Multi-Colour Distance Fields

Andrew W. Hlynka
School of Computer Science, Carleton University
Ottawa, Ontario, Canada
andrewhlynka@cmail.carleton.ca

David Mould
School of Computer Science, Carleton University
Ottawa, Ontario, Canada
mould@scs.carleton.ca

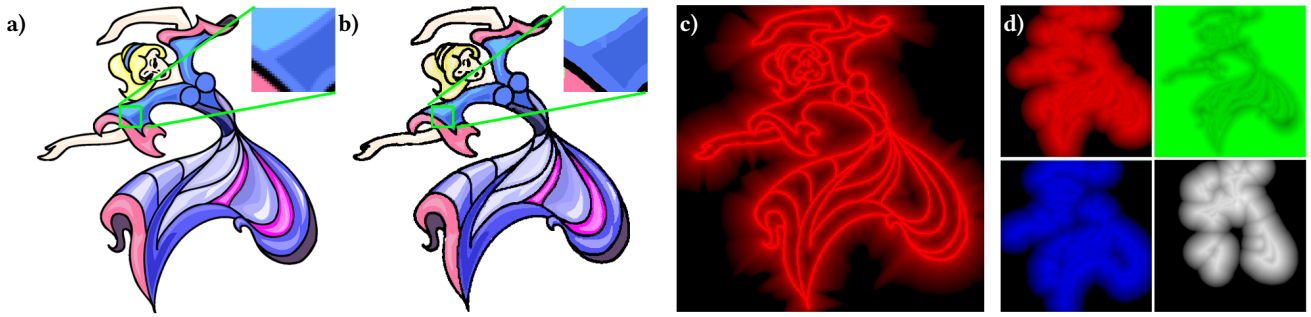

Figure 1: From left to right: a) a 512x512 input image of a dancer, b) a render from a 256x256 multi-channel Distance Field using our method, c) the single channel representing our approach for the outlines, and d) four channels representing neighbouring coloured regions in our method.

Abstract

Distance field rendering is an effective technique to render low-resolution vector-like raster assets with sharp detail. In 2D, this technique is commonly used to render simplified mono-colour elements such as text or glyphs alongside 3D objects. We extend signed distance fields for improved rendering of crisp outlines and rendering interior multi-colour regions. Outlines are rendered through a nonlinear distance metric that prevents detail loss during downsampling. Interior regions are represented through a fixed number of four distance field channels, plus four channels to store colour indices, inspired by the four-colour mapping problem. Our method enables cartoon-like sprites, images, and icons to be rendered at high quality with reduced storage and memory requirements for real-time rendering.

CCS Concepts

• **Computing methodologies** → *Computer graphics*; *Rasterization*; *Non-photorealistic rendering*; *Image processing*; **Image-based rendering**; *Image compression*.

Keywords

Distance Fields, Real-Time Rendering, Image-Based Rendering

GI '26, Kitchener-Waterloo, ON

ACM ISBN 978-1-4503-XXXX-X/2026/06
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Andrew W. Hlynka and David Mould. 2026. Multi-Colour Distance Fields. In *Proceedings of Graphics Interface (GI '26).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Storage and memory resources to run interactive media has grown rapidly in recent years. Traditionally, sharp detail could be achieved with high-resolution textures and sprites, but as support for 4K (and above) resolution screens became common, so too did larger image assets. Modern computer games can use over 100 GB of storage space, more than half of which is commonly used for texture assets [6]. Contrary to intuition, this does not apply only to photo-realistic textures; this issue can plague simple, non-photorealistic textures and sprites, and especially sprite sheets with frame-by-frame animation. Vector assets could be used in place of raster (a.k.a. bitmap) assets for low-detail graphics, but as of this writing, mainstream game engines do not natively support vector formats. Runtime performance on high-complexity (or large numbers of) vector assets is also a concern for real-time rendering.

To support crisp textures and sprites in raster-based game engines, alternative formats can be used. Signed Distance Fields (SDFs) are an implicit structure that captures the shape of an object in a rasterized gradient. Valve's developers popularized the use of SDFs in games, rendering sharp text along 3D walls in *Team Fortress 2*. SDFs enabled them to render smooth curves and to apply some effects, such as outlines and drop shadows [10].

While now widespread for rendering fonts and glyphs, distance fields for general 2D rendering remains limited and has yet to be explored for more complex sprites and textures. Raster remains the standard for general 2D game assets.

Our proposed approach focuses on distance fields for sprites and textures with simplified details, specifically cartoon-like images.

Prior work only uses distance fields to render simple 2D shapes of a single colour; our method supports multiple colours for the same asset. Additionally, traditional SDFs retain no information on line thickness, which can be an issue when line thickness varies within the same image; our method addresses this with an alternative distance metric that both represents line thickness and preserves its details better at different resolutions. These two solutions (colour and outlines) are accomplished separately with multiple distance fields, stored in different channels of RGBA-based image assets.

In this paper, we introduce an enhanced Distance Field method for rendering cartoon sprites. We make the following contributions:

- "Relative Signed Distance Field" (RelSDF): a distance metric that represents lines of varying thickness within the same image, and maintains relative thickness when rendering parameters are adjusted for the distance field. This metric also better supports thin lines (as little as 1 texel wide at downsampled resolutions).
- "Regional Signed Distance Field" (RegSDF): provides crisp and smooth boundaries between interior colour-filled regions, through the use of multiple distance fields.

2 Related Work

This section briefly discusses past work on distance fields and relevant alternative solutions for real-time 2D and 3D rendering.

