DOI: 10.1111/cgf.13265

COMPUTER GRAPHICS *forum*
Volume 37 (2018), number 1 pp. 273–287

Improved Corners with Multi-Channel Signed Distance Fields

V. Chlumský[1], J. Sloup[2] and I. Šimeček[1]

[1]Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
viktor.chlumsky@gmail.com, ivan.simecek@fit.cvut.cz
[2]Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic
sloup@fel.cvut.cz

Abstract
We propose an extension to the state-of-the-art text rendering technique based on sampling a 2D signed distance field from a texture. This extension significantly improves the visual quality of sharp corners, which is the most problematic feature to reproduce for the original technique. We achieve this by using a combination of multiple distance fields in conjunction, which together provide a more thorough representation of the given glyph's (or any other 2D shape's) geometry. This multi-channel distance field representation is described along with its application in shader-based rendering. The rendering process itself remains very simple and efficient, and is fully compatible with previous monochrome distance fields. The introduced method of multi-channel distance field construction requires a vector representation of the input shape. A comparative measurement of rendering quality shows that the error in the output image can be reduced by up to several orders of magnitude.

Keywords: curves and surfaces, modelling, texture synthesis, rendering

ACM CCS: I.3.3 [Computer Graphics]: Picture/Image Generation-Line and curve generation

1. Introduction

The problem of text rendering is among the most prevalent in computer graphics. Sometimes, an expensive method can be used to pre-render the text once in high quality, and the result is reused for the remainder of its time on screen. In other situations, however, such as when the appearance of the text rapidly changes along with perspective or due to other factors, a more efficient method has to be employed.

One of the much used state-of-the-art methods of rendering text and other vector-based two-dimensional (2D) art efficiently is the technique of sampling a pre-computed signed distance field (SDF) from a texture, as presented by Green [Gre07]. This technique is especially powerful in real-time rendering of 3D scenes, but its extraordinary quality to performance ratio makes it more than viable in many other settings as well. Because of very low video memory consumption and simple implementation, it is a popular choice even for 2D text in computer games (e.g. Team Fortress 2 by Valve as stated in [Gre07]). Due to the availability of 3D transformations in HTML 5 and CSS3, it has recently been, for example, adopted into the rendering engine of Google Chrome (see [tC15]).

The aforementioned technique, however, suffers from an inability to correctly reproduce sharp corners. Although this can be mitigated by increasing the resolution of the distance field texture (see Figure 1—left and middle), using up more video memory and lowering performance, the imperfection can never be eliminated completely. We have developed an extension to Green's technique, which is capable of rendering sharp corners precisely. We achieve this by using a combination of several distance fields instead of only one, which are stored together in a colour texture—as a multi-channel distance field (see Figure 1—right).

In the rest of the paper, we first briefly review the related state-of-the-art text rendering techniques based on sampling a 2D SDF from a texture and several alternative methods used in practice in Section 2. Section 3 explains the nature of the multi-channel distance field representation. The proposed method and its building blocks are presented in Section 4. Section 5 goes further into the implementation details of multi-channel distance field construction, with Section 6 covering how the SDFs can be used to draw text or other vector shapes. Experimental results are shown in Section 7 and summarized in Section 9.

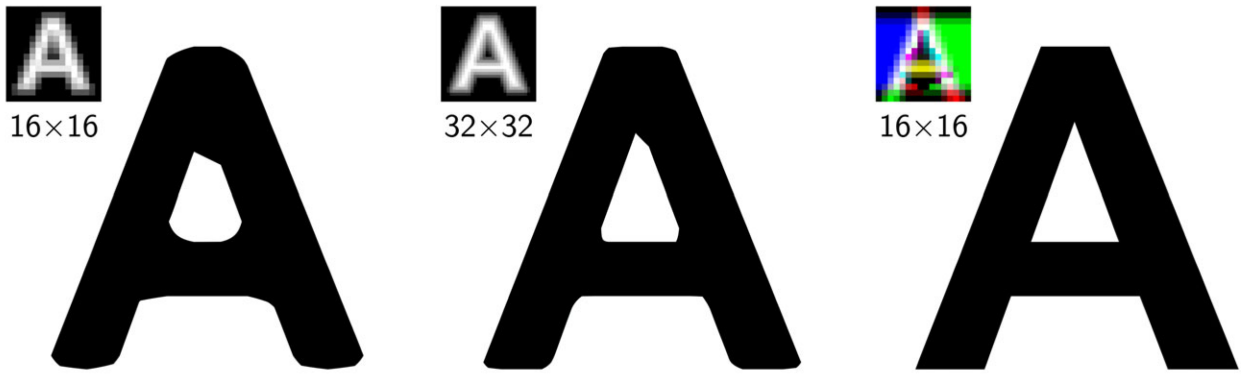


Figure 1: *Rendering of the glyph 'A' using conventional distance fields at different resolutions (*16 × 16 *on the left,* 32 × 32 *in the middle), and using our multi-channel distance field method at distance field resolution* 16 × 16 *(right).*

2. Previous Work

There has been a lot of work done to perfect text and 2D vector rendering and many different technologies have been developed, each with specific use cases, advantages and drawbacks. We summarize the most significant ones and compare them to the distance field technique. Apart from the simplest, yet widely used methods, of using raster glyphs or triangle meshes, there are very few alternatives to SDFs used in practice.

Raster glyphs This is the simplest method, where character glyphs are treated as regular images. When the glyphs are magnified, they will become blurry or pixelated. Compare middle and right images in Figure 2 as a good example of the quality difference. It is only marginally faster than the distance field method, but this is negated by the need for higher image resolution.

Triangle meshes Another possibility is converting the vector glyphs into triangle meshes. However, since modern font formats define their glyph shapes using parametric curves, this conversion has to be approximate, and depending on the original complexity of the glyph's shape and the desired quality, the number of triangles can become impractically high, especially considering that plenty of characters may be needed to be displayed at the same time. Therefore, the quality to performance ratio is generally much worse than what distance fields have to offer.

Exact vector rendering A more advanced version of the triangle mesh rendering method is the possibility of using programmable shaders to draw parametric curves exactly [LB05, LB07]. This is an exact method, and therefore provides higher quality than distance fields, but its performance cost is even higher than that of the triangle mesh method. The glyphs also have to be divided into a lot of triangles, but additionally, complex shader calculations must be performed in the rendering process (see Figure 3). A similar, resolution independent 'stencil then cover' approach intended for path rendering [KB12] was implemented directly as an NVIDIA-proprietary OpenGL extension. Stand-alone direct implementation in OpenGL leads to a high command overhead and results in a higher number of stencil operations growing with screen resolution [YKB*14].

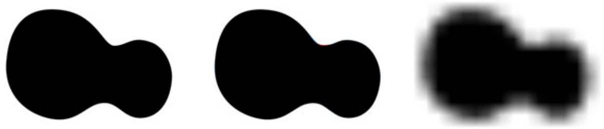


Figure 2: *Demonstration of reconstructing the original shape (left) from a low-resolution distance field (middle) and from a low-resolution image (right).*

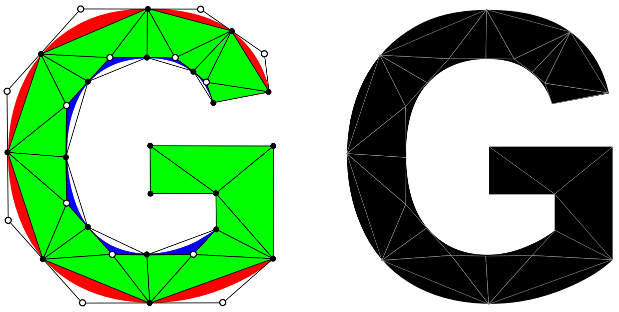


Figure 3: *An illustration of a TrueType font outline triangulation is shown on the left. As stated by [LB05] "The green triangles are interior to the shape and are entirely filled. The red (convex) and blue (concave) curves within triangles are rendered using a pixel shader program. The resulting shape on the right can be arbitrarily transformed projectively and remains resolution independent."*

Advanced texture-based methods There are other more advanced niche techniques which attempt to eliminate the need for large triangle meshes and encode the geometry of the shapes in textures. One such technology is [RCL05], which reconstructs the shape using functions whose coefficients it looks up from texture data. Another one is the experimental library called GLyphy [Esf14], which approximates the shape geometry using circular arcs, and also stores their definitions as texture data. Alternatively, a piece-wise linear representation of the glyph's boundary could be encoded in a separate silhouette map [SCH03].

Pre-existing solutions All of the aforementioned methods offer higher quality than distance fields; however, they require complex shader computations, including multiple texture lookups, making them far less efficient. Because of this, they are rarely used in practice. However, we will focus solely on the state-of-the-art technique based on SDFs and its derivatives.

