Exact Signed Distance Function Representation of Polygons

Csaba Bálint[1], Gábor Valasek[2], Róbert Bán[3]

[1]Eötvös Loránd University, csabix@inf.elte.hu
[2]Eötvös Loránd University, valasek@inf.elte.hu
[3]Eötvös Loránd University, rob.ban@inf.elte.hu

Corresponding author: Csaba Bálint, csabix@inf.elte.hu

Abstract. We present an explicit representation of the exact signed distance function of polygons. We bound the generalized Voronoi regions of the vertices and edges by polygons and approximate them by employing planar cuts. The signed distance function within these regions is explicitly represented by either a point-point or a point-line distance formula. We show that the triangulations of these polygonal bounds can be efficiently rendered using rasterization and used in real-time applications. We also present conservative optimizations to the region generation algorithm.

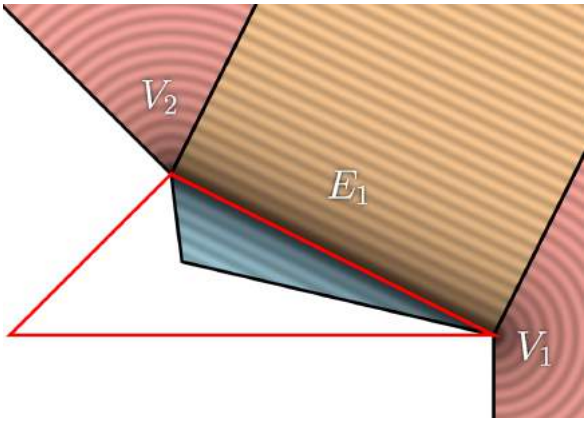
Keywords: Signed Distance Function, Implicit Representation, Geometric Modeling, Collision Detection
DOI: https://doi.org/10.14733/cadaps.2023.1029-1042

1 INTRODUCTION

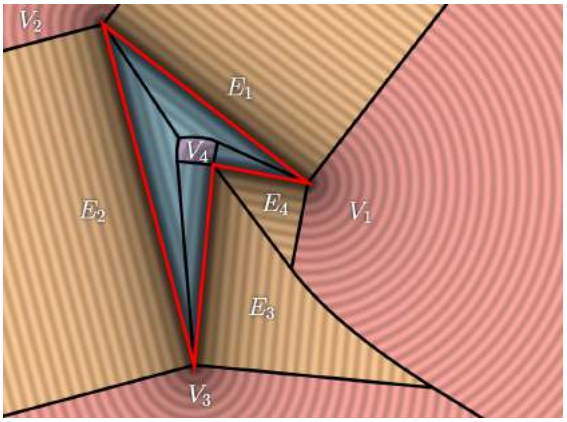
Signed distance functions (SDF) are used in a wide range of applications, for example in high-quality text rendering [7], geometric representation for collision detection [11, 12], 3D printing, additive manufacturing [4], and advanced real-time graphics effects [20]. Most of these applications rely on representations that evaluate approximations of the scene SDF.

Although the signed distance function is continuous everywhere, it is not differentiable at points that do not have a unique closest boundary point. The set of these points is the cut locus, and SDF approximations struggle in its neighborhood. Neither spatial subdivision nor higher-order algebraic SDF approximations can adapt to the structure of the cut locus without enlisting other means to resolve the ambiguity of the closest boundary entity. In this sense, the main challenge in high accuracy SDF representations is to properly handle the complexity of the cut locus and the regions it defines.

We propose an exact SDF representation for planar polygons by noting that the SDF of a polygon consists of instances of two different classes of elementary regions, depending on the topological type of the closest boundary point. Regions have a circular SDF if the closest point of the boundary is a vertex of the polygon;



(a) The SDF of an edge affects three regions, two corner and a linear edge region with increasing SDF values from the inside negative values to positive outside values.



(b) The exact signed distance function and Voronoi diagram of a concave quadrilateral. Note the parabolic boundaries between V_1 and E_3, and around V_4.

Figure 1: The exact signed distance function (SDF) of a line segment within a triangle (a) and a concave quadrilateral (b). Vertex regions are red and blue and line regions are orange and sky-blue colored. Distance-based coloring highlights their linear and circular nature.

that is, the exact SDF of the polygon is evaluated by a signed point-point distance computation. Regions, where the closest point is an interior point of an edge, possess a linear SDF; thus, the exact signed distance of the polygon is linear. Both of these elementary SDF region types possess only two scalar degrees of freedom. The way these partition the plane is where the algebraic complexity of the SDF lies, as these regions are separated by parabolic and linear boundaries.

We propose an algorithm that constructs conservative polygonal bounds for these regions. First, we compute the initial infinite vertex and edge regions of the polygon without considering the neighbors. Then our algorithm performs a series of cuts to determine the bounding polygons to represent the elementary SDF regions. During these cuts, the boundaries of the regions may become parabolic. We approximate these by a predefined number of tangent polylines that produce a conservative bound on the exact boundary. The exact SDF can be evaluated using these polygons and their associated SDF computations.

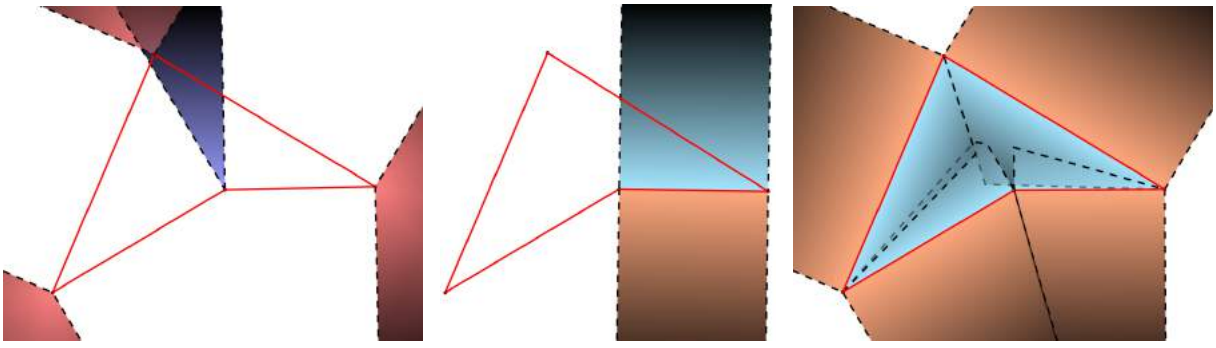
Section 2 provides a short overview of the related work in the literature. Section 3 defines the vertex and edge SDF region types and investigates their geometric and analytic properties. We present our main algorithm in Section 4 and propose generation and render time improvements in Section 5. Finally, we present our test results on generation times and rendering performance in Section 6.

