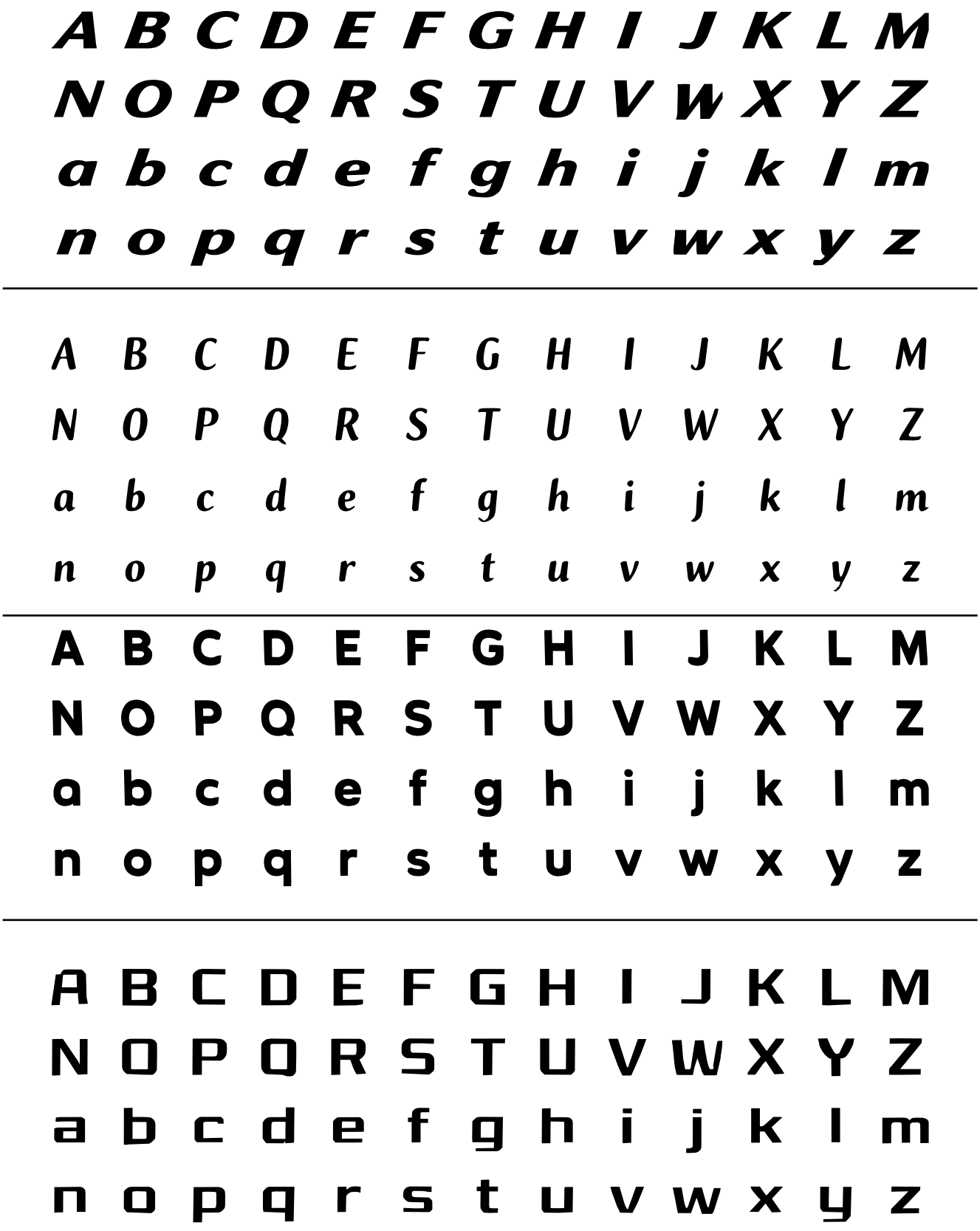

Figure 4. New fonts generated by DualVector in SVG format.