2.1 Distance Fields

This work was partly inspired by online projects that used classic distance fields for outlines on simple 2D cartoon sprites [14, 20], referencing an influential talk by Valve from 2007 [10]. Dating back as far as 1966 [27], distance fields can be used for both 2D and 3D representations. In 2D, a common application is to render low-resolution text font with crisp boundaries, and in 3D, distance fields can similarly be used as an implicit representation of 3D shapes. In addition to representing the boundaries of the shape, signed distance fields use positive and negative sign to indicate the inside and outside of the shape. If rendered with a single colour, then only a single greyscale distance field asset is required at runtime, and past work showed sizes as little as 128 x 128 to be effective [10].

Challenges remain due to limitations with distance fields. Distance fields are frequently shown for rendering mono-colour 2D shapes, but rendering multi-coloured images has not yet been explored. Distance fields struggle to render sharp corners, but workaround solutions have been proven to work, such as storing additional annotation data and adapting shader logic to utilize it [10]. Further shortcomings include edge placement not being perfectly accurate, and lines or shapes less than two texels wide becoming lost in SDFs [11]; we address the latter problem with RelSDF.

In their survey, Frisken and Perry [8] focused on the capabilities of distance fields for interactive design with both 2D and 3D examples, noting that they are well-suited for boolean operations. They acknowledged that the distance measurement in a distance field is a scalar function commonly set as, but not limited to, Euclidean distance. We refer to an analysis by Biswas and Shapiro for further discussion on alternative metrics [5], but in short, not limiting ourselves to Euclidean distance allows us to address the challenge of relative line thickness with RelSDF.

Our motivation to handle multiple colours is relevant to 2D distance fields, and less so for 3D fields. Implicit shape representation is still an actively researched application for 3D distance fields. We refer to Jones et al. [16] for a more in-depth survey of classic techniques still used today, including discussion of formats, computation methods and applications, with special attention on Distance Transforms and the Fast Marching Method.

Frisken et al. introduced Adaptively Sampled Distance Fields [9] to retain small details through adaptive resolution in segments where required. Related work to improve both detail and storage space includes utilization of octrees [9, 17], first-order derivatives [3], or differentiable systems [15, 31]. This level of optimization is outside the scope of this paper.

Recent work exploited distance field data both near and far from a boundary for a more accurate construction (contouring) of the implicit surface [18]. Some work on 2D distance fields [3], was extended by Horvath et al. [12] to 3D and for general surfaces, embedding first-order data within the distance field to recreate details with lower-resolution fields. In our work, we use relatively high-resolution fields to retain detail, and this existing work could be a helpful follow-up. Aumentado-Armstrong et al. [2] proposed Directed Distance Fields that store direction as well as surface position to more efficiently access surface data.

Adjacent to the concept of distance fields, the Walk-On Spheres sampling method uses spheres to estimate the surface boundaries of a shape by distance, whose behavior can be used for other problem sets, such as simulating heat transfer [21, 24]. Recent studies explored 3D shape representations and renderings with neural networks, using distance fields to improve results [19], as input for training [4], or to represent level of detail [35].

Chlumský explored using multiple SDFs (stored as RGB channels) to address the problem of sharp corner rendering [7]. This could be used alongside the channels we introduce with RelSDF and RegSDF.

2.2 Other Implicit Representations

Another strategy to maintain crisp rendering with low-resolution input is to use Silhouette Maps [29], which uses a second 2D asset storing relative coordinates for each pixel cell, describing boundaries from the original RGB image. With this, the renderer can draw lines without being restricted to a uniform grid appearance. Originally meant for sharp shadows [29], further work extended it for general texture rendering [28]. Silhouette maps were shown to be best suited for decals with solid colours rather than photorealistic textures, and defect artifacts are possible. While more complex than distance fields to implement and manipulate for supplementary effects, the textures they render are similar to our target class of images – silhouette maps could be a viable alternative.

Diffusion Curves are a vector-based representation that embeds gradient-fill data with each curve, allowing complex smooth-shading effects [22]. Elsewhere, the properties of distance fields, especially with the distance function we use, share some similarities to level sets and the speed function for implicit modeling [23].

The goals of this paper is to reduce memory requirements while providing rendering quality comparable to high-resolution 2D assets. Vector-based graphics formats are a natural use-case for this purpose. Vector tools have become increasingly prevalent for static

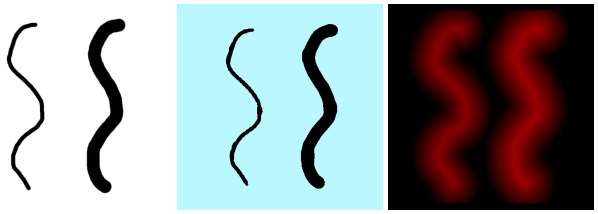

Figure 2: Left: the original input image (512x512). Centre: rendering from SDF. Right: the SDF (64x64).

art and offline-render animation. However, as of this writing, off-the-shelf 3D game engines do not natively support vector assets at runtime [30]. Different solutions have been proposed for introducing vector graphics into a raster-based pipeline, such as storing curve data inside raster-based textures [30], or mapping vector textures to 3D surfaces [25, 26]. But for wider support and to avoid reliance on third-party suites, adapting to the common raster-based rendering pipeline is the path of least resistance. We also appreciate the control raster assets allow on memory and processing requirements by adapting resolution, and easier debugging and editing, which remain critical for game production.