The main concepts of the distance field technique have been presented most notably by [QMK06], including SDFs and pseudo-distance fields, as well as a method of anti-aliasing and rendering special effects, such as outlines and shadows. With the advent of a programmable graphics pipeline, [Gre07] demonstrated how the technique can be applied in real-time graphics with minimal performance and memory cost, while at the same time, retaining some compatibility with older hardware, where the programmable pipeline is not available. Both of these articles hint at possible improvements in corner reconstruction by utilizing multiple distance fields in conjunction, but no concrete method is given.

As of May 2016, to our best knowledge, no satisfactory solution to this problem has been published. Although Green hinted at the possible improvement of adding more channels to the distance field in 2007, a working solution for multi-channel distance field construction has not been published until November 2014 [Pat14]. This only somewhat successful attempt, which also features multi-channelled distance fields, takes an approach very different to ours. The author himself mentions problems with artefacts and lower robustness, as well as having to treat several special cases. He also uses four colour channels, while our method only needs three for the same task. Based on this limited information about the method, we believe that our final solution is more effective and more reliable, as well as faster and simpler.

3. Multi-Channelled Representation

As [Gre07] suggests, a sharp convex corner can be represented as the intersection of two half-planes. Similarly, a concave corner can be represented as the union of two half-planes. If these half-planes are encoded in two SDFs, the intersection operation can be facilitated as the minimum of the signed distances, and union as the maximum (see Figure 12). However, in general, many corners are present in a single shape or glyph, both convex and concave (e.g. glyph ‘A’ on Figure 1), and therefore a more versatile model for the composition of SDFs is necessary to capture both cases simultaneously. Furthermore, we want to keep the number of distance field components as low as possible since they will be encoded as texture channels. In that sense, we build upon Green’s suggestion to arrive at a general working solution which is presented in the rest of this section.

3.1. Corner analysis

Sampling of a SDF texture during shape reconstruction introduces artefacts which are a consequence of bilinear interpolation, and depending on the alignment of the distance field grid, it may cause corners to appear rounded to a varying degree, as shown in Figure 4.

In order to correctly adjust the distance field to accommodate for sharp corners, we must first understand what exactly happens in their proximity. Let us divide the plane into four quadrants, with the ones lying inside the shape filled. Figure 5 shows the average

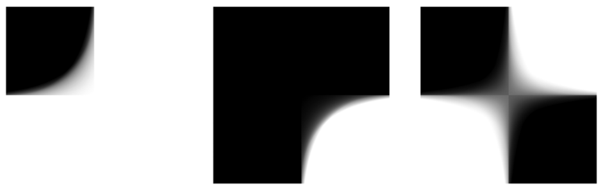


Figure 4: *The average of all possible results of corner reconstruction with varying distance field grid alignment.*

case behaviour of the image reconstructed from an SDF. Figure 5(a) corresponds to a convex corner of the shape, Figure 5(b) to a straight edge and Figure 5(c) to a concave corner. The correct border is shown hatched. There is, however, one other case. When only the opposite quadrants are filled, the result can be one of several possibilities depending on the alignment with the grid of the distance field. The possibilities include the cases (d)–(f) illustrated in Figure 5 and anything in-between.

The amount of rounding off at the corner also depends on the alignment with the distance field’s grid and therefore is essentially random. What is important is that some quadrants are guaranteed to remain homogeneous. Using the observations from Figure 5, we can divide any corner into eight roughly homogeneous areas, as illustrated in Figure 6 on the left.

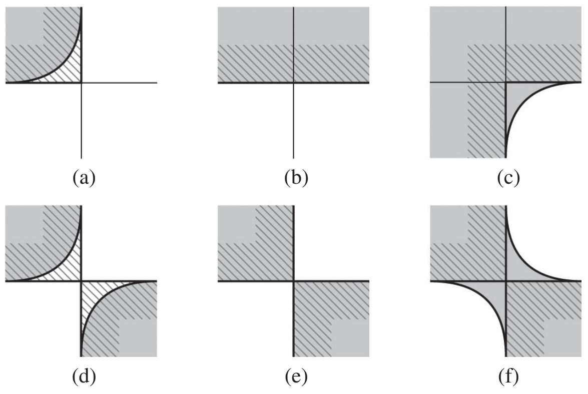


Figure 5: *The average resulting image of filled quadrants from an SDF—top-left only quadrant (a), top-left and top-right quadrants (b), all quadrants except bottom-right (c) and the possible results of filled opposing quadrants (d, e, f).*

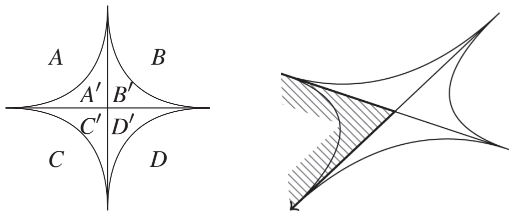


Figure 6: *Dividing the plane into quadrants and subquadrants (left). Example of quadrant alignment of a non-orthogonal corner (right).*

Table 1: *Truth table of the filling of subquadrant areas.*

Intended filling of quadrants				Resulting filling of subquadrants			
A	B	C	D	A'	B'	C'	D'
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	1	1
0	1	0	0	0	0	0	0
0	1	0	1	0	1	0	1
0	1	1	0	?	?	?	?
0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0
1	0	0	1	?	?	?	?
1	0	1	0	1	0	1	0
1	0	1	1	1	1	1	1
1	1	0	0	1	1	0	0
1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1

All of the previous illustrations only capture the case of an axis aligned and orthogonal corner. Of course, a corner can be oriented in any way and can form any angle. The quadrants would then also be aligned differently, always along the two edge segments at the corner, as illustrated in Figure 6 on the right.

Based on whether the shape is filled (1) or not (0) in a given quadrant A through D, it can be determined if the areas A' through D' will appear filled in the resulting image. This is captured in Table 1. The derived filling of subquadrants can be expressed by a Boolean function $f(Q_1, Q_2, Q_3, Q_4, r)$ that determines whether the subquadrant Q'_1 is filled depending on the intended filling of the quadrants Q_1, Q_2, Q_3 and Q_4. Its first parameter Q_1 is the fill of the local quadrant, the following two Q_2 and Q_3 are the neighbouring quadrants, the fourth one Q_4 is the opposite quadrant, and the last one, r, is a random bit, which has a role in the two uncertain scenarios. Taking into account the symmetry of the problem, with respect to each quadrant, we get

$$A' = f(A, B, C, D, r), \tag{1}$$

$$B' = f(B, A, D, C, r), \tag{2}$$

$$C' = f(C, A, D, B, r), \tag{3}$$

$$D' = f(D, B, C, A, r). \tag{4}$$

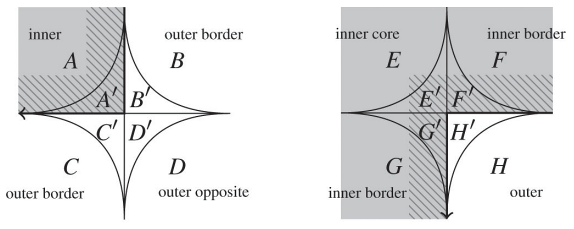
Using a Karnaugh map to represent values in truth table 1 the function $f(Q_1, Q_2, Q_3, Q_4, r)$ can be defined as follows:

$$f(Q_1, Q_2, Q_3, Q_4, r) = f(A, B, C, D, r), \tag{5}$$

$$f(A, B, C, D, r) \overset{\text{def}}{=} AB + AC + BCD + rA\overline{B}\,\overline{C}D + \overline{r}\,\overline{A}BC\overline{D}. \tag{6}$$

3.2. Switching to multiple channels

Using more than one channel in the distance field, the same rules apply for the values in each channel. Instead of using a binary value to denote the filling of an area, we will use binary vectors,



(a) A convex corner (b) A concave corner

Figure 7: *The two possible corner types and their quadrants.*

Table 2: *The required differentiation of binary vectors that denote the inside and outside of the shape as deduced from Figure 7.*

Inside the shape		Outside the shape	
quadrant A	$f_n(A, B, C, D, 0)$ $f_n(A, B, C, D, 1)$	$f_n(B, A, D, C, 0)$ $f_n(B, A, D, C, 1)$	quadrant B
quadrant E	$f_n(E, F, G, H, 0)$ $f_n(E, F, G, H, 1)$	$f_n(C, A, D, B, 0)$ $f_n(C, A, D, B, 1)$	quadrant C
quadrant F	$f_n(F, E, H, G, 0)$ $f_n(F, E, H, G, 1)$	$f_n(D, B, C, A, 0)$ $f_n(D, B, C, A, 1)$	quadrant D
quadrant G	$f_n(G, E, H, F, 0)$ $f_n(G, E, H, F, 1)$	$f_n(H, F, G, E, 0)$ $f_n(H, F, G, E, 1)$	quadrant H

where each dimension corresponds to a certain channel. For any such vector $A \in \mathbb{B}^n$, a function $\mathcal{F}(A) \in \mathbb{B}$ will determine if it marks an area inside the shape (with value 1) or outside (with value 0).