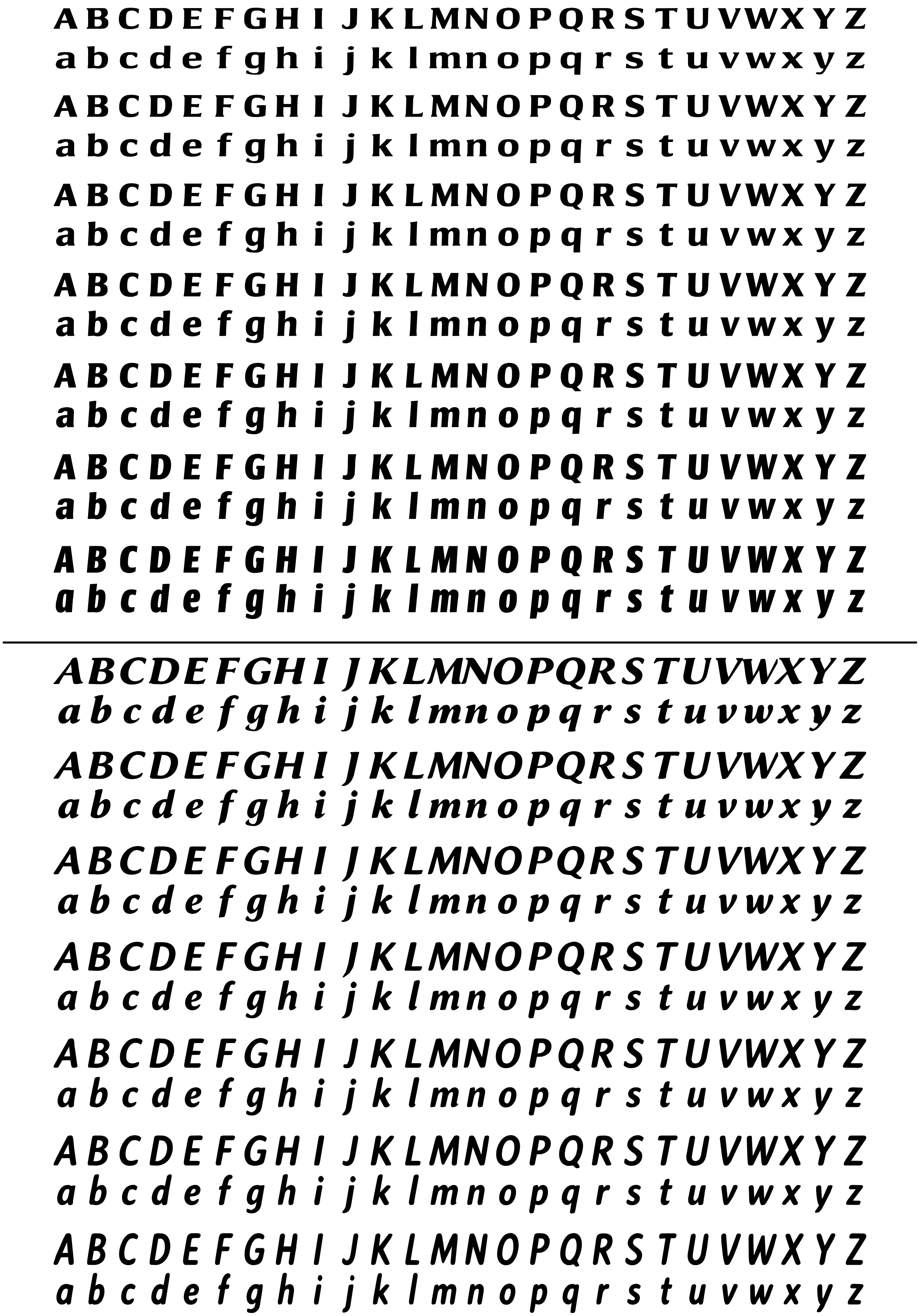

Figure 5. Font examples with linear interpolated styles in the latent space.