3 Our Method

Section 3.1 describes a basic implementation of signed and non-signed distance fields, which we use as a basis for our methods, and also test against in Section 4. Sections 3.2 and 3.3 present our two contributions: Relative Signed-Distance Fields (RelSDF) and Regional Signed-Distance Fields (RegSDF).

3.1 Conventional Distance Field Encoding

We implement an SDF as a discretized uniform grid of texels, each texel storing a numeric value representing the distance from a target shape. In a raster format, this can be stored as a single unsigned channel in a mono or mutli-channel (e.g. RGBA) texture. Following Green [10], we normalize the values between 0.0 and 1.0, with 0.5 representing the shape boundary, values above 0.5 representing the shape's interior, and below 0.5 the shape's exterior.

At the render stage, the distance field raster channel is passed to a shader. Similar to a basic alpha cutoff shader [13], a distance field shader can clip values below a threshold, and render a solid colour on what remains. Both the threshold value and colour are adjustable parameters.

A non-signed distance field relies on knowing the centre of a shape. In our case, after extracting outlines from an image, we skeletonize them with the Zhang-Suen thinning algorithm [34]. In the field channel, the algorithm iterates outwards from line centres to set Euclidean distance to the nearest outline pixel. Distance values are coded with normalized values between [0.0, 1.0], with 0.0 being farthest from the centre. The field is first computed at the full resolution of the input image, then downsampled to a smaller resolution for use. For an SDF, the skeletonization step is not used, but unlike an unsigned distance field, the original thickness of shapes can be maintained during rendering (if using a cutoff value of 0.5). An example visual can be seen in Fig. 2.

3.2 Relative Line Thickness with RelSDF

For an SDF rendered with a cutoff of 0.5, the original shape is successfully reproduced. However, thin shapes (1 texel wide) with no interior pixels above 0.5 are lost. Additionally, supplemental effects are typically accomplished by varying the cutoff value. Changing the cutoff will change the thickness of the shape at a linear rate, but not relative to the thickness of different shapes in the image, which may or may not be desirable. Consider the example in Fig. 2: as the cutoff approaches zero, the two lines appear to have a similar thickness, and as the cutoff approaches 1.0, the thin line disappears completely, well before the thicker line does. By comparison, the non-signed distance field described in Section 3.1 does not have this issue as the cutoff approaches 1.0, but stores no information on the thickness or boundaries of the original shape.

To retain information on both line thickness and line centres, we define the Relative Signed Distance Field (RelSDF). Here, the distance function is mapped to an exponential curve rather than a linear curve (e.g. a line of thickness 2 would render approximately twice as thick as a line of thickness 1 at all cutoff values). When normalized, we want the value 0.5 for the exterior edges of our outlines, but also want a value close to 1.0 for the centre of all outlines. In short, our requirements are 1) n = 1.0 when f => 0, 2) n = 0.5 when f => line thickness, and 3) n = 0.0 when f => $+\infty$. One such formula that fits these requirements is

$$f = w * b^g \tag{1}$$
$$g = r * (0.5 - n)$$

Or specifically to find n:

$$n = -(log(f/w)/log(b) - 5)/r \tag{2}$$

Let f be the distance of a given pixel to the nearest outline centre, w be the width of an outline along that distance, and n be the new value to store at each texel in the SDF. We set b to 2, and r to 10. This formula was determined by intuition, and other function definitions that fit our desired requirements may be possible - the key to this feature are those requirements, not the function itself.

To calculate the RelSDF with this formula, the center of each line must be determined (see Section 3.1) to determine the thickness of the line at a given point. Thickness is determined by expanding a radius from a line's skeletonized point until an exterior pixel is found, and storing Euclidean distance from the line center to the nearest exterior point. Every texel in the field has a reference to its nearest line centre and its thickness, providing all required parameters to calculate n. Example results can be seen in Fig. 3, where line thicknesses appear appropriate at varying cutoff values.

As shown in Fig. 3, the design of RelSDF results in boundaries within the field when each texel is associated with a different outline, and therefore has a different gradient. Visual issues can occur when outlines of distinct thicknesses intersect and gradients conflict. Further discussion related to the RelSDF distance function and conflicting gradients can be seen in the Supplementary Materials.

3.3 Sharp Interior Colour Regions with RegSDF

At this point, a single colour channel representing an SDF for outlines can produce crisp rendered output, provided the outline separates all interior coloured regions. Otherwise, if interior colors

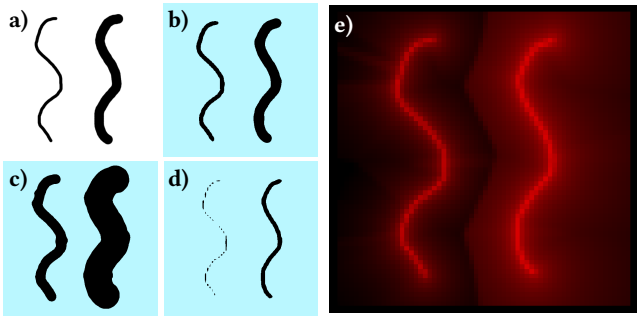

Figure 3: An example output at different cutoff values with RelSDF: a) The original input image (512x512), and RelSDF renders with b) cutoff 0.5, c) cutoff 0.35, d) cutoff 0.65, and e) the RelSDF (64x64).