The function f can also be generalized to accept and return binary vectors, performing the original operation separately for each channel:

$$f_n(A, B, C, D, r) \overset{\text{def}}{=} (f(a_1, b_1, c_1, d_1, r), \ldots, f(a_n, b_n, c_n, d_n, r))$$
$$: A, B, C, D \in \mathbb{B}^n, r \in \mathbb{B}. \tag{7}$$

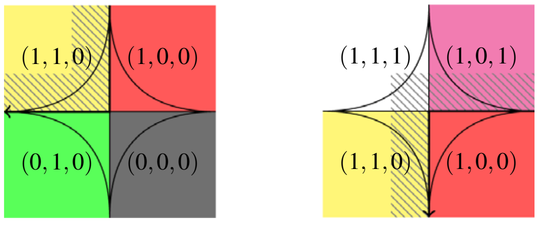
All that is left to do is assign the correct binary vectors (interpreted as colour combinations) for quadrants of convex and concave corners (Figure 7).

To ensure the corners are sharp, the channels must be chosen in such a way that the areas that are supposed to be filled are distinguishable from those that are not. Therefore, the values of vectors $A, B, C, D, E, F, G, H \in \mathbb{B}^n$ must be chosen so that function $\mathcal{F}$ may be evaluated as 1 for all vectors in the left column of Table 2 and as 0 for all vectors in the right column.

3.3. Three distance fields model

By enumeration of all possible values of binary vectors A through H, we have found that the minimum dimension n, for which the vectors in the two columns of Table 2 can be reliably told apart, is 3.

Our proposed model is based on *median of three*. It uses not two but three distance fields, which are combined using the median function. This function can, indeed, simulate both minimum (intersection) and maximum (union) since for any signed distances



(a) A convex corner (b) A concave corner

Figure 8: *Possible colour encoding of a corner's quadrants using the median of three model.*

$d_1 < d_2$, the median of (d_1, d_1, d_2) is the minimum, and the median of (d_1, d_2, d_2) is the maximum.

$$\mathcal{F}(A) \overset{\text{def}}{=} \text{median}(a_1, a_2, a_3). \tag{8}$$

An example of this is shown in Figure 8, where the first dimension of the binary vectors is encoded using the red colour channel, the second dimension with green channel and the third one with blue channel.

The individual dimensions in this model are interchangeable because median disregards the order of arguments. For a convex corner with inner quadrant A, outer border quadrants B and C and outer opposite quadrant D, it must hold that:

$$B + C \leq A, \tag{9}$$

$$BC = D = \vec{0}, \tag{10}$$

$$\mathcal{F}(B) = \mathcal{F}(C) = 0. \tag{11}$$

For any concave corner with inner core quadrant E, inner border quadrants F and G and outer quadrant H, it must hold that:

$$FG \geq H. \tag{12}$$

$$F + G = E = \vec{1}, \tag{13}$$

$$\mathcal{F}(F) = \mathcal{F}(G) = 1. \tag{14}$$

The median of three function can be implemented using four min/max operations, which are natively supported by current graphics hardware:

$$\text{median}(d_1, d_2, d_3) = \max(\min(d_1, d_2), \min(\max(d_1, d_2), d_3)). \tag{15}$$

The three-channel distance field representation can be supplied to the fragment shader in the form of an RGB texture, and the median evaluated immediately after sampling it. From that point on, the median value can be used in exactly the same fashion as the value sampled from an ordinary single-channel distance field in the original method [Gre07]. For distance fields constructed using our method described further on, the median value still corresponds to an approximation of the signed pseudo-distance in all parts of the image, but additionally, it possesses a significantly greater precision near corners.

4. Building Blocks of Proposed Method

Before moving on to the construction of our multi-channel distance field, we need to establish some preliminary concepts—the format of the input, different types of distances and distance fields and the way they are constructed.

4.1. Input representation

Unlike the original technique, we need information about the shape's geometry. Therefore, a vector representation is necessary for the construction of our multi-channel distance field. If it is unavailable only for some shapes or some of its parts, a single-channel distance field can be incorporated into a multi-channel one, using the same value for each channel. This is perfectly compatible with the median model.

The vector representation of each shape consists of closed contours, and each contour is composed of oriented edge segments, which are oriented so that the inside of the shape is always on the same side (see Figure 9). This is the standard representation used by OpenType Font and TrueType Font typeface formats (see [Con00, Ado06]), as well as SVG (see [Gro11]), and other vector formats. A single edge segment can be either a line segment, or a parametric curve (a quadratic or cubic Bézier curve).

We also require that there are no self-intersections in the shape (edges crossing one another). If this is the case, the input representation has to be first transformed by the user to eliminate such occurrences.

4.2. Signed distance and pseudo-distance

By distance, we refer to Euclidean distance in 2D space throughout the text. Although it is normally non-negative, we use a generalized signed distance, which can be negative. The absolute value of the signed distance between a point P and a shape is the minimum distance between P and a point on the shape's edge, and its sign is positive if P lies inside the shape, and negative if it lies outside. Note that this is often defined the opposite way, but our formulation translates better to the distance field representation.

Since all shape's edges are oriented with one side always facing the inside, the signed distance can be determined simply as the

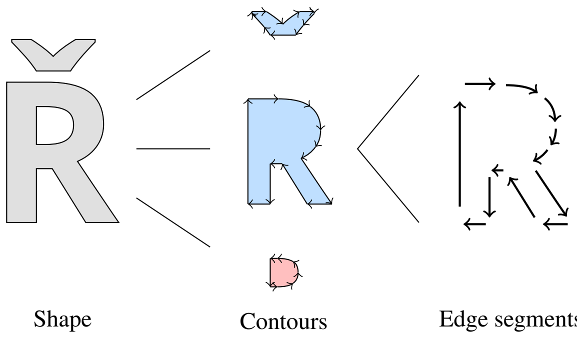


Figure 9: *The structure of a glyph's vector representation. Note that negative contours (cut-outs) are oriented in the opposite direction.*

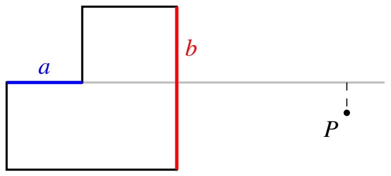


Figure 10: *The signed pseudo-distance from point P to edge a.*

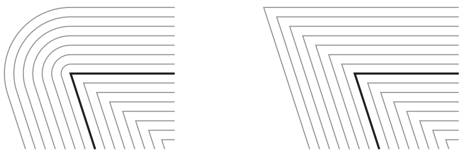


Figure 11: *Contour graph of a regular distance field (left) and a pseudo-distance field (right) near a corner.*

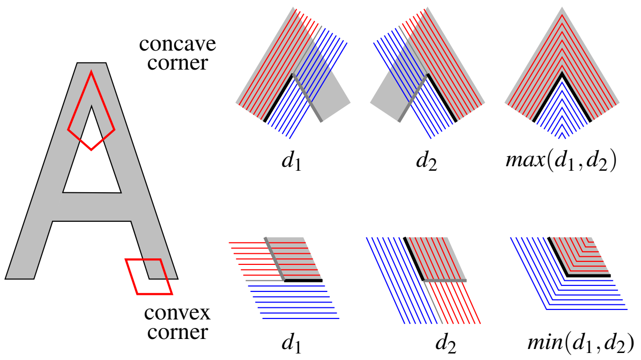


Figure 12: *Preservation of sharp corners utilizing two pseudo-distance fields. The concave corner is created as the maximum of two pseudo-distance fields (top row) and the convex corner as the minimum (bottom row). Note that red colour encodes positive distances and blue negative distances.*

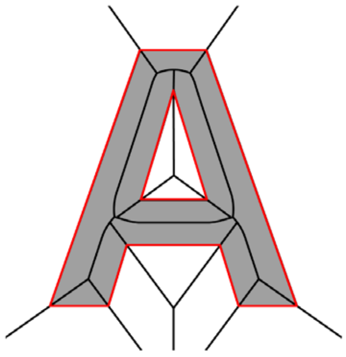


Figure 13: *A generalized Voronoi diagram of the letter 'A' generated by dividing the plane by the closest edge. Note that even though all of the sites are straight lines in this case, the borders between regions become curved.*

absolute distance to the closest edge, with a sign depending on which side of the edge the point of origin lies on.