2 PREVIOUS WORK

This overview considers two approaches to represent the signed distance function of a composite geometry. Direct or explicit methods generate data at prescribed positions and construct local approximations or exact representations of the SDF in an area. Indirect or procedural approaches employ proxy entities or acceleration structures to evaluate the exact SDF on a smaller subset of the initial geometry or their approximations.

Most direct methods only offer approximations albeit at a fixed evaluation cost. Procedural or indirect methods are more often exact, however, their evaluation cost may depend on the complexity of the signed distance function around the query position.

In real-time graphics, SDF representations are predominantly explicit [1, 7, 20]. The most common tech-



(a) Vertex regions are initially cut along neighboring edge normals from the vertex. The red regions have positive SDF values, these outside regions have convex angles.

(b) A single edge region contains both inside and outside regions. They are initialized as the region between these perpendicular lines.

(c) Overlapping edge regions after the edge regions have been cut with every other entity. The regions remain convex and the overlap is resolved by the Z-buffer algorithm.

Figure 2: Although vertex (a) and edge (b) regions are initially unbounded, the sign of the SDF is determined already. The regions remain convex throughout the cutting process.

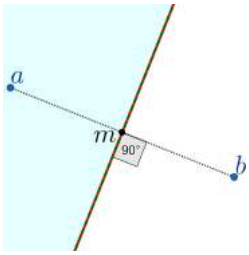
niques sample the SDF of the scene on a regular grid or a more adaptive spatial subdivision [6] and use interpolation, usually trilinear hardware-accelerated filtering, to infer an approximation to the true distance. These samples may be augmented by gradient data [3, 5]. Wu and Kobbelt [21] proposed binary space partitioning for subdivision and stored linear approximations to the SDF in each cell. Sud et al. [17] proposed a linear factorization of the SDF for efficient distance field computation of piece-wise linear input geometries.

A more functional approach uses higher-order interpolation and approximation schemes to estimate the exact signed distance values. Koschier et al. [11] proposed an octree subdivision of space where each cell contained a best fit polynomial of different degrees. The reconstructed SDF was discontinuous along cell boundaries. Song et al. [16] constructed globally continuously differentiable hierarchical spline approximations using Hermite interpolation over T-meshes. Our algorithm generates an exact continuous SDF without enforcing C^1 continuity everywhere.

Procedural methods store closest entities, often referred to as sites, to reconstruct the exact signed distance function on a smaller data set. A bounding volume hierarchy-based evaluation was proposed in [14], while all closest sites were stored explicitly in [9] for each cell. In both cases, the evaluation of the SDF required finding the smallest distance on a restricted number of entities. Machine learning methods have also been applied to extract SDF representation of shapes for processing and various other tasks [13, 19].

In contrast to the previous methods, we store the exact signed distance functions of piece-wise linear planar shapes. We explicitly compute the Voronoi regions of the topological entities of the geometry, i.e., all sets of points closest to either a particular vertex or interior point of a particular edge. The SDF within these regions is trivial and resolved precisely by evaluating either a point-point or a point-plane signed distance computation. The boundaries of these regions are composed of linear and parabolic segments. We bound these regions by convex polygons, so computing the closest point requires the SDF evaluation of overlapping polygons.

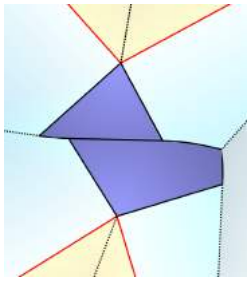
Efficient algorithms were proposed [8, 22] to compute the generalized Voronoi diagram of planar shapes composed of vertices, line segments, and optionally circular arcs. These Voronoi diagrams and medial axis computations are related to these SDF regions. The cut locus is the set of points with a non-unique closest point to the boundary and separates the elementary SDF regions. The medial axis is restricted to the interior of the polygon [18]. Aichholzer et al. [2] proposed using biarcs to approximate the boundary and computed the exact medial axis of this biarc approximation. This paper does not consider circular arcs, but the SDF



(a) Vertex regions are cut with the perpendicular bisector.

Algorithm 1 VertexRegion.VertexCut

Input: $\boldsymbol{a} \in \mathbb{E}^2$, $V_{\boldsymbol{a}}$ a vertex and its polygonal region; $\boldsymbol{b} \in \mathbb{E}^2$ another vertex.
Output: $V'_{\boldsymbol{a}}$ region that has been cut
$\boldsymbol{v} = \boldsymbol{b} - \boldsymbol{a}$ ▷ This will be the normal of the line we cut with
if $\|\boldsymbol{v}\| \leq 10^{-13}$ **then**
  return $V_{\boldsymbol{a}}$ ▷ Safeguard for duplicate vertices.
else
  $\boldsymbol{m} = \frac{1}{2}(\boldsymbol{a} + \boldsymbol{b})$ ▷ Midpoint
  return polycut$(V_{\boldsymbol{a}}, \frac{1}{2}(\boldsymbol{a} + \boldsymbol{b}), \boldsymbol{v})$ ▷ Convex polygon cut
end if



(b) Vertex region cuts leave no overlap between vertex regions.

Figure 3: Cutting a vertex region $V_{\boldsymbol{a}}$ with the perpendicular bisector from another vertex $\boldsymbol{b}$.

values are stored in the Voronoi cells.

We propose a similar approach to compute an approximate SDF to planar shapes with nonlinear parametric boundaries by first approximating the boundaries by polylines and computing the exact SDF of the resulting polygons. The cut locus can be trivially extracted from our representation by enumerating the boundaries of the vertex and edge SDF regions.