Figure 4: Left: The original image with outlines extracted and removed (downsampled to 64x64). Right: the render with RegSDF (64x64).

vary little, bilinear filtering can be effective. However, many images do not fit these criteria. Distance fields can be used to produce sharp, smooth boundaries between interior colour regions. But distance fields portray shape, not colour. A naïve solution for multiple colours would be a separate distance field for each colour, possibly for thousands of colours in a single image. Our goal here is to limit the number of distance fields required for this purpose, which we call a Regional Signed Distance Field (RegSDF).

Starting with a simpler case, we could use two classic SDFs (as described in Section 3.1) to represent two adjacent interior regions where, like the outline renderer, a value of 0.5 represents the point where the boundary lies. If a render cutoff value of 0.5 is used for both, the boundary should appear sharp without a pixelated stepping effect, and without gaps (see Fig. 4). The fields would overlap beyond shape boundaries, thus requiring two separate fields each in their own channel, and unlike RelSDF, the gradient would be linear throughout to have a consistent distance metric.

Inspired by the Four-Colour Mapping Problem [1], we suggest that four distance fields would suffice to represent all interior regions. We first use an algorithm that solves the Mapping Problem to separate distinct regions (based on colour, within a threshold of similarity) and assign them ID values from 1 to 4, such that no two neighbouring regions have the same ID. We then generate four

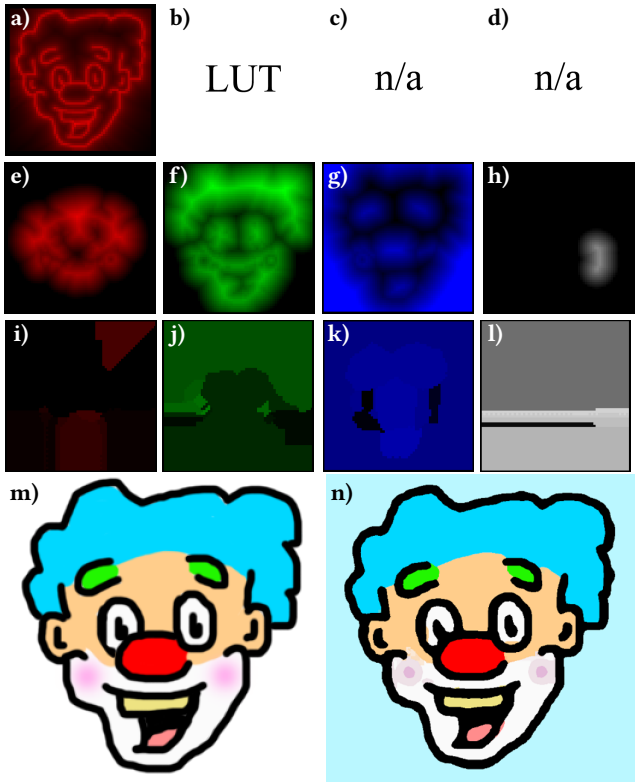

Figure 5: Channels of the complete solution (RelSDF and RegSDF). a) the RelSDF, b) RegSDF - colour Look-Up Table (LUT) with 4 values per colour, c)-d) (unused channels), e)-h) RegSDF - neighboring interior regions represented as Distance Fields, i)-l) RegSDF - RGBA mapping values to LUT for each corresponding interior region, m) input image, n) rendered output.

SDFs, one per ID. Gradients aren't guaranteed to reach 0.0 or 1.0 as distant SDFs overlap within the same channel, but values of 0.5 still accurately represent region boundaries. The four channels can be stored in a single RGBA texture.

RegSDF represents regions, but does not describe what colour to render at each region. A second RGBA texture of four channels stores an index at each texel to a Look Up Table (LUT) of colours. We embed the LUT in a single channel of a raster image (each quarter dedicated to a value for either R, G, B or A). Having separate channels of overlapping colour indices better allows the shader logic to know what colour to render within a texel (one of two colours - a single channel could only store one colour at that texel).

On its own, RegSDF adds nine additional channels (see Fig. 5) to the original one used in RelSDF. Even so, if these ten data channels were stored at 128x128 resolution, against a single 512x512 original RGBA texture, the savings when uncompressed in system memory would be significant. And in situations where preserving fine details requires a higher resolution distance field, the smoother, crisper rendering further justifies the use of distance fields over the typical alternative of higher-resolution raster textures.

4 Results and Analysis

Experiments: To validate the rendering results of RelSDF, we chose a variety of tests against a basic distance field (DF) and a basic SDF. The implementation details we use for DF and SDF are described in Section 3.1. First, we generate the fields at a resolution of 256x256, and set a render cutoff value of 0.5 to directly resemble the original input image (or in the case of DF, a fixed value that approximates 0.5 to be more comparable to the other methods). Different scales of resolution were experimented with, and 256x256 was chosen to retain some of the thin details in the original image while still justifying itself through memory savings. For quantitative comparison, metrics for LPIPS [33], PSNR, and SSIM [32] were captured for each method against the raster input image. Code for LPIPS came from Zhang et al.'s code repository, and PSNR and SSIM from the skimage.metrics Python package. To see RelSDF's strengths, we add two additional tests: we increase the cutoff value to produce thinner outlines and compare the results again to the original input image, and separately, downsample the resolution further to 128x128 to compare DF, SDF and RelSDF. Note that DF and SDF do not support multi-colour images, so RegSDF is used during the render of all three candidates for these tests. The results for these three tests can be seen in Table 1 and 2, Fig. 6 and Fig. 9.