A special variant is the signed pseudo-distance, introduced by [QMK06]. It is the signed distance from the shape's edges and their infinite continuations, i.e. unlike before, the distance to endpoints is not considered. For parametric curves, there are at least two possible ways this continuation may be formed. Either as straight lines continuing from the endpoints in the curve's immediate direction, or by extending the range of the curve's parameter. Both are feasible with slight visual differences but the former is much less prone to unpredictable results.

An example of pseudo-distance is shown in Figure 10, which also demonstrates that edge pseudo-distances would not be enough to find the shape pseudo-distance. Note that at point P, the minimum pseudo-distance is to the edge a and is positive, which would place P inside the shape. It is clear, however, that edge a is completely irrelevant in this area and that P is, in fact, outside. The correct pseudo-distance to the shape is therefore the pseudo-distance to the edge segment that is closest in terms of true distance (edge b).

The visual difference between the true distance and pseudo-distance metric is that pseudo-distance forms sharp mitres around corners, as demonstrated in Figure 11. The key property of the pseudo-distance metric, however, is that in the distance field representation, it essentially extends the edge segments, as demonstrated in Figure 12.

4.3. Voronoi analysis

To determine which edges affect a certain area of the distance field, one can perform a Voronoi analysis of the shape. The Voronoi diagram is a well-known concept (see [AK00, dBvKOS08]). In its basic form defined by a set of points in the plane, called sites, it partitions the plane into regions, where each region is the set of all points sharing the common closest site in terms of absolute Euclidean distance. In our case, we consider a generalization of the Voronoi diagram, where sites are not zero-dimensional points, but one-dimensional edge segments of the shape—line segments or splines. An example of a generalized Voronoi diagram generated by dividing the plane by the closest edge is shown in Figure 13.

We do not construct a generalized Voronoi diagram directly, but instead determine the Voronoi site, the closest edge segment, independently at each discrete point of the distance field.

As is expected of edge segments, one usually begins exactly where another ends. If we use nothing more than distance as the metric, there will be ambiguous areas, where two neighbouring edge segments are equally distant. This area is marked red in Figure 14. Our goal is to divide the plane between the two edge segments equally, along the axis p. To resolve this, we use a secondary metric when distances are equal, which we will call the *dot* metric. It is the absolute dot product of the direction towards the nearest point of the edge, and the edge's direction at that point. In other words, it says how slanted in respect to the edge the shortest path to that point is. By choosing the closest edge segment with the lowest *dot*, we arrive at a partitioning that will divide the area between the two edges exactly along the axis of their common vertex.

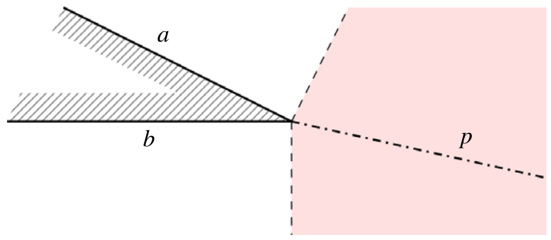


Figure 14: *All points in the red area are equally distant to edges a and b. Axis p splits this area in two equal parts to determine the unique closest edge segment.*

Here, we would like to emphasize that since the computation involves floating point arithmetic, extra care has to be taken to ensure that equal distances are identified. In our implementation, we made sure that the common endpoints of adjacent edges are exactly equal, and that under these circumstances, the distance is computed precisely as the distance between the origin and the endpoint. This guarantees that the result will be exactly the same for both edges.

4.4. Distance fields

Now we need to establish distance fields and their variants. A 2D Euclidean distance field is used as a representation of the input shape. It is a uniform discrete grid of samples of the distance transform function that yields for each point in the plane the signed distance between that point and the shape (see Figure 16 on the left). Since this function usually behaves very predictably, applying bilinear interpolation on the discrete distance field provides a very good approximation of this function. The only areas where the function changes unpredictably are the corners, which is the root of our problem.

A distance field can be constructed simply by evaluating the distance from the shape at each point of the grid, using the desired metric. The true distance field will use the true distance metric, and in a similar way, the pseudo-distance field will utilize the pseudo-distance metric.

Discontinuous distance fields So far, the values in the distance field have been continuous, as is intuitive—when moving in a plane, the distance to some stationary object will always change gradually. This property of the distance field was guaranteed by the fact that the oriented edge segments of a shape always form a continuous closed path. We introduce a new type of distance field, called discontinuous, a generalization of the pseudo-distance field, where the edge segments do not need to meet this requirement. In the space where edges are ‘missing’, areas with vastly different distance values may meet, forming discontinuous ‘fronts’. A disconnected path and its distance transform function can be seen in Figure 16 (middle), as well as a distance field constructed from it (right).

5. Method of Construction

Having established all necessary concepts, the method of creation of the multi-channel distance field itself is rather simple and involves

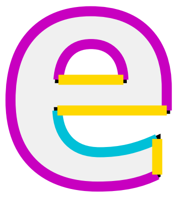


Figure 15: *A valid edge colouring of the lowercase ‘e’ glyph.*

the following three steps: (1) edge colouring, (2) distance field construction and (3) error correction. These steps are described in detail in the rest of the section.

5.1. Edge colouring

First of all, each edge of the shape must be assigned to which of the three channels it belongs to. The assignment rules are:

1. Every edge segment must belong to at least two channels.
2. Two edges meeting at a corner must have only one channel in common.
3. Two smoothly connected edge segments must have at least two channels in common.

We can see these rules applied on an example of the lowercase ‘e’ glyph in Figure 15. The red channel, for instance, contains all yellow (red+green) and magenta (red+blue) edges, but not the single cyan (green+blue) edge.

Edges of most shapes can be coloured according to these rules in many different ways. A trivial way is to start for each contour with one secondary colour at an arbitrary edge, and then continue along the contour, alternating between the remaining two, switching the colour at every corner (see Algorithm 1). This strategy works for any path except one special case, when a single smooth edge connects to itself with a sharp corner, as in a teardrop-like shape. In this case, we simply split the edge into three segments, and proceed to colour them according to the rules, which implies the middle segment being white. The white colour should also be used for any completely smooth contours.

Algorithm 1 A simple edge colouring procedure satisfying the established rules.

```
procedure EDGECOLOURING($s$)
  for each contour $c$ of shape $s$ do
    if $c$ has only one edge then
      $current \leftarrow (1,1,1)$
    else
      $current \leftarrow (1,0,1)$
    for each edge $e$ of contour $c$ do
      $colour(e) \leftarrow current$
      if $current = (1,1,0)$ then
        $current \leftarrow (0,1,1)$
      else
        $current \leftarrow (1,1,0)$
```

Since the distance field is only an approximate representation, some colour configurations may yield better results than others.

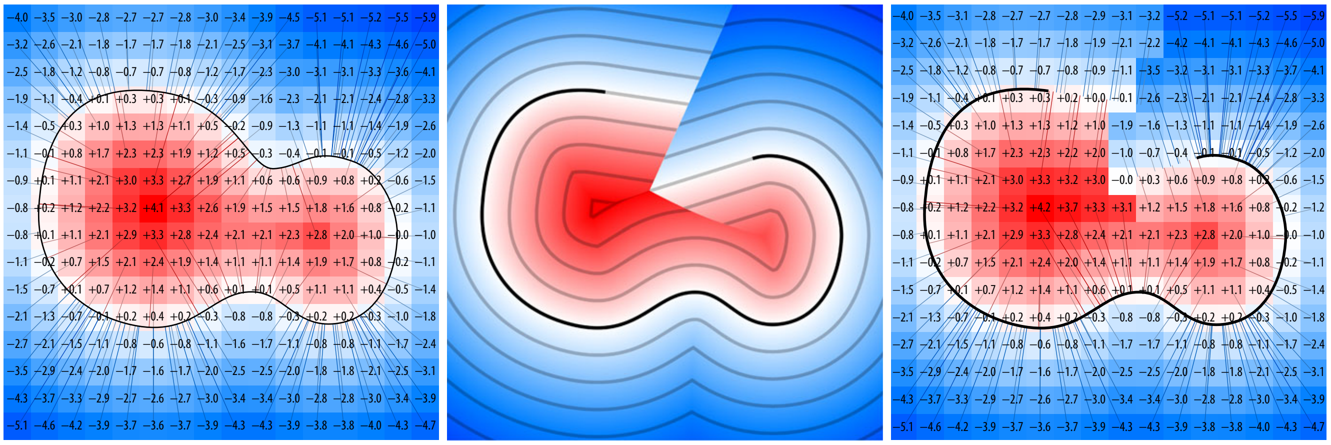


Figure 16: *Example of a (continuous) signed distance field values (left). A discontinuous distance function (middle) and a distance field generated from it (right).*

Inventing an optimal colouring strategy may be the objective of further research.