3 VORONOI INTERPRETATION

Let us consider an arbitrary polygon, defined by its vertices $\boldsymbol{v}_i \in \mathbb{E}^2$, $i = 1, \ldots, N$. Let

$$e_i = \{(1-t)\boldsymbol{v}_i + t\boldsymbol{v}_{i+1} \mid t \in (0,1)\} \subseteq \mathbb{E}^2$$

denote the open line segments between $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i+1}$, where $\boldsymbol{v}_{N+1} = \boldsymbol{v}_1$. Then the SDF of a polygon is interpreted as the extension of Voronoi regions to line segments, that is, it consists of two types of sets: *vertex regions* V_i and *edge regions* E_i. Let us denote the set of points that are closest to $\boldsymbol{v}_i$ as:

$$V_i = \left\{\boldsymbol{x} \in \mathbb{E}^2 \mid \forall j \in \{1, \ldots, N\} : \|\boldsymbol{x} - \boldsymbol{v}_i\| \leq d(\boldsymbol{x}, e_j)\right\} \subseteq \mathbb{E}^2 .$$

where $\|\boldsymbol{x}\| = \|x\|_2$ is the Euclidean norm and $d(\boldsymbol{x}, A) = \inf_{\boldsymbol{a} \in A} \|\boldsymbol{x} - \boldsymbol{a}\|$ denotes the point-set distance. We cannot define edge regions analogously since the distance to an endpoint is the same as to the edge. Thus, we exclude the endpoint vertex regions:

$$E_i = \left\{\boldsymbol{x} \in \mathbb{E}^2 \mid \forall j \in \{1, \ldots, N\} : d(\boldsymbol{x}, e_i) \leq d(\boldsymbol{x}, e_j)\right\} \setminus (V_i \cup V_{i+1}) \subseteq \mathbb{E}^2. \tag{1}$$

Figure 1a depicts an edge region E_1 of a triangle with the two neighboring vertex regions V_1 and V_2. Every point in an edge region is strictly closer to its closest edge than to any vertex. There are always $2N$ regions for a given polygon with N vertices: N vertex and N edge regions.

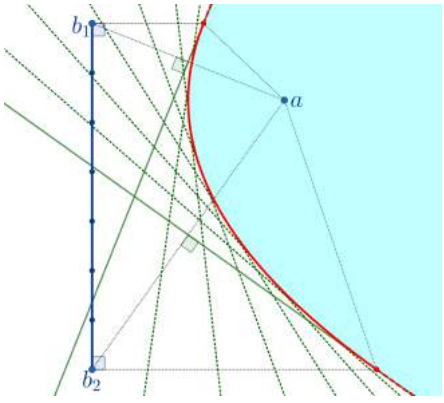
3.1 Vertex Regions

Vertex regions inherit the following important property from Voronoi regions that edge regions do not.

Proposition 1. *For any n-dimensional $\emptyset \neq Q \subseteq \mathbb{E}^n$ closed set, the vertex region generated by a $\boldsymbol{p} \in Q$ point*

$$V_{\boldsymbol{p}} = \left\{\boldsymbol{x} \in \mathbb{E}^n \mid \|\boldsymbol{x} - \boldsymbol{p}\| \leq d(\boldsymbol{x}, Q)\right\} \subseteq \mathbb{E}^n\}$$

is convex.



(a) Cutting parabola tangents are the dashed green lines between vertex $\boldsymbol{a}$ and edge $\boldsymbol{b}_1\boldsymbol{b}_2$.

Algorithm 2 VertexRegion.EdgeCut

Input: $\boldsymbol{a} \in \mathbb{E}^2$, $V_{\boldsymbol{a}}$ a vertex and its polygonal region
$\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathbb{E}^2$ another edge.
Output: $V'_{\boldsymbol{a}}$ region that has been cut
if $\|\boldsymbol{a} - \boldsymbol{b}_1\| \leq 10^{-13}$ **or** $\|\boldsymbol{a} - \boldsymbol{b}_2\| \leq 10^{-13}$ **then**
  return $V_{\boldsymbol{a}}$ ▷ If $\boldsymbol{a} \approx \boldsymbol{b}_1$ or $\boldsymbol{a} \approx \boldsymbol{b}_2$ then skip cutting
end if
for $t = \{\frac{1}{M}, \frac{2}{M}, \ldots, 1 - \frac{1}{M}\}$ **do** ▷ Parabola tangents of Fig. 4a
  $\boldsymbol{n} = \boldsymbol{b}_1 + t \cdot (\boldsymbol{b}_2 - \boldsymbol{b}_1) - \boldsymbol{a}$ ▷ Points to dividend point
  $\boldsymbol{p} = \boldsymbol{a} + \frac{1}{2}\boldsymbol{n}$ ▷ A point of the parabola tangent
  $V_{\boldsymbol{a}} = \text{polycut}(V_{\boldsymbol{a}}, \boldsymbol{p}, \boldsymbol{n})$ ▷ Perform the current cut
end for
return $V_{\boldsymbol{a}}$

Figure 4: The bounding polygon of a vertex region is cut with an approximation of the parabolic cut-locus between the vertex and another edge. Larger number of cuts $(M-1)$ yields tighter convex bounding polygon.

Proof. Let us prove by contradiction. Assuming that $V_{\boldsymbol{p}}$ is concave means that

$$\exists\, \boldsymbol{a}, \boldsymbol{b} \in V_{\boldsymbol{p}}, \exists\, t \in (0,1) \quad : \quad (1-t)\boldsymbol{a} + t\boldsymbol{b} = \boldsymbol{x} \notin V_{\boldsymbol{p}}\ .$$

Therefore, to have $\|\boldsymbol{x} - \boldsymbol{p}\| > d(\boldsymbol{x}, Q)$ there must exist a $\boldsymbol{q}^* \in Q$ point such that $\|\boldsymbol{x} - \boldsymbol{p}\| > \|\boldsymbol{x} - \boldsymbol{q}^*\|$. However,

$$\begin{aligned} \boldsymbol{a} \in V_{\boldsymbol{p}} \quad &\Longrightarrow \quad \|\boldsymbol{a} - \boldsymbol{p}\| \leq d(\boldsymbol{a}, Q) \leq \|\boldsymbol{a} - \boldsymbol{q}^*\| \\ \boldsymbol{b} \in V_{\boldsymbol{p}} \quad &\Longrightarrow \quad \|\boldsymbol{b} - \boldsymbol{p}\| \leq d(\boldsymbol{b}, Q) \leq \|\boldsymbol{b} - \boldsymbol{q}^*\| \end{aligned}$$

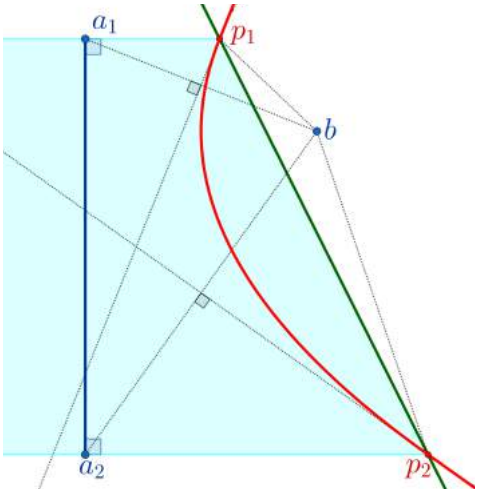
Thus, even though both $\boldsymbol{a}$ and $\boldsymbol{b}$ are closer to $\boldsymbol{p}$ than to $\boldsymbol{q}^*$, there exists some point $\boldsymbol{x}$ between them that is closer to $\boldsymbol{q}^*$ than to $\boldsymbol{p}$. This is a contradiction, since the perpendicular bisector of $\boldsymbol{p}\boldsymbol{q}^*$ cannot intersect the $\boldsymbol{a}\boldsymbol{b}$ segment. □