RegSDF is a special case since its feature is not supported by DF or SDF, so a direct comparison against them would be meaningless. As a supplementary experiment, we hand-picked four images in our dataset and zoom the camera into spots of interest, based on the variety of outlines or distinct regions present in the patch. While the prior tests use a raster image as input, the original vector (.svg) files for these four images were used as a ground truth. RegSDF with 256x256-sized fields was compared to a 512x512 rasterized version of the image, to be consistent with the resolutions of the main test for RelSDF. The zoomed-in patches were rendered at 512x512, but contain approximately 32x32 from the raster image, or 16x16 from the RegSDF fields. Render cutoff values for the RelSDF were manually adjusted between 0.45 and 0.55 to get the best result.

The focus of this study is sprites, glyphs and icons, with simple details and colours, and strong outlines. We constructed a dataset of 76 images for testing, made up of clip-art assets selected from OpenClipArt.org, plus a few custom-made sprites. Images were stored in raster format at 512x512 resolution (four images were in their original vector format for testing RegSDF). Images varied from simple to detailed cartoons, and simple patterns, colour and monochrome. Some representative results appear in Figs. 6 and 7.

RelSDF Results: In the overall average, Table 1 shows SDF performs better on average than DF and RelSDF for all three of our metrics, though RelSDF is close enough to be considered an adequate alternative. The test results in Table 2 prove to be more interesting. In both cases, RelSDF achieves the best LPIPS and SSIM score, while SDF still has the best PSNR score. Both tests show RelSDF to be more resilient in retaining details for both scenarios. The first test in Table 2 focuses on higher cutoff rendering values to result in thinner lines, and SDF struggles to retain those lines compared to RelSDF, and even against DF. Logically, distance values inside a thin line wouldn't be much higher than 0.50 in an SDF, but would be closer to 1.0 in both DF and RelSDF. The second test in Table 2 downsamples the images from Table 1 to 128x128, and similarly, SDF's less extreme distance values inside an outline cause those details to become lost more quickly, while RelSDF retains them. The LPIPS and SSIM results suggest that at varying cutoff values and resolutions, RelSDF is better at retaining structural and perceptual details that the viewer would recognize.

Table 1: Average metric scores against dataset of input raster images for basic Distance Field (DF), Signed Distance Field (SDF), and Relative Signed Distance Field (RelSDF). Metrics are LPIPS, PNSR, SSIM.

	Average Metric Scores - field of size 256x256, cutoff 0.5		
	LPIPS ↓	PSNR ↑	SSIM ↑
DF	0.246	12.082	0.712
SDF	**0.113**	**19.34**8	**0.879**
RelSDF	0.117	18.884	0.875

Table 2: Left: Results to reproduce thinner line size, cutoff values of 0.75 (DF), 0.515 (SDF) and 0.65 (RelSDF). Right: Render results, cutoff 0.5, rendered from 128x128 distance fields.

	Avg. Metric Scores - cutoff above 0.5			Avg. Metric Scores - field size 128x128		
	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑
DF	0.199	14.764	0.815	0.267	12.398	0.698
SDF	0.228	**15.946**	0.798	0.188	**17.638**	0.816
RelSDF	**0.191**	15.264	**0.816**	**0.166**	17.279	**0.819**

In Fig. 6, and even more obviously in Fig. 9, we can see qualitative examples that explain the results in Table 2. When downsampling to a lower resolution, thin lines of only 1 or 2 pixels wide in the 512x512 input image would be lost with the SDF (a value slightly above 0.50 would be averaged down to under 0.50 – using a cutoff value of 0.495 brings back details to look closer to the original, at the expense of other lines looking too thick). The distance metric used by RelSDF places a higher value in those thin lines at the original resolution, helping preserve the feature after downsampling (although the features look slightly thicker than they should be). It corroborates the claim in the literature [11] that SDFs have trouble with features as little as one texel wide, and RelSDF overcomes this for thin lines in the input image by storing larger values at the centre of those lines. However, thin lines will have wave-like silhouettes, instead of the desired smooth boundary seen in DF or SDF. We attribute this artifact to the non-linear rate of change in the RelSDF gradient, and specifically, to how bilinear interpolation in the rendering shader uses this gradient data. The effect is reduced when the gradient change is less severe (e.g. if downsampling to even lower resolutions, or if outlines in the original image are wider). No temporal artifacts occur in motion from this effect.

RegSDF + RelSDF Results: Up close, issues with the RSDF (RelSDF + RegSDF) become apparent, as seen in Fig. 8. Outline thickness do not perfectly match the input raster image throughout, in part because of ambiguity in extracting the line boundary. Interior regions also have mild waves when rendered, a side-effect of