5.2. Distance field construction

For each of the three colour channels, a discontinuous pseudo-distance field must be constructed, considering only edge segments assigned to that channel. It may be advantageous to generate all three distance fields simultaneously, since at each point, the value will be the same in at least two of them.

Algorithm 2 Evaluation of a pixel of a corner preserving multi-channel distance field.

```
1:  function GeneratePixel($P$)
2:    $dRed \leftarrow \infty, dGreen \leftarrow \infty, dBlue \leftarrow \infty$
3:    for each contour $c$ of shape $s$ do
4:      for each edge $e$ of contour $c$ do
5:        $d \leftarrow$ EdgeSignedDistance($P, e$)
6:        if $colour(e) \cdot (1,0,0) \neq 0$ and Cmp($d, dRed$) $< 0$ then
7:          $dRed \leftarrow d, eRed \leftarrow e$
8:        if $colour(e) \cdot (0,1,0) \neq 0$ and Cmp($d, dGreen$) $< 0$ then
9:          $dGreen \leftarrow d, eGreen \leftarrow e$
10:       if $colour(e) \cdot (0,0,1) \neq 0$ and Cmp($d, dBlue$) $< 0$ then
11:         $dBlue \leftarrow d, eBlue \leftarrow e$
12:   $dRed \leftarrow$ EdgeSignedPseudoDistance($P, eRed$)
13:   $dGreen \leftarrow$ EdgeSignedPseudoDistance($P, eGreen$)
14:   $dBlue \leftarrow$ EdgeSignedPseudoDistance($P, eBlue$)
15:   return distanceToColour(($dRed, dGreen, dBlue$))
```

The signed distance at each point of the distance field shall be evaluated in the following way (see Algorithm 2). First, using the Voronoi tessellation described in Section 4.3, determine the closest edge segment to this point. One may use a spatial search algorithm for this task. Then, determine the closest point on that edge. If it is an endpoint, compute the signed pseudo-distance by finding the minimum distance to the infinite extension of the edge segment from that endpoint. Otherwise, use the true signed distance.

As described in Section 4.3, to determine which edge is the closest, the distance measure has to include two values—the actual distance, and a measure of orthogonality, the dot metric. All signed distance values in the presented algorithm hold both of these components, and the comparison function Cmp correctly takes both into account to determine which distance value is factually closer. EdgeSignedDistance function returns the correct signed distance from an edge segment (line, quadratic or cubic Bézier curve) in this format. Considering that cubic Bézier curves require solving a fifth-degree polynomial equation, we utilize Henrik Vestermark's implementation of the Jenkins–Traub algorithm [JT70], which is a globally convergent approximate solution method.

An example of the multi-channel distance field of the lower-case 'e' glyph is shown in Figure 18. The same method can be applied to construct a conventional signed pseudo-distance field as well.

5.3. Error correction

Unfortunately, certain rare scenarios may cause artefacts. Although the 'fronts' of discontinuity in the three distance fields never cross, they often converge towards one another, and at some point become too close due to the limited resolution of the distance field. In such a case, the area of the shape may transition from being represented, e.g. by channels R and G to channels R and B. Due to the imprecise bilinear interpolation of the distance field, there might arise a small area where neither G nor B is evaluated as filled, causing a small gap in the output image. This transition is illustrated in Figure 17.

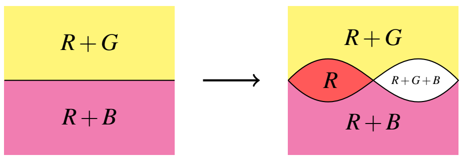


Figure 17: *The transition between areas represented by different pairs of channels (left) and a possible result after reconstruction (right).*

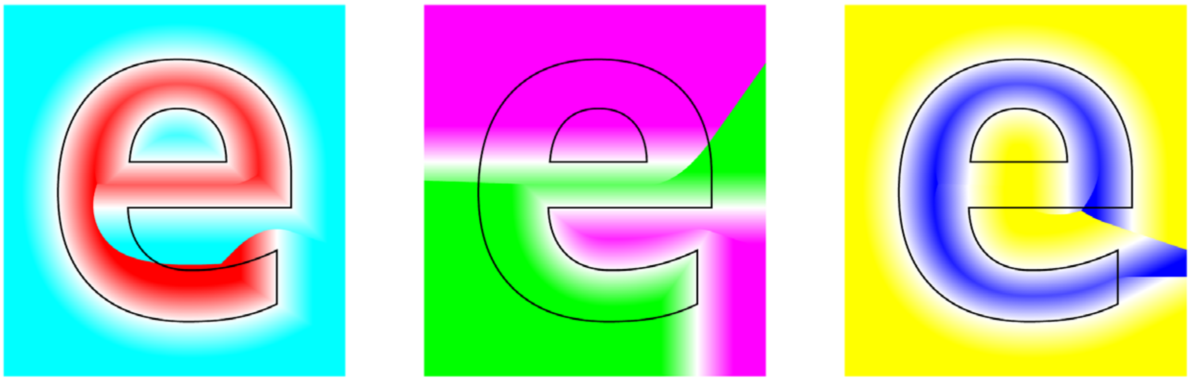


Figure 18: *The three channels of the multi-channel distance field of the lowercase 'e' glyph.*

Any such errors can be easily corrected though, as long as they are identified. The median function can be applied to any cell in the multi-channel distance field in advance, equalizing its three components to the median value, which corresponds to the actual pseudo-distance that would be found in a single-channel pseudo-distance field. Of course, this negates the improved fidelity of corners in that area, but more importantly any artefacts arising from this method.

To detect such potential errors, we have used the following procedure: Consider all pairs of four-connected neighbouring cells in the multi-channel distance field. If the difference in their value is greater than the distance between them in at least two channels, indicating two discontinuities in the area, flag the cell with the greater absolute distance value in the remaining channel—that one lies further away from any edges or corners. Afterwards, apply the median equalization on all flagged cells at once.

6. Shape Reconstruction and Rendering

The intended usage of this rendering method is in real-time graphics, where a graphics library such as OpenGL [Khr15] is commonly used. With these libraries, the SDF has to be loaded into a 2D texture, and its evaluation happens in a fragment shader. The graphics library is capable of performing the bilinear sampling routine by itself since it is a very common task in 3D graphics. This is one of the reasons why distance fields are a natural choice in this context.

We will look at the complete procedure of reconstructing a raster image of the original shape from an SDF. This is done by determining at each pixel of the raster image, whether or not it lies inside the shape. Let us assume that using the correct transformation, we have computed that the current pixel lies at point P in the coordinate system of the distance field's grid. Now, we need to sample a value from that position in the distance field utilizing bilinear sampling. Having sampled a value from the distance field, we can convert it back to a signed distance, using the inverse of the distanceToColour function:

$$\text{colourToDistance}(\text{distanceToColour}(x)) = x, \tag{16}$$

$$\text{colourToDistance}(colour) \overset{\text{def}}{=} \left(\frac{colour}{maxColour} - \frac{1}{2}\right) \cdot distanceRange. \tag{17}$$

With the median of three model, the reconstruction process is almost equally simple, with only one additional step. In this case, the result of the sampling would not be a scalar value, but a vector, where each component is the sample of one colour channel. Immediately after acquiring this vector, its median component can be extracted and used the same way as the scalar value in the single-channel distance field.

Instead of always using either the inside colour or the outside colour, we could smoothly blend from one to the other at the edge, thereby eliminating hard pixelated edges and achieving anti-aliasing. For this, a threshold value t should be chosen so that the interval $\langle -t, t\rangle$ in signed distance units is about as wide as a single pixel in the target bitmap. In a 3D scene, the scale of the distance field in the resulting image might not be constant throughout, and therefore the threshold t cannot be determined globally before rendering. Instead, it has to be computed separately for each pixel or fragment. Fortunately, the OpenGL shading language (GLSL) offers the `fwidth` function for this task, which returns the amount of change of a variable between adjacent pixels.

Assuming that the correct position P has been computed in the vertex shader, a complete OpenGL fragment shader is available in Listing 1 which renders a shape encoded by a multi-channel distance field stored in a texture and applies anti-aliasing. The shader can be easily modified or expanded in many ways, for example, to make some parameters adjustable, or to support the visual effects mentioned further down.

The shader is also backwards compatible with single-channel distance fields, where all three components of `s` would be equal, and although the call to the median function would be unnecessary, the result would be the same. This again shows that one simple median calculation is the only additional operation required in our multi-channel distance field rendering method.