The borders of vertex region V_i are either straight segments or parabola segments with $\boldsymbol{v}_i$ as a focus. This is because the cut-locus, the equidistant set of points between two vertices, is the perpendicular bisector, as depicted in Figure 3a. The cut-locus between a segment and a point is a 'winged parabola', consisting of a parabola segment and two tangential rays that continue the parabola at both ends. The red line depicts this winged parabola in Figure 4a and 5a. The ray segments are perpendicular bisectors between the generator vertex of the vertex region and one of the segment endpoints. The parabola segment is defined by the generator vertex as a focus, and the segment as its directrix. The borders between vertex V_i and edge E_j regions are parabolas that curve around the $\boldsymbol{v}_i$ focus point. These convex boundary parabola segments agree with Proposition 1.

The distance function within a vertex region is just the distance to $\boldsymbol{v}_i$, e.g., $f(\boldsymbol{x}) = \|x - \boldsymbol{v}_i\|$, $x \in V_i$. However, each vertex region lies entirely either inside or outside the polygon. Assuming that the vertex order defines the inside partition properly, we have a signed distance function at each vertex as

$$f_{V_i}(\boldsymbol{x}) = \text{sgn}\left((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)^T R_{\frac{\pi}{2}}(\boldsymbol{v}_i - \boldsymbol{v}_{i-1})\right)\|\boldsymbol{x} - \boldsymbol{v}_i\| \qquad \boldsymbol{x} \in V_i\ , \tag{2}$$

where $R_{\frac{\pi}{2}} \in \mathbb{R}^{2\times 2}$ is the 90 degrees rotation matrix in the positive direction, and $\boldsymbol{v}_0 = \boldsymbol{v}_N$.



(a) Cutting the edge region (cyan) of $\boldsymbol{a}_1\boldsymbol{a}_2$ with the green line bounds the red cut locus.

Algorithm 3 EdgeRegion.VertexCut

Input: $\boldsymbol{a}_1, \boldsymbol{a}_2 \in \mathbb{E}^2$, $E_{\boldsymbol{a}}$ an edge and its polygonal region
$\boldsymbol{b} \in \mathbb{E}^2$ another vertex.
Output: $E'_{\boldsymbol{a}}$ edge region that has been cut
$\boldsymbol{n}_a = R_{\frac{\pi}{2}} \cdot [\boldsymbol{a}_2 - \boldsymbol{a}_1]_0$ ▷ Normal vector. $R_{\frac{\pi}{2}}$ is a $90°$ rotation
if $\boldsymbol{n}_a^T(\boldsymbol{b} - \boldsymbol{a}_1) \leq 10^{-13}$ **then**
  return $E_{\boldsymbol{a}}$ ▷ Do nothing if $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{b}$ are colinear
end if
$\forall\, i \in \{1, 2\} \;:\; t_i = \dfrac{\|\boldsymbol{b} - \boldsymbol{a}_i\|^2}{2\boldsymbol{n}_a^T(\boldsymbol{b} - \boldsymbol{a}_i)}$ ▷ Parabola intersect
$\forall\, i \in \{1, 2\} \;:\; \boldsymbol{p}_i = \boldsymbol{a}_i + t_i \cdot \boldsymbol{n}_a$ ▷ Calculate green cut-line
$\boldsymbol{n}_p = R_{\frac{\pi}{2}} \cdot (\boldsymbol{p}_2 - \boldsymbol{p}_1)$ ▷ Normal vector
if $\boldsymbol{n}_p^T(\boldsymbol{p}_1 - \boldsymbol{a}_1) < 0$ **then**
  $\boldsymbol{n}_p = -\boldsymbol{n}_p$ ▷ Flip the normal if it is facing the wrong way
end if
return polycut$(E_{\boldsymbol{a}}, \boldsymbol{p}_1, \boldsymbol{n}_p)$

Figure 5: Cutting an edge region $E_{\boldsymbol{a}}$ with a parabola defined with directrix $\boldsymbol{a}_1\boldsymbol{a}_1$ and another vertex $\boldsymbol{b}$. Since the parabola is concave from the edge side, a single linear cut is performed.

3.2 Edge Regions

Like vertex regions, edge regions are bordered by line and parabola segments. However, unlike vertex regions, edge regions can be concave; for example, in Figure 1b. This is, in part, because the cut-locus between an edge and a vertex forms the same 'winged parabola', as before on Figure 4a and 5a, but the directrix edge is on the concave side. The cut-locus between two edges consists of two parabola segments, a segment of the angular bisector, and two rays on perpendicular bisectors at each pair of endpoints. Two such configurations are visualized on Figure 6b and 6a.

For the signed distance function of an edge region, we need to project points within the E_i region onto the edge normal vector:

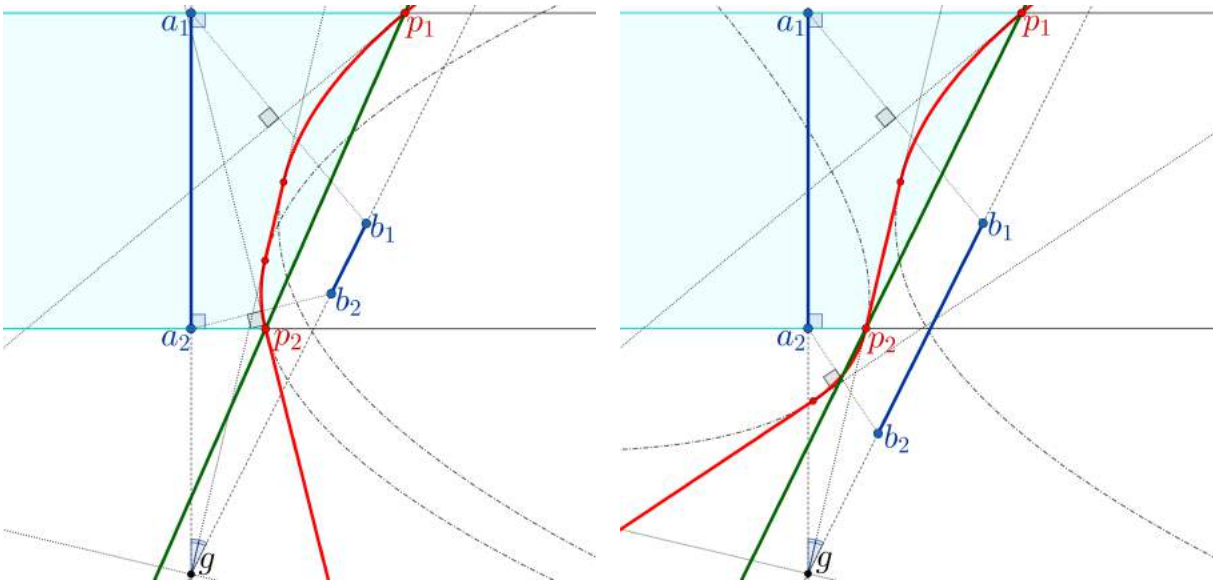
$$f_{E_i}(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{v}_i)^T R_{\frac{\pi}{2}} \, [\boldsymbol{v}_{i+1} - \boldsymbol{v}_i]_0 \qquad \boldsymbol{x} \in E_i \;, \tag{3}$$

where $[\boldsymbol{x}]_0 = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$ denotes vector normalization. This SDF is linear, and splits all E_i into inside and outside partitions along the $\boldsymbol{v}_i\boldsymbol{v}_{i+1}$ edge. By convention, we consider the negative partition to be the inside, which is achieved by clockwise enumeration of $\boldsymbol{v}_i$ polygon vertices.

3.3 SDF of a Polygon

The vertex and edge regions together cover all of $\mathbb{E}^2$, that is

$$\bigcup_{i=1}^{N} E_i \cup V_i = \mathbb{E}^2$$



(a) The projections of $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ onto $\boldsymbol{a}_1\boldsymbol{a}_2$ lie within the segment, so $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are parabola intersections.

(b) Projection of $\boldsymbol{b}_2$ onto $\boldsymbol{a}_1\boldsymbol{a}_2$ lies outside of the segment, so $\boldsymbol{p}_2$ is the intersection on the angular bisector.

Figure 6: Two cases for cutting edge regions for $\boldsymbol{a}_1\boldsymbol{a}_2$ with the cut-locus (red) from another edge $\boldsymbol{b}_1\boldsymbol{b}_2$. In each case, we calculate the $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ positions defining the line (green) to cut the edge region with (cyan).

The intersection of any two regions, $V_i \cap V_j$, $V_i \cap E_j$, or $E_i \cap E_j$ produce a set of cut-loci, each a zero-measure set. Now with Eq. 2 and 3, we can formulate the exact signed distance function of the whole polygon on $\mathbb{E}^2$:

$$f(\boldsymbol{x}) = \begin{cases} s_i \cdot \|\boldsymbol{x} - \boldsymbol{v}_i\| & \text{if } \boldsymbol{x} \in V_i \\ \boldsymbol{x}^T \boldsymbol{n}_i - \boldsymbol{v}_i^T \boldsymbol{n}_i & \text{if } \boldsymbol{x} \in E_i \end{cases} \qquad \boldsymbol{x} \in \mathbb{E}^2$$

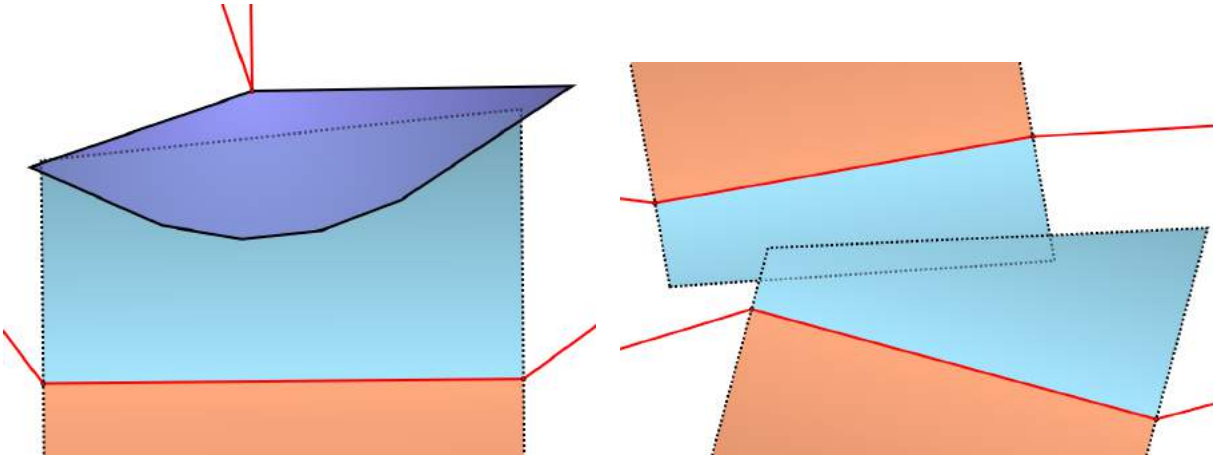
where $\boldsymbol{n}_i = R_{\frac{\pi}{2}} [\boldsymbol{v}_{i+1} - \boldsymbol{v}_i]_0$ is the unit normal of edge $\boldsymbol{v}_i, \boldsymbol{v}_{i+1}$ and $s_i = \operatorname{sgn}\big((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)^T R_{\frac{\pi}{2}} (\boldsymbol{v}_i - \boldsymbol{v}_{i-1})\big)$ the sign of the vertex region.

4 POLYGONIAL REGION BOUNDS

Our algorithm considers each possible pair of polygon entities, that is, combinations of vertices and edges, and gradually computes bounds to their Voronoi regions. This section gives an overview of the bounding region generation method, while Section 4.1 and 4.2 detail the algorithms for the various region cutting cases.

First, our method constructs the initial polygonal regions for all vertices and edges of the polygon. The vertex (Fig. 2a) and edge (Fig. 2b) regions are bounded by two half-planes that are perpendicular to the adjacent edges and the edge, respectively. We bound the infinite regions with a sizeable axis-aligned box. Within this box, exact SDF queries may be performed.