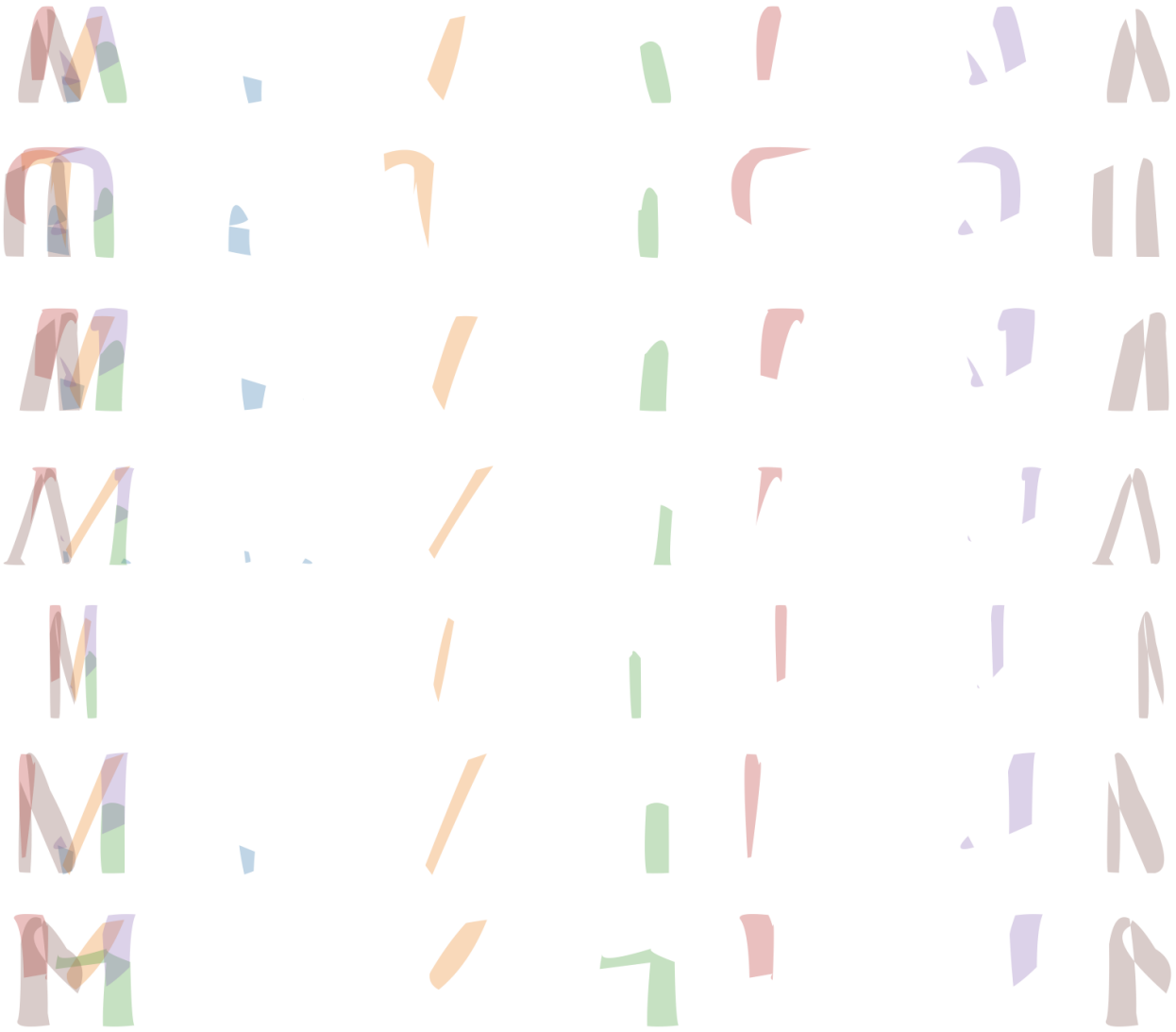

Figure 6. Different styles of the letter ‘M’ are formed by dual parts with correspondence.

Figure 6. Different styles of the letter ‘R’ are formed by dual parts with correspondence.