Visual effects By using a different transformation from the signed distance d to the pixel colour, a number of different visual effects

```
in vec2 P;
uniform sampler2D msdf;
uniform vec2 msdfUnit;
uniform vec4 outsideColour;
uniform vec4 insideColour;
out vec4 fragColour;

// Median value using minima and maxima
float median(float a, float b, float c) {
  return max(min(a, b), min(max(a, b), c));
}

void main() {
  // Bilinear sampling of the distance field
  vec3 s = texture(msdf, P).rgb;
  // Acquire the signed distance
  float d = median(s.r, s.g, s.b) - 0.5;
  // Convert the distance to screen pixels
  d *= dot(msdfUnit, 0.5/fwidth(P));
  // The anti-aliased measure of how "inside" the
      fragment lies
  float w = clamp(d+0.5, 0.0, 1.0);
  // Combine the two colours
 fragColour = mix(outsideColour, insideColour, w);
}
```

Listing 1: *Example fragment shader for multi-channel distance field evaluation.*

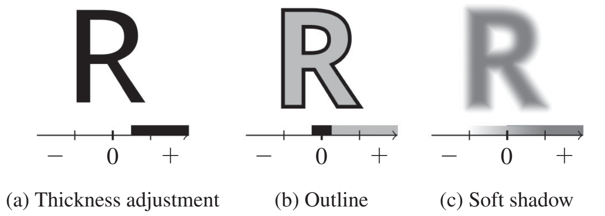


Figure 19: *Examples of distance-based visual effects.*

can be achieved. Some examples of such transformations are shown in Figure 19. The used transformation from signed distance to colour can be seen at the bottom.

Text rendering So far, we have covered how to reconstruct the image of a single shape from its distance field representation. To display text, however, the images of individual glyphs have to be composed together.

In real-time graphics, the distance fields of glyphs of the entire character set will be typically compiled into a single texture. Each glyph will be located at a known position in this texture. Examples of such textures are shown in Figure 20. Note that much of the space in these textures is empty due to a regular but inefficient distribution of the glyphs. To utilize the maximum possible available space of the texture, a 2D bin packing algorithm may be used.

A text can be rendered as a sequence of rectangles whose vertices are correctly positioned and mapped to glyph positions inside the texture. All of the necessary metrics can be retrieved from the definition of the font. Figure 21 shows an example application of this principle in the form of a triangle mesh.

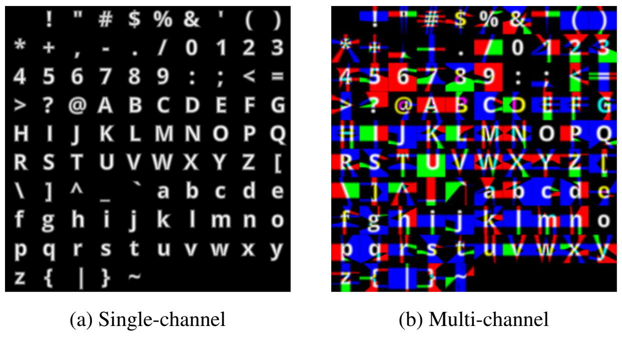


Figure 20: *SDF textures, from which individual glyphs can be mapped.*

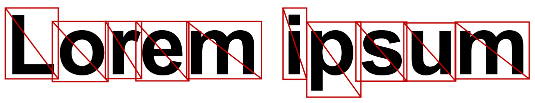


Figure 21: *A string of text as a textured triangle mesh.*

7. Results

We provide a multi-channel SDF generator implementation based on the proposed approach in the form of an open-source GitHub project [Chl15]. In this section, we evaluate the outputs of the devised algorithm and measure the factual improvement in quality of rendering using our multi-channel distance field technique, and compare the results with the original single-channel version. We also discuss the performance of the algorithm, namely the cost of using its outputs to render images and the artefacts it may produce.

7.1. Rendering quality

We have tested our multi-channel distance fields on several widely used typefaces. Below are presented results for Open Sans, a typical sans-serif typeface, the regular weight, non-italic variant. First, we have rendered the entire ASCII character set into ultra-high-resolution bitmaps using the original distance field method, and our multi-channel method, and counted the portion of pixels that do not match the exact image of each glyph. We have done this using different distance field resolutions, ranging from 16×16 to 48×48 per glyph. The average results are listed in Table 3.

However, when we only look at characters that do not include any curves, the difference becomes much greater. The results for those characters only are listed in Table 4. Figure 22 demonstrates the range of distance field resolutions we used in our tests. The shape in the leftmost image (resolution 7×8 pixels) is already disintegrating, and therefore going any lower would be pointless. The rightmost image (43×50 pixels), on the other hand, seems to possess a satisfying precision, apart from the sharpness of corners.

Table 3: *A comparison of average rendering error of the original and our technique for the ASCII character set glyphs.*

	Average error	
SDF size	Original	Multi-channel
16	$7.55 \cdot 10^{-3} \pm 3.1 \cdot 10^{-3}$	$3.25 \cdot 10^{-3} \pm 2.6 \cdot 10^{-3}$
24	$3.22 \cdot 10^{-3} \pm 1.2 \cdot 10^{-3}$	$8.57 \cdot 10^{-4} \pm 8.8 \cdot 10^{-4}$
32	$1.71 \cdot 10^{-3} \pm 6.4 \cdot 10^{-4}$	$4.65 \cdot 10^{-4} \pm 4.8 \cdot 10^{-4}$
48	$7.41 \cdot 10^{-4} \pm 3.0 \cdot 10^{-4}$	$2.04 \cdot 10^{-4} \pm 2.1 \cdot 10^{-4}$

Table 4: *A comparison of average rendering error for non-curved glyphs only from the ASCII character set.*

	Average error	
SDF size	Original	Multi-channel
16	$7.24 \cdot 10^{-3} \pm 4.1 \cdot 10^{-3}$	$1.45 \cdot 10^{-3} \pm 2.0 \cdot 10^{-3}$
24	$3.22 \cdot 10^{-3} \pm 1.5 \cdot 10^{-3}$	$2.43 \cdot 10^{-5} \pm 5.6 \cdot 10^{-5}$
32	$1.71 \cdot 10^{-3} \pm 7.8 \cdot 10^{-4}$	$7.63 \cdot 10^{-6} \pm 2.8 \cdot 10^{-5}$
48	$7.09 \cdot 10^{-4} \pm 3.8 \cdot 10^{-4}$	$3.80 \cdot 10^{-7} \pm 1.3 \cdot 10^{-6}$

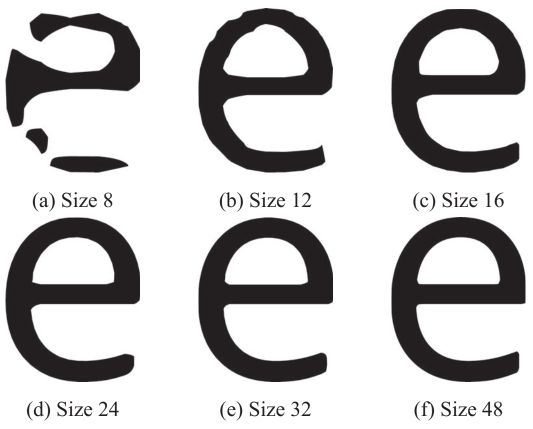


Figure 22: *Reconstruction of the letter 'e' from a single-channel pseudo-distance field of varying resolutions.*

For non-curved glyphs, the error is several orders of magnitude lower with our method, meaning that in the previous case, most of the error has accumulated from an imprecise reconstruction of parametric curves, at which our method performs the same as the original. However, this has far lower impact on image quality than the appearance of corners, since a slightly different appearance of a curve is indistinguishable to the human eye. Therefore, we will observe another property of the rendered images—*the weighted error*.

The weighted error, listed in Table 5, should serve as a better indicator of subjective quality. In it, errors further away from the actual edge of the glyph have greater weight, directly proportional to this distance. This means that shifting a large portion of the outline slightly will have a lesser impact on the result than a short portion straying greatly. A visual comparison of the rendering quality can be seen in Figure 1 on the title page and in Figure 24.