Second, we perform a series of planar cuts to reduce the region sizes to approximate the true one as close as possible. Figure 2c depicts the overlapping edge regions after the planar cuts. Note that we need not only intersect the regions of the two current entities; we also have to factor in the cut locus between them and consider how it splits and overlaps to form our convex polygonal bounds. Figure 2b illustrates the vertex-vertex, and Figure 7 visualizes vertex-edge, and edge-edge cases. Our algorithm is summarized as follows:



(a) Vertex region is cut with 5 lines approximating the parabola between it and the edge. The edge region is cut with a single line containing the whole parabola segment above the vertex.

(b) Edge regions usually have concave parabolic borders that we bound with a single cut. There are quite a few cases depending on the relative position of the segments that we had to consider.

Figure 7: We cut regions that are close to each other with bounding lines to retain the convexity of the polygonal regions. The vertex – edge (a) and edge – edge (b) cuts usually produce small overlaps that our rendering algorithm can handle.

1. Process input by splitting polygons into boundary components and interior holes. Outer boundaries are processed counterclockwise; hole boundaries are enumerated clockwise to generate correct SDF signs.
2. Initialize each region with its vertex or edge data to reconstruct the distance values, and add two perpendicular cuts using neighbors, see Figure 2.
3. For each pair of regions within each polygon and between neighboring polygons, perform a planar cut, as in Figure 7. The following section explains each type of intersection.

After the bounding region generation, the triangulated regions may be stored or rasterized to produce an exact signed distance field quickly. Although the bounding regions overlap, i.e., the regions do not cover the plane uniquely, the true SDF at any point may still be obtained by evaluating a small number of overlapping regions and using the minimum value.

4.1 Vertex Region Cuts

First, the region polygon starts as a sufficiently large origin-centered axis-aligned box. These large boxes are cut with the perpendicular lines of the adjoining edges going through this point, resulting in regions shown in Figure 2a. Each vertex region is initialized with its sign and its vertex position to represent the signed distance function on the region.

Second, each vertex region is cut with the cut locus formed with every other region according to Algorithm 1. This means that between vertex regions, we can perform exact cuts because the cut locus between two points is the perpendicular bisector (see Fig. 3a). If the input contained no edges, the algorithm would produce the Voronoi diagram without overlaps between the regions.

Finally, the vertex regions are cut with every other edge in Algorithm 2. Even though the vertex regions remain convex (see Proposition 1), the cut-locus between will be parabolic. Hence, we can cut with $M-1 \in \mathbb{N}$

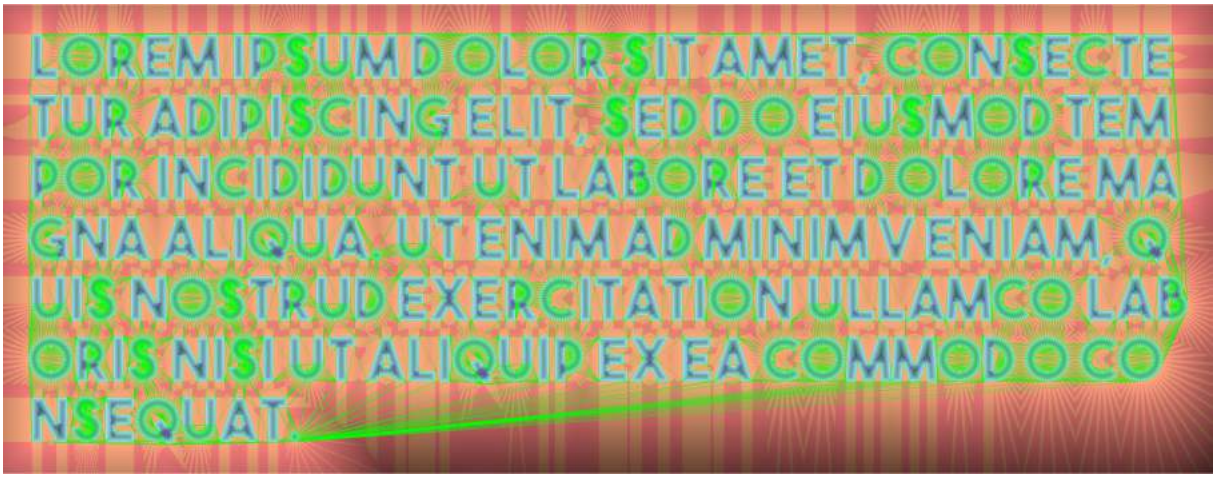


Figure 8: Cutting with Delaunay neighbours (green) allows generating vertex and edge regions for large inputs.

number of parabola tangents for producing a bounding polygon. Tangents of the parabola segment defined by $\boldsymbol{a}$ focus point and $\boldsymbol{b}_1\boldsymbol{b}_2$ directrix segment can be constructed by taking any $\boldsymbol{x} \in \boldsymbol{b}_1\boldsymbol{b}_2$ point on the segment. The tangent is the perpendicular bisector between the $\boldsymbol{x}$ and $\boldsymbol{a}$ (see Fig. 4a).

Note that in Algorithm 2, we do not cut tangents corresponding to endpoints. This is because Algorithm 1 performs the same cuts, and we found that cutting a polygon with the same line twice can result in numerical issues. We could cut with only one endpoint in Algorithm 2, but lose out on the generality of our method.

The signed distance function of the region is stored in two scalars containing the generating vertex of the region along with the sign of the distance values. Then, the triangulated polygon of the vertex region is stored.

4.2 Edge Region Cuts

Edge region bounds are also initialized to a large axis-aligned box which are then cut along the edge normal from both endpoints. An initialized edge looks like the one in Figure 2b.

Then, edge regions are cut with every vertex region with Algorithm 3. The algorithm computes the cut-locus parabola intersections $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ (see Figure 5a) and cuts the edge region bounding polygon with the line through these. Although this expands the regions and can create significant overlaps (see Fig. 7a), it simplifies the algorithm as we only consider linear and convex boundaries. The $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ parabola intersections with the perpendicular lines from $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ are obtained by intersecting the orthogonal lines at each endpoint with the perpendicular bisectors of $\boldsymbol{b}\boldsymbol{a}_1$ and $\boldsymbol{b}\boldsymbol{a}_2$, respectively.

Lastly, our algorithm cuts each edge region boundary polygon with the cut-locus between it and every other edge in Algorithm 4. The algorithm constructs a single bounding cut between $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ that contains the cut locus between the orthogonal lines at $\boldsymbol{a}_1\boldsymbol{a}_2$ endpoints. Figure 6 depicts two construction cases out of many, such as when the edges are connecting at an endpoint or when they are parallel. These geometric cases are applied in the computation of both endpoints to apply the cut. The method sometimes produces overlaps between regions, as in Figure 7b.