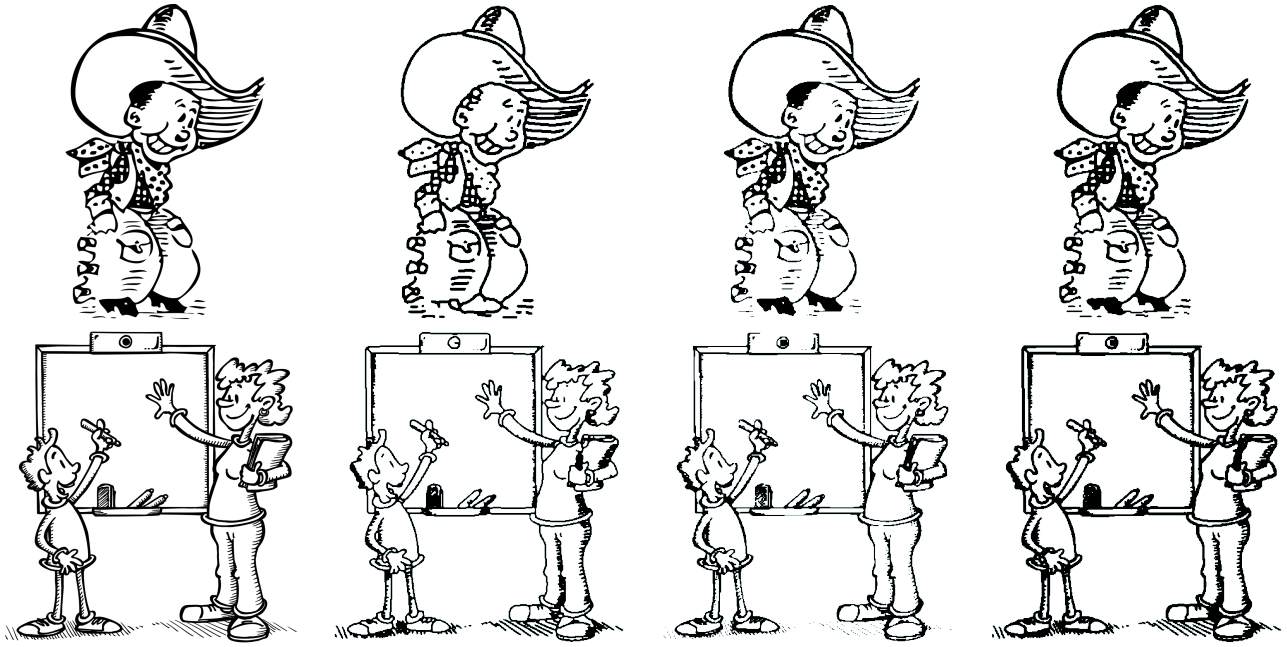

Figure 6: Some examples of line-only sprites. Distance fields each rendered with 256x256 resolution, against original input of 512x512 resolution. From left to right: a) the original input sprite, b) output render from DF (cutoff value 0.80 top row, 0.85 bottom row), c) output render from SDF (cutoff value 0.5), d) output render from RelSDF (cutoff value 0.5).

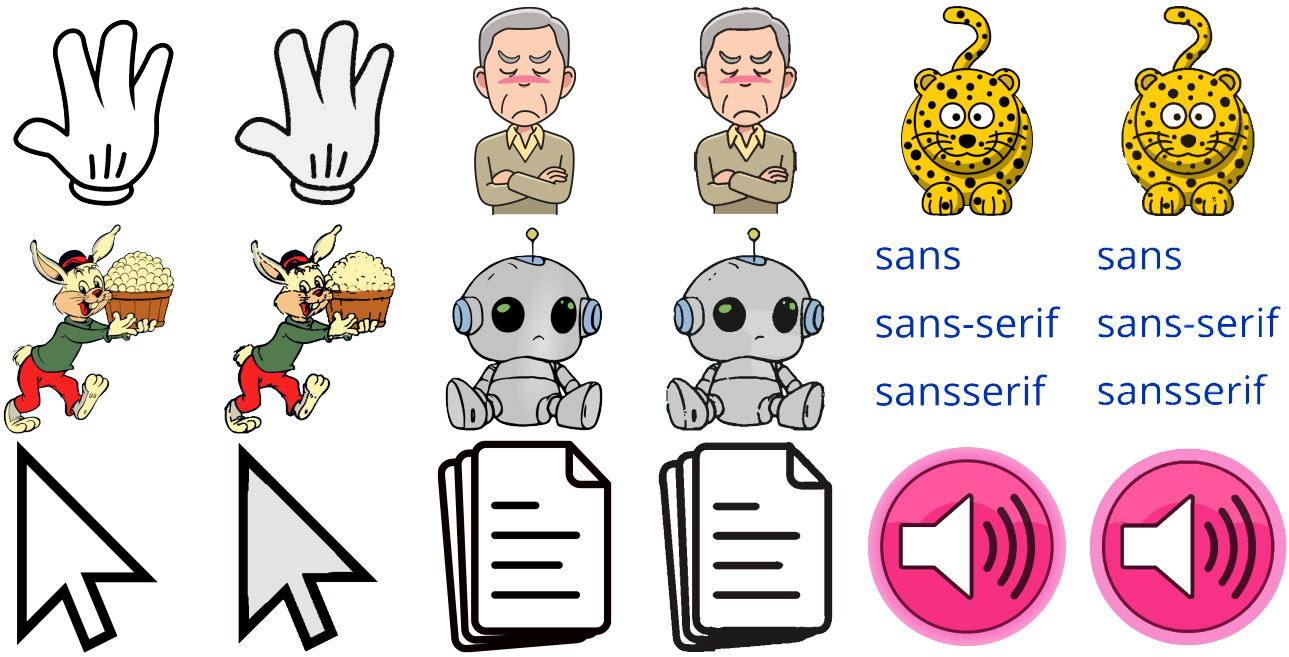

Figure 7: Some examples of sprites with different colours and styles. For each pair, the left is the original image (512x512), and the right is the output render from RelSDF + RegSDF (256x256, cutoff value 0.5).

Figure 8: Examples from test comparing zoomed-in (4x) render results, with vector asset as ground truth. From left to right: a) original input raster image, b) rendered image using RelSDF + RegSDF, c) zoom in of vector image, d) zoom in of raster image, e) zoom in of RelSDF + RegSDF render.