Table 5: *A comparison of weighted rendering error for all glyphs from the ASCII character set.*

	Average weighted error	
SDF size	Original	Multi-channel
16	$4.62 \cdot 10^{-2} \pm 2.5 \cdot 10^{-2}$	$1.26 \cdot 10^{-2} \pm 1.8 \cdot 10^{-2}$
24	$1.39 \cdot 10^{-2} \pm 8.0 \cdot 10^{-3}$	$6.52 \cdot 10^{-4} \pm 9.7 \cdot 10^{-4}$
32	$5.22 \cdot 10^{-3} \pm 3.2 \cdot 10^{-3}$	$1.96 \cdot 10^{-4} \pm 2.9 \cdot 10^{-4}$
48	$1.55 \cdot 10^{-3} \pm 1.0 \cdot 10^{-3}$	$3.70 \cdot 10^{-5} \pm 6.0 \cdot 10^{-5}$

Table 6: *Comparison of the error in sampled distance values throughout the entire plane.*

	Average weighted distance difference	
SDF size	Original	Multi-channel
12	2.180 ± 0.607	1.434 ± 0.654
16	1.281 ± 0.320	0.7575 ± 0.337
24	0.6016 ± 0.163	0.3290 ± 0.157
32	0.3198 ± 0.0864	0.1699 ± 0.0840
48	0.1447 ± 0.0400	0.07638 ± 0.0374

Preservation of distance information To evaluate the distance field's precision in other areas than just around the outline, which is important for visual effects that use signed distance values, we use the weighted distance difference metric that measures the difference in reported and actual signed distance. Since the distance values are less likely needed further away from the outline, the difference will be given a greater weight the closer to the outline it is. The value is computed as

$$\sum_P |d - d_S| \mathrm{e}^{-\frac{|d|}{k}}, \tag{18}$$

where P are the sampled points, d is the correct signed distance, d_S is the signed distance sampled from the distance field and k is an adjustable weight distribution parameter, which we set to 60.

The measured data in Table 6 show that the multi-channel method encodes the signed distance with approximately half the original error. The signed pseudo-distance reconstruction of a single glyph in the form of equidistant contour lines is shown in Figure 23. At the outline and in its vicinity, the distance is represented accurately by our multi-channel technique. It even preserves the sharp corners further outside the shape but gradually loses precision.

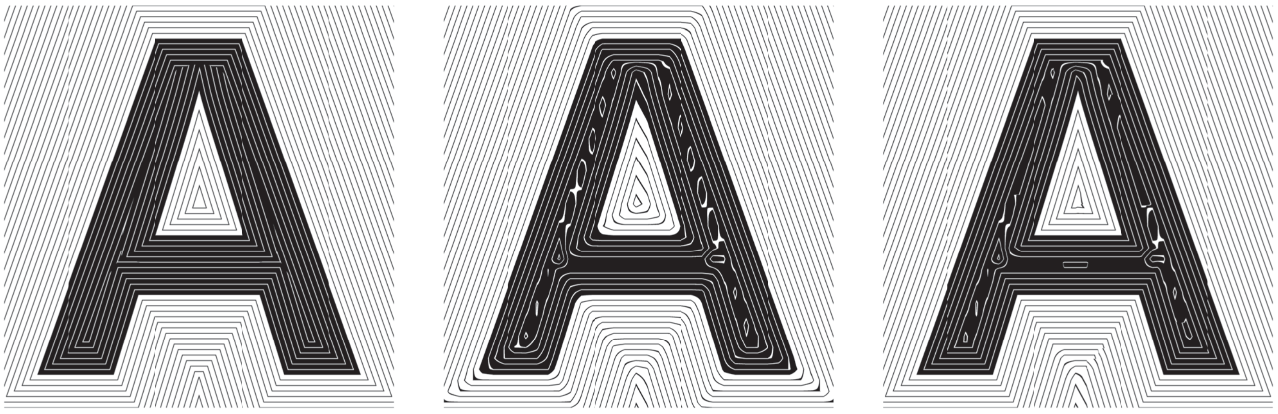


Figure 23: *Contour diagram of the letter 'A' constructed exactly (a), and reconstructed from distance fields by the original (b) and our multi-channel method (c).*

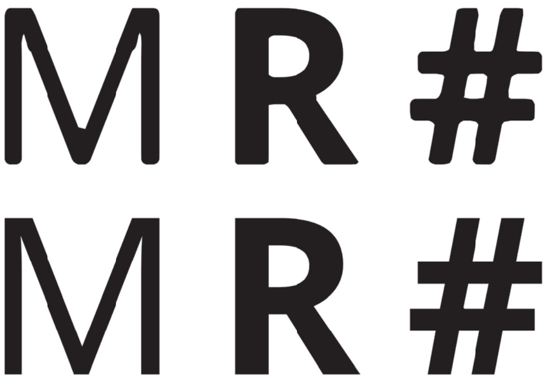


Figure 24: *Comparison of the reconstruction of several glyphs of varying thickness using the original (top) and our multi-channel distance field method (bottom).*

Table 7: *Comparison of text rendering frame rates when using single-channel and multi-channel distance field textures.*

	Distance field dimensions			
Distance field type	256	512	1024	2048
Single channel (R)	30.81	28.60	26.44	21.02
Single channel (RGB)	29.08	26.20	20.15	9.18
Multi-channel	27.18	24.76	19.56	9.16
Single channel (R) + AA	28.43	26.33	24.22	18.98
Single channel (RGB) + AA	26.65	23.98	18.43	8.42
Multi-channel + AA	25.05	22.86	17.96	8.41

7.2. Real-time rendering performance

Since rendering performance is the original distance field technique's primary advantage, it is extremely important that this advantage is not lost. To test this, we have created an OpenGL program, where an enormous amount of text is drawn on the screen from a distance field. We have designed it so that as little extra time as possible is spent on other tasks than the rendering itself. The text is drawn in batches of about 32 characters per draw call, and no uniform variables or other states are changed in between, only the vertex array object. The vertex shader only performs a single matrix transformation of the coordinates and for the fragment shader, variants of the implementation from Section 6 have been used.

The measured frame rates (frames per second) are shown in Table 7. Since the measurements have been performed on a specific machine (NVIDIA GTX760), only the differences are of importance. The test has been performed with both single-channel and multi-channel distance fields in several resolutions, and with and without anti-aliasing (AA). Furthermore, the single-channel variant has been tested storing the texture both as only one channel (R), and as a regular 3-channel image (RGB).

It is evident that the performance impact of using a multi-channel distance field varies by the size of the distance field texture. For small resolutions, the decrease in frame rate is only about 12%. For higher resolutions, it seems that the sampling of the texture is a significantly slower operation, and interestingly, a very noticeable difference can be seen just in sampling a monochrome (R) versus a colour (RGB) texture. The cost of the additional median computation is negligible in these cases, but the increase in the number of texture channels brings up to a 56% drop in the performance. The anti-aliasing routine seems like a relatively inexpensive addition to the rendering process, causing a slowdown of 8–10%.

In conclusion, the performance of the multi-channel distance field rendering method is approximately 12–56% worse than that of single-channel distance field rendering depending on the distance field resolution. An interesting observation is that since

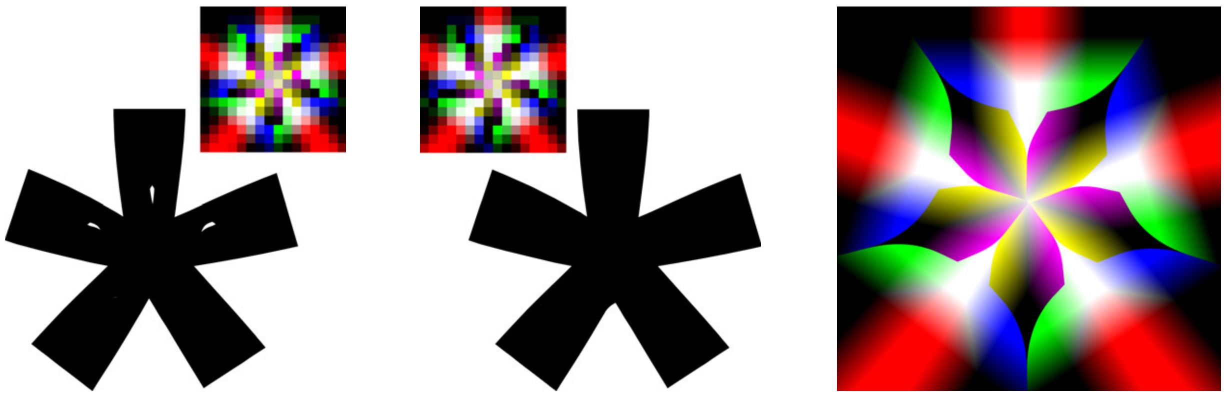


Figure 25: *Comparison of the reconstruction of the asterisk glyph from MSDF of resolution* 16×16 *with and without the application of the described error correction.*

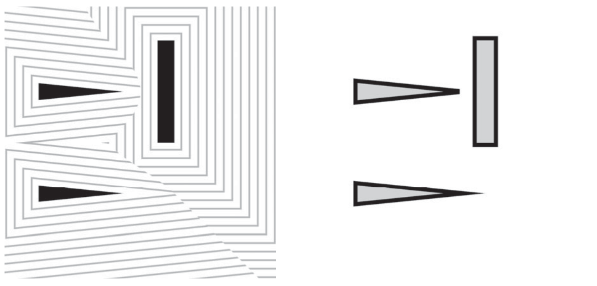


Figure 26: *Contour discontinuities and resulting artefact in reconstructed shape stroke.*

texture sampling is the most expensive operation, the original method is almost just as slow as the improved one when the single-channel distance field is needlessly stored as and sampled from a three-channel texture.