Along with the triangulated edge border polygon, the SDF of the region is stored in three scalars: a unit-length normal and constant scalar term.

Algorithm 4 EdgeRegion.EdgeCut

Input: $\boldsymbol{a}_1, \boldsymbol{a}_2 \in \mathbb{E}^2$, $E_{\boldsymbol{a}}$ an edge and its polygonal region; $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathbb{E}^2$ another edge.
Output: $E'_{\boldsymbol{a}}$ edge region that has been cut

$\forall\, i \in \{1,2\}\ :\ d_i = \dfrac{(\boldsymbol{b}_i - \boldsymbol{a}_1)^T(\boldsymbol{a}_2 - \boldsymbol{a}_1)}{\|\boldsymbol{a}_2 - \boldsymbol{a}_1\|^2}$ ▷ Projection, such that $(1-d_i)\boldsymbol{a}_1 + d_i\boldsymbol{a}_2 - \boldsymbol{b}_i \ \parallel\ \boldsymbol{n}_a$
if $d_2 < d_1$ **or** $d_2 < 0$ **or** $1 < d_1$ **then** ▷ Check if input is reversed, or only intersects with one parabola
  return $E_{\boldsymbol{a}}$ ▷ Reverse input is omitted; parabolic cut is handled by Algorithm 3
end if
$\boldsymbol{n}_a = R_{\frac{\pi}{2}} \cdot [\boldsymbol{a}_2 - \boldsymbol{a}_1]_0$, $\boldsymbol{n}_b = R_{\frac{\pi}{2}} \cdot [\boldsymbol{b}_2 - \boldsymbol{b}_1]_0$ ▷ Unit-length perpendicular vectors
for $i \in \{1,2\}$ **do** ▷ For each of the two endpoints, there are four possibilities
  if $0 \le d_i \le 1$ **then** ▷ If the $\boldsymbol{b}_i$ is between $\boldsymbol{a}_1\boldsymbol{a}_1$, the intersection is parabolic
    if $\|\boldsymbol{a}_i - \boldsymbol{b}_i\| \le 10^{-13}$ **then**
      $t_i = 0$ ▷ If $\boldsymbol{a}_i \approx \boldsymbol{b}_i$, then we cut through $\boldsymbol{a}_i$
    else
      $t_i = \dfrac{\|\boldsymbol{b}_i - \boldsymbol{a}_i\|^2}{2\boldsymbol{n}_a^T(\boldsymbol{b}_i - \boldsymbol{a}_i)}$ ▷ Parabola with focus $\boldsymbol{b}_i$ directrix $\boldsymbol{a}_1\boldsymbol{a}_2$ intersects at $\boldsymbol{a}_i + t_i\boldsymbol{n}_a$
    end if
  else ▷ b_i is not above $\boldsymbol{a}_1\boldsymbol{a}_2$, we must cut with a line
    if $\left|\boldsymbol{n}_b^T(\boldsymbol{a}_2 - \boldsymbol{a}_1)\right| \le 10^{-13}$ **then**
      $t_i = \frac{1}{2}\boldsymbol{n}_a^T(\boldsymbol{b}_i - \boldsymbol{a}_i)$ ▷ If the lines are parallel, stop halfway between $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$
    else
      $t_i = \dfrac{\boldsymbol{n}_b^T(\boldsymbol{b}_i - \boldsymbol{a}_i)}{1 + \boldsymbol{n}_a^T\boldsymbol{n}_b}$ ▷ Angular bisector intersection solving $\boldsymbol{n}_b^T(\boldsymbol{b}_i - \boldsymbol{a}_i - t_i\boldsymbol{n}_a) = t_i$
    end if
  end if
  $\boldsymbol{p}_i = \boldsymbol{a}_i + t_i\boldsymbol{n}_a$ ▷ Calculate endpoint of the new convex cut
end for
$\boldsymbol{n}_p = R_{\frac{\pi}{2}} \cdot [\boldsymbol{p}_2 - \boldsymbol{p}_1]_0$ ▷ Normal vector of the new cut
if $\boldsymbol{n}_p^T(\boldsymbol{p}_2 - \boldsymbol{a}_1) < 0$ **or** $\boldsymbol{n}_p^T(\boldsymbol{p}_1 - \boldsymbol{a}_2) < 0$ **then**
  $\boldsymbol{n}_p = -\boldsymbol{n}_p$; ▷ Must flip normal if it is facing the wrong way
end if
return polycut$(E_{\boldsymbol{b}}, \boldsymbol{p}_1, \boldsymbol{n}_p)$ ▷ Execute edge region convex boundary cut

5 OPTIMIZATION

This paper ultimately presents two algorithms: one for generating Voronoi-like regions for polygons and another for efficiently rasterizing such regions to produce detailed signed distance fields.

Constructing $2N$ number of convex regions with each potentially having any number of vertices can be computationally expensive, especially if each region is cut with every other. For a more extensive polygonal, the $\binom{2N}{2}$ number of polygon cuts is unacceptable. For this reason, employing a spatial acceleration structure may be required, such as a regular grid, quadtree, or KD-tree. Note that it is not essential to cut with every other region because if we omit distant cuts, it most likely does not affect the final region shape. Even if omitting cuts results in a slightly larger bounding region at the end, the SDF rasterizing application can handle the occasional extra workload.

Delaunay triangulation For the aforementioned acceleration, we settled on a cutting with neighbors in a constrained Delaunay triangulation. We construct a Delaunay triangulation of all the vertices inside the input polygons and require the triangulation to contain the original polygon edges. We cut with neighboring vertices for each vertex region within the triangulation and all edges that connect to those neighbors. We employ the same strategy for the edge regions: we intersect with every vertex that neighbors one of its endpoints and all connecting edges. Delaunay triangulation construction runs in $O(N\log(N))$ time and prepares an index pair list for region cutting.

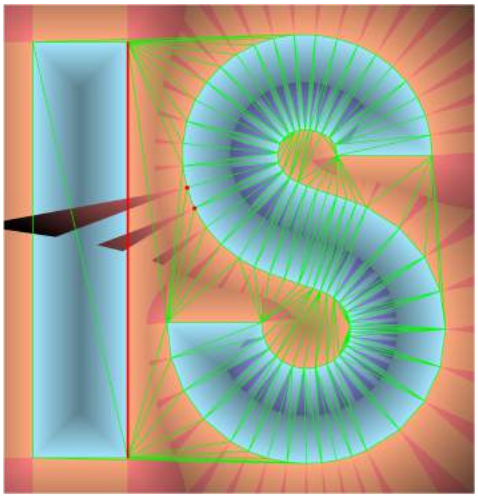


Figure 9: The oversized vertex regions of the red vertices are not cut with the red edge in the Delaunay triangulation, but the Z-buffer algorithm hides such artifacts.