Figure 6. Different styles of the letter ‘o’ are formed by dual parts with correspondence.

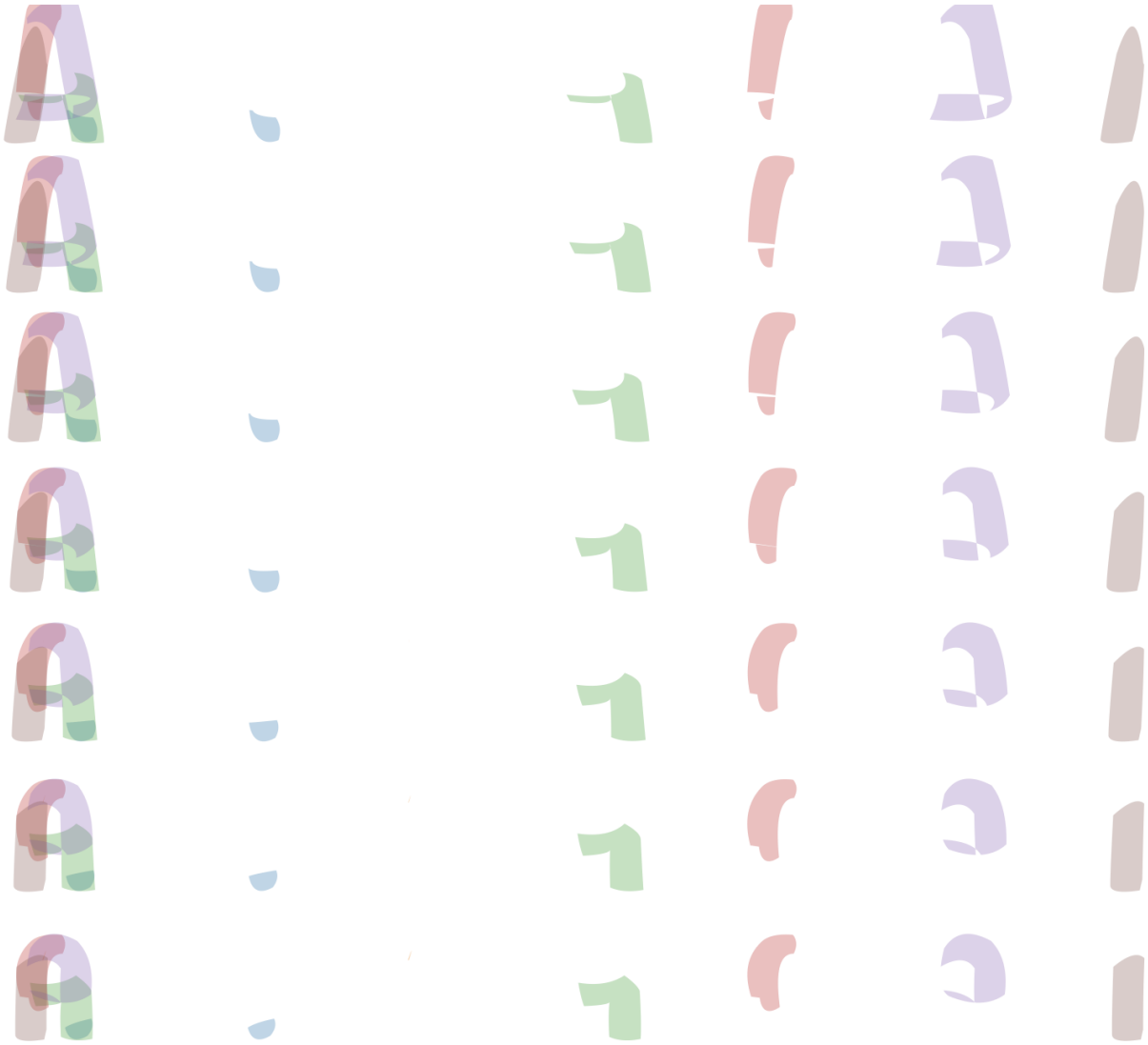

Figure 7. The interpolation of dual parts between two styles of ‘A’.