8. Limitations of the Method

The error correction method described in Section 5.3 is not perfect but has been able to eliminate a vast majority of the artefacts we have encountered, an example of which is illustrated in Figure 25. It can be observed that while the correction has repaired the holes within the glyph, it has worsened the appearance of the lower corner as a side effect, which happens since the error correction locally degrades the multi-channel SDF to a plain SDF. If the distance field has a sufficient resolution, however, the areas where the errors occur tend to lie away from any corners. Figure 25(c) captures the distance field at a higher resolution, which reveals the origin of the errors—the sharp interfaces between yellow and magenta colours, where the pairs of channels that represent the inside of the glyph alternate. Similarly, the potential risk also exists at the green and blue interfaces in the outer part, where surplus islands could form, but this was not the case.

We have shown that our method provides a good signed distance approximation at any point in the distance field, making it

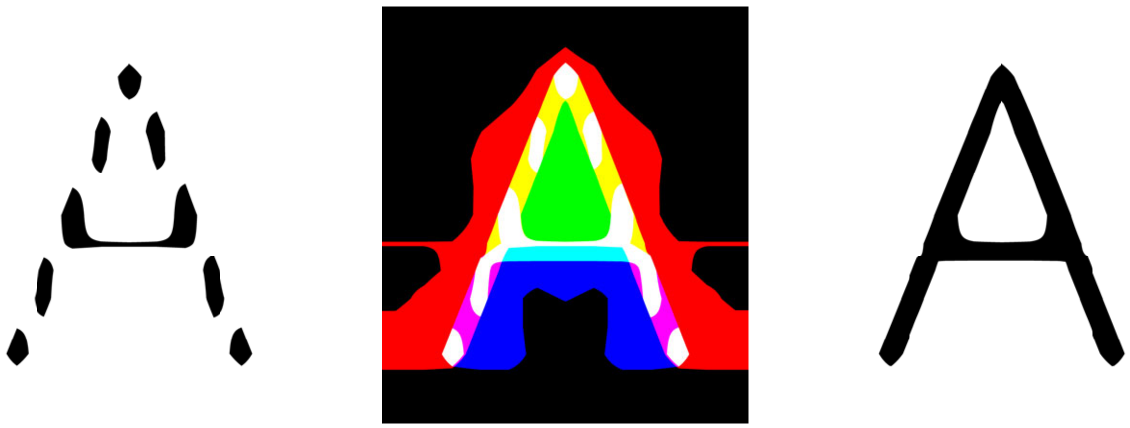


Figure 27: *Reconstruction of the glyph 'A' with thin strokes using specialized distance fields.*

suitable for rendering additional visual effects. Nevertheless, the algorithm, as presented, has at least one known issue with correctly determining distance from sharp mitres. This is demonstrated in Figure 26, where you can see the mitre being cut-off at a certain distance, due to the pseudo-distance vastly diverging from the Euclidean distance, which determines the closest segment. We believe that the algorithm could be adjusted to account for such cases.

9. Conclusions

We have presented an improvement to the state-of-the-art technique for rendering text and other 2D shapes. By utilizing three distance field channels instead of just one, we have significantly improved the precision of rendering sharp corners, and reduced the rendering error for these features by up to several orders of magnitude.

Future work may involve inventing an optimal colouring strategy that may yield better results for some colour configurations and improving the error correction method, which is not perfect but has been able to eliminate a vast majority of the artefacts we have encountered.

Thickening decomposition There are other disadvantages of the distance field rendering technique that could be solved by a multi-channel decomposition. One of them is the representation of thin features. If a stroke is thinner than about two cells of the distance field grid across, it cannot be encoded properly and will probably be heavily distorted if at all visible in the reconstructed image (see Figure 27a). However, if the thin feature were to be represented as a union of two much thicker strokes (one on each side) encoded in multiple channels, the issue could be resolved. This would reduce the minimum required resolution of the distance field to properly encode thinner fonts.

Figure 27 shows an example of this idea. On the left, you can see the result of using a low-resolution single-channel distance field on a thin typeface. The centre image demonstrates the reconstruction of the image using a thickening multi-channel decomposition. In addition to the white parts, which represent areas where all three colour components are evaluated as inside, we can also use the yellow, cyan, and magenta areas, where only two components are. The final result after applying the median function is seen in Figure 27(c). Even though it is not perfect, it certainly makes the character readable.

Although we have primarily used the median of three colouring model, we theorize that a higher number of channels could be utilized for an even higher reconstruction precision. One possibility would be the combination of corner preservation and thickening.

Acknowledgements

We thank Ondřej Jamriška for many insightful remarks that improve the clarity of the presentation as well as the reviewers for their valuable comments. This research has been supported by CTU internal grant SGS17/215/OHK3/3T/18.

References

[Ado06] Adobe Systems Incorporated: Font formats. https://www.adobe.com/type/browser/info/formats.html, 2006.

[AK00] Aurenhammer F., Klein R.: Voronoi diagrams. In *Handbook of Computational Geometry*. J.-R. Sack and J. Urrutia (Eds.). North-Holland, Amsterdam, (2000), pp. 201–290.

[Chl15] Chlumský V.: Multi-channel signed distance field generator. online, 2015. URL: https://github.com/Chlumsky/msdfgen.

[Con00] Constable P.: Understanding multilingual software on MS Windows: The answer to the ultimate question of fonts, keyboards and everything. https://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&id=IWS-Chapter08, 2000. Available in CTC Resource Collection 2000 CD-ROM, by SIL International.

[dBvKOS08] de Berg M., Cheong O., van Kreveld M., Overmar M.: *Computational Geometry*: Algorithms and Applications (3rd edition). Springer-Verlag Berlin Heidelberg, 2008.

[Esf14] Esfahbod B.: GLyphy – high-quality glyph rendering using OpenGL ES2 shading language. https://glyphy.org, 2014.

[Gre07] Green C.: Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 Courses* (2007), ACM, pp. 9–18.

[Gro11] W3C SVG Working Group: *Scalable Vector Graphics (SVG)* 1.1. (2nd edition). https://www.w3.org/TR/SVG11/, 2011.

[JT70] Jenkins M. A., Traub J. F.: A three-stage algorithm for real polynomials using quadratic iteration. *SIAM Journal on Numerical Analysis 7*, 4 (1970), 545–566.

[KB12] Kilgard M. J., Bolz J.: GPU-accelerated path rendering. *ACM Transactions on Graphics 31*, 6 (November 2012), 172:1–172:10.

[Khr15] Khronos Group: OpenGL overview. https://www.opengl.org/about/, May 2015.

[LB05] Loop C., Blinn J.: Resolution independent curve rendering using programmable graphics hardware. *ACM Transactions on Graphics (TOG) 24* (2005), 1000–1009.

[LB07] Loop C., Blinn J.: Rendering vector art on the GPU. *GPU Gems 3* (2007), 543–562.

[Pat14] Patai G.: Playing around with distance field font rendering. https://lambdacube3d.wordpress.com/2014/11/12/playing-around-with-font-rendering/, November 2014.

[QMK06] Qin Z., McCool M. D., Kaplan C. S.: Real-time texture-mapped vector glyphs. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (2006), ACM, pp. 125–132.

[RCL05] Ray N., Cavin X., Lévy B.: *Vector Texture Maps on the GPU*. Tech. Rep. ALICE-TR-05-003, INRIA Research Centre Nancy - Grand Est, 2005.

[SCH03] Sen P., Cammarano M., Hanrahan P.: Shadow silhouette maps. *ACM Transactions on Graphics 22*, 3 (July 2003), 521–526.

[tC15] ten Cate T.: Distance field fonts. https://github.com/libgdx/libgdx/wiki/Distance-field-fonts/, December 2015.

[YKB*14] Yoo J.-J., Krishnadasan S., Brothers J., Jung S., Ryu S., Kim J.: Path rendering for high resolution mobile device. In *SIGGRAPH Asia 2014 Mobile Graphics and Interactive Applications* (2014), ACM, pp. 13:1–13:5.