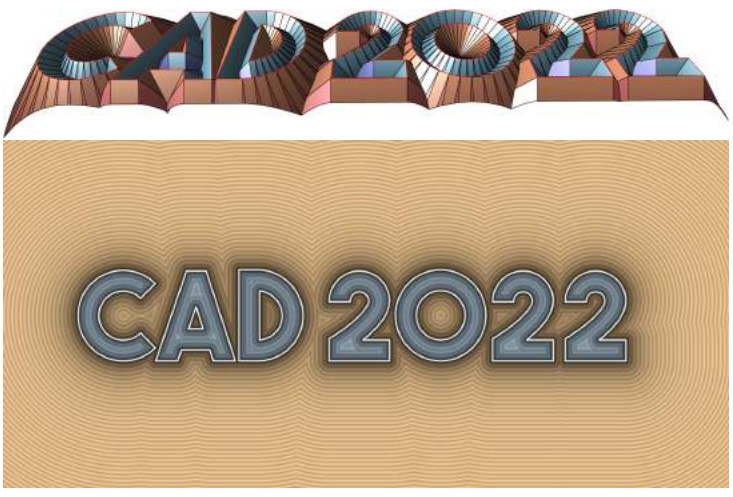


Figure 10: Region boundary overlaps are resolved in the third dimension (top), this allows fast exact SDF rasterization (bottom) via the Z-buffer. The ratio of overlaps is typically small. The performance is only limited by the GPU fill rate.

Creating the triangulation is always cheap compared to the region cutting algorithm, where in each step, a convex polygon is cut with a line. We found that the dynamic nature of the Delaunay neighbors gave almost consistently the neighboring regions, but not always as illustrated in Figure 9. The red vertices on the letter S do not have any endpoint of the red edge as a neighbor in the triangulation, that vertex region is not cut with the letter I. However, this is not an issue since the regions are bounding polygons, and such missing cuts can only lead to reduced performance when rendering the SDF. The occurrence of missing cuts is infrequent; we found the issue by chance in the long text on Figure 8.

Cutting an N-sided polygon can be done in $O(\log(N))$ steps [15]; however, such a divide and conquer algorithm would be slow since most regions will only have a few sides. Thus, we opted for a simpler linear algorithm polygon cutting algorithm that we optimized in Matlab.

SDF Rasterization The region generation outputs triangulated polygons that we can render using the graphics pipeline on the GPU. For each pixel-sized fragment, we set the depth value to that of the distance function. Since some bounding regions overlap, the Z-buffer algorithm will decide which edge or vertex is closer for each pixel. Note that the distance functions of the edge regions in Eq. 3 are linear, we can set the z coordinate or depth value of the region vertices to $\left|f_{E_i}(\boldsymbol{x})\right|$ and let the GPU interpolate between. However, the SDF of the vertex regions are non-linear, so each pixel-sized fragment depth value must be adjusted in the shader code to $\left|f_{V_i}(\boldsymbol{x})\right|$ from Eq. 2. We implemented a proof-of-concept SDF rasterizer in the NVIDIA Falcor framework [10].

6 TEST RESULTS

Table 1 summarizes SDF region generation times for TrueType font glyphs with Delaunay optimization using our Matlab implementation. A Matlab render of the Lorem text is shown in Figure 8, and a high-performance rasterization-based render of the bunny, deer, and tiger glyphs is shown in Figure 11. All measurements were carried out on a Ryzen 5 1600X CPU with 32GB memory. The top row of the table shows how many line

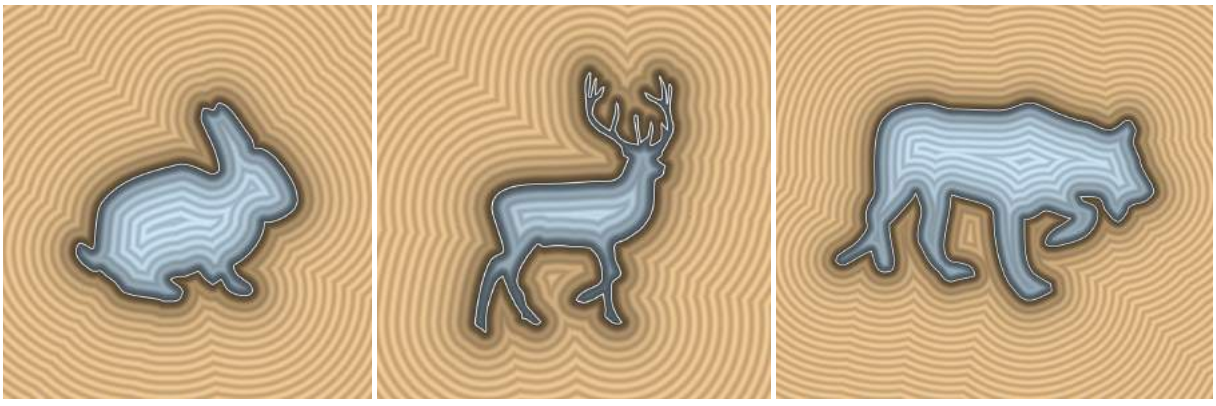


Figure 11: Real-time rasterization of the bunny (left), deer (center), and tiger (right) glyphs.

	1 line segment			2 line segments			3 line segments			4 line segments		
	#tris	runtime	overlap	#tris	runtime	overlap	#tris	runtime	overlap	#tris	runtime	overlap
Bunny	583	0.08s	0.84%	1117	0.16s	0.28%	1665	0.22s	0.18%	2208	0.18s	0.15%
Tiger	858	0.12s	0.68%	1684	0.23s	0.26%	2495	0.35s	0.16%	3303	0.28s	0.12%
Deer	1412	0.21s	1.06%	2740	0.41s	0.44%	4109	0.59s	0.30%	5411	0.49s	0.24%
Lorem	24257	11.77s	6.99%	39153	19.12s	7.18%	54273	25.03s	7.28%	69634	26.30s	7.39%

Table 1: Vertex and edge region bound number of triangles, generation times, and overlap ratios for the four animals from Fig. 11 and text in Fig. 8. The Bézier curves in the glyph input are divided into 1 to 4 segments.

segments were used to approximate the Bézier segments in the glyphs. For each discretization count