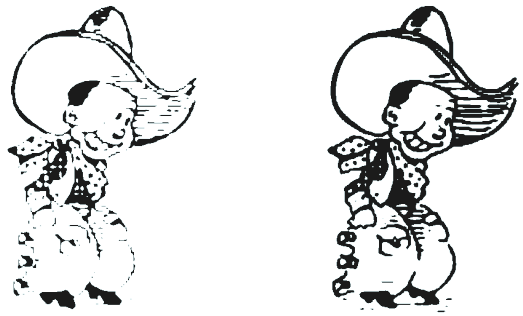

Figure 9: Example output from further downsampled 128x128 fields. Left: SDF render. Right: RelSDF render.

Table 3: Average metric scores against dataset of zoomed-in (x4) input vector images for RelSDF + RegSDF (256x256), and the raster image used to produce the SDF (512x512).

Average Metric Scores			
	LPIPS ↓	PSNR ↑	SSIM ↑
Raster	0.318	**19.334**	0.805
RelSDF + RegSDF	**0.203**	17.321	**0.836**

generating the field from a raster input instead of vector. Coloured regions are mostly reproduced well, but some detailed boundaries are smudged together. To the human eye, these results are not as blocky as the raster image, but not as clean as the vector original.

In the quantitative metrics (Table 3), RSDF performs well compared to the raster image. The LPIPS and SSIM score is better for RSDF, but worse with PSNR. Looking at the individual results for the eight image sets, RSDF achieves a better LPIPS score for all eight, and a better SSIM score for five of eight. Raster achieves better PSNR scores for all eight images. In particular, one image set that had only interior coloured regions and no outlines (as seen in the bottom row of Fig. 8) performed especially well with RSDF, suggesting the bold outlines in particular are the weakness, and possibly validating RegSDF (although the colours being similar between these interior regions was also likely a factor).

When distance fields are actually used, they are stored uncompressed as 32-bit RGBA in system or video memory; real-time decompression is not practical for large sets of images. Under these conditions, even three 256×256 RSDF fields occupy less space than a single 512×512 image, and they can be downsampled further without visible pixelation. Independent from resolution, note that RSDF data (gradients, rather than solid colurs) is not competitive with common compression methods for local storage size. As a brief example, consider the Clown image in Fig. 5, with an original raster input image as a 512x512 .png, compared to an RSDF stored as three 256x256 .png files, or again as an RSDF as three 128x128. Their on-disk and RAM size respectively are [64KB, 8.39MB], [164KB, 5.24MB], [54KB, 1.31MB]. RSDF's local storage cost does not affect runtime performance once loaded into memory.

In summary, RelSDF has benefits over SDF for preserving line details at varying resolutions and render cutoff values. RegSDF renders results closer in quality to a vector asset, compared to a higher-resolution raster asset.

5 Conclusions and Future Work

In this paper, we introduced an enhanced 2D distance field for rendering cartoon-like images. This includes using a distance metric that maintains relative line thickness within the same image (RelSDF), and a fixed number of distance fields (inspired by the 4-colour mapping problem) to render multiple interior coloured

regions (RegSDF). Compatible with modern game engines, this allows a more general class of images to be rendered with a crisp sharpness that normally requires high-resolution assets.

The success of RegSDF relies on input images not having noise or faded blending between coloured regions. More advanced methods for flattening colours should be explored to improve this part of the pipeline. Our implementation of RelSDF naïvely assumes (like standard implementations of distance fields) that outlines are of a single colour, rather than making use of the now available LUT. Solutions for the wavy artifacts present in RelSDF could be explored, such as specialized shader interpolation method between texels.

Reducing the number of channels to store the RelSDF and RegSDF is of significant interest. Strategies include combining data in a shared channel through alternating texels, or by dividing bits among each dataset. Alternatively, the two empty channels could be used for rendering sharp corners, possibly with lower resolutions [7].

Notably, we focus on simplified shapes, and not complex subjects or gradients. The data in RegSDF adapts well to blending multiple colours at a given texel, and some initial experimentation was done to render this with a gradient-strength control parameter. Currently, localized gradient information similar to functionality in diffusion curves [22], and automated extraction of gradients from an input image, are absent and would be a valuable contribution. The separation of outlines with RelSDF allows new types of effects, such as adding dynamic noise or masks to the outlines, to be explored.

Acknowledgments

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference numbers 521533407, RGPIN-2018-06298].

Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), [numéros de référence 521533407, RGPIN-2018-06298].

References

[1] Kenneth Appel and Wolfgang Haken. 1976. Every planar map is four colorable. (1976).

[2] Tristan Aumentado-Armstrong, Stavros Tsogkas, Sven Dickinson, and Allan Jepson. 2025. Probabilistic Directed Distance Fields for Ray-Based Shape Representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2025). doi:10.1109/TPAMI.2025.3594225

[3] Róbert Bán and Gábor Valasek. 2020. First Order Signed Distance Fields.. In *Eurographics (Short Papers)*. 33–36. doi:10.2312/egs.20201011

[4] Annada Prasad Behera and Subhankar Mishra. 2023. Neural directional distance field object representation for uni-directional path-traced rendering. In *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE, 1–6. doi:10.48550/arXiv.2306.16142

[5] Arpan Biswas and Vadim Shapiro. 2004. Approximate distance fields with non-vanishing gradients. *Graphical Models* 66, 3 (2004), 133–159. doi:10.1016/j.gmod.2004.01.003