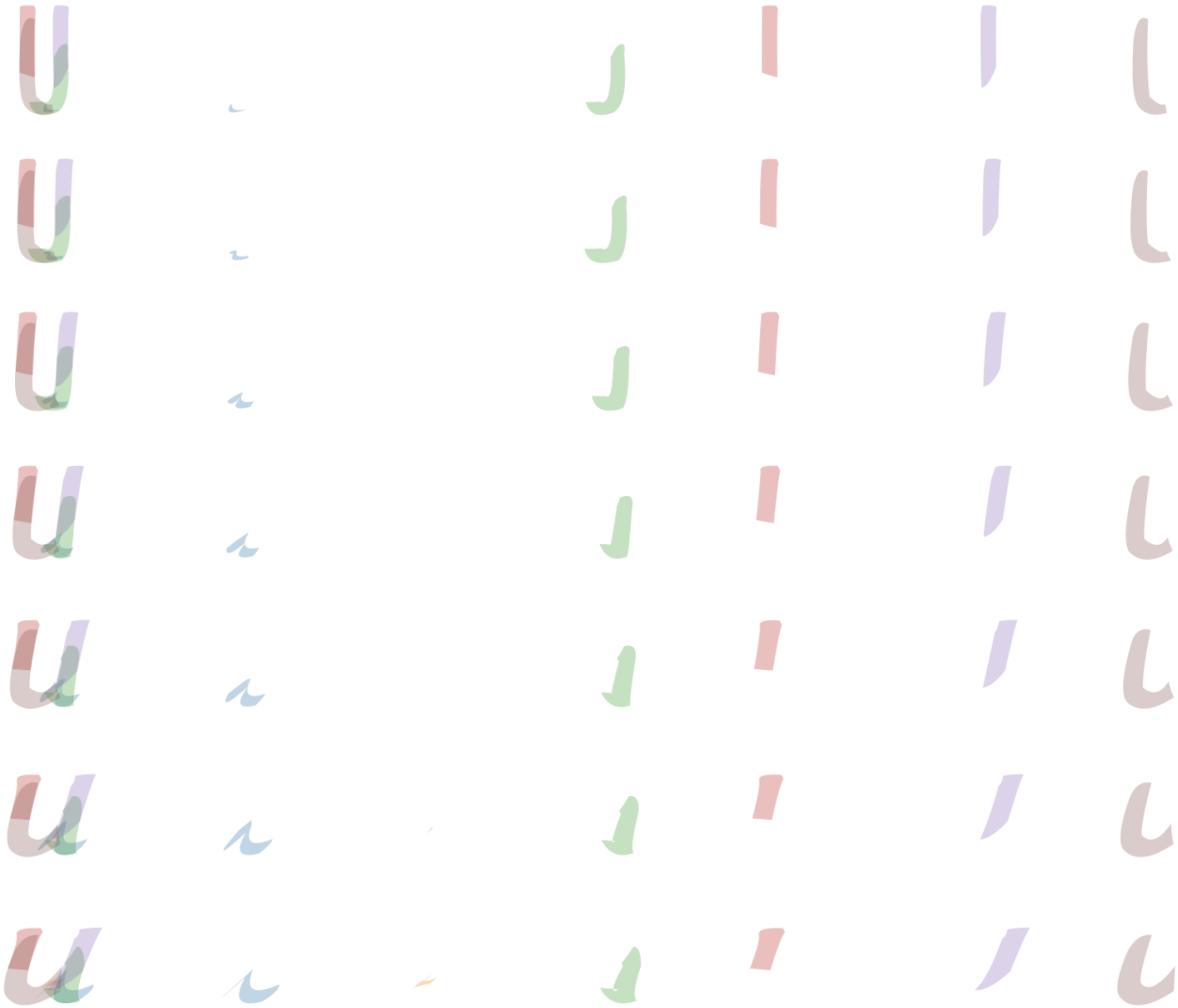

Figure 7. The interpolation of dual parts between two styles of ‘U’.