[6] Andy Chalk. 2024. Space Marine 2's free 4K texture pack takes up more space than the entire base game, and you're gonna need some serious hardware to run it. Retrieved December 12, 2025 from https://www.pcgamer.com/games/third-person-shooter/space-marine-2s-free-4k-texture-pack-takes-up-more-space-than-the-entire-base-game-and-youre-gonna-need-some-serious-hardware-to-run-it/

[7] Viktor Chlumskỳ, Jaroslav Sloup, and I Šimeček. 2018. Improved corners with multi-channel signed distance fields. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 273–287. doi:10.1111/cgf.13265

[8] Sarah F Frisken and Ronald N Perry. 2006. Designing with distance fields. *ACM SIGGRAPH 2006 Courses* (2006), 60–66. doi:10.1145/1185657.1185675

[9] Sarah F Frisken, Ronald N Perry, Alyn P Rockwood, and Thouis R Jones. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 249–254. doi:10.1145/344779.344899

[10] Chris Green. 2007. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 courses*. 9–18. doi:10.1145/1281500.1281665

[11] Stefan Gustavson. 2012. 2D shape rendering by distance fields. (2012).

[12] Anna Lili Horváth, Gábor Valasek, and Róbert Bán. 2025. Distance field generation by proxies. *Computer-Aided Design & Applications* 22, 5 (2025). doi:10.14733/cadaps.2025.805-824

[13] Daniel Ilett. 2022. Transparency and Alpha. In *Building Quality Shaders for Unity®: Using Shader Graphs and HLSL Shaders*. Springer, 273–320.

[14] JayTheDevGuy. 2023. 2D Characters in 3D Worlds (and how I improved them). Retrieved December 12, 2025 from https://www.youtube.com/watch?v=s4dBvSj9Zpo

[15] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. 2020. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 1251–1261. doi:10.1109/CVPR42600.2020.00133

[16] Mark W Jones, J Andreas Baerentzen, and Milos Sramek. 2006. 3D distance fields: A survey of techniques and applications. *IEEE Transactions on visualization and Computer Graphics* 12, 4 (2006), 581–599. doi:10.1109/TVCG.2006.56

[17] Mark W Jones and Richard Satherley. 2001. Using distance fields for object representation and rendering. In *Proc. 19th Ann. Conf. of Eurographics (UK Chapter)*. London, 37–44.

[18] Maximilian Kohlbrenner and Marc Alexa. 2025. Isosurface Extraction for Signed Distance Functions using Power Diagrams. In *Computer Graphics Forum*. Wiley Online Library, e70037. doi:10.1111/cgf.70037

[19] Yu-Tao Liu, Li Wang, Jie Yang, Weikai Chen, Xiaoxu Meng, Bo Yang, and Lin Gao. 2023. Neudf: Leaning neural unsigned distance fields with volume rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 237–247. doi:10.1109/TPAMI.2023.3335353

[20] MartinDonald. 2020. Glyphs, shapes, fonts, signed distance fields. Retrieved December 12, 2025 from https://www.youtube.com/watch?v=1b5hIMqz_wM

[21] Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. 2024. Differential Walk on Spheres. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–18. doi:10.1145/3687913

[22] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 1–8. doi:10.1145/1360612.1360691

[23] Stanley Osher, Ronald Fedkiw, and Krzysztof Piechor. 2004. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.* 57, 3 (2004), B15–B15. doi:10.1115/1.1760521

[24] Yang Qi, Dario Seyb, Benedikt Bitterli, and Wojciech Jarosz. 2022. A bidirectional formulation for Walk on Spheres. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 51–62. doi:10.1111/cgf.14586

[25] Zheng Qin, Michael D McCool, and Craig Kaplan. 2008. Precise vector textures for real-time 3D rendering. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*. 199–206. doi:10.1145/1342250.1342281

[26] Nicolas Ray, Xavier Cavin, and Bruno Lévy. 2005. Vector texture maps on the GPU. *Inst. ALICE (Algorithms, Comput., Geometry Image Dept. INRIA Nancy Grand-Est/Loria), Tech. Rep. ALICE-TR-05-003* (2005).

[27] Azriel Rosenfeld and John L Pfaltz. 1966. Sequential operations in digital picture processing. *Journal of the ACM (JACM)* 13, 4 (1966), 471–494.

[28] Pradeep Sen. 2004. Silhouette maps for improved texture magnification. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. 65–73. doi:10.1145/1058129.1058139

[29] Pradeep Sen, Mike Cammarano, and Pat Hanrahan. 2003. Shadow silhouette maps. *ACM Transactions on Graphics (TOG)* 22, 3 (2003), 521–526. doi:10.1145/882262.882301

[30] JB van Velzen. 2018. Texture-based Rendering of Vector-based Shapes. (2018).

[31] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022. Differentiable signed distance function rendering. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–18. doi:10.1145/3528223.3530139

[32] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612. doi:10.1109/TIP.2003.819861

[33] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*. doi:10.1109/CVPR.2018.00068

[34] Tongjie Y Zhang and Ching Y. Suen. 1984. A fast parallel algorithm for thinning digital patterns. *Commun. ACM* 27, 3 (1984), 236–239. doi:10.1145/357994.358023

[35] Yiyu Zhuang, Qi Zhang, Ying Feng, Hao Zhu, Yao Yao, Xiaoyu Li, Yan-Pei Cao, Ying Shan, and Xun Cao. 2023. Anti-aliased neural implicit surfaces with encoding level of detail. In *SIGGRAPH Asia 2023 Conference Papers*. 1–10. doi:10.1145/3610548.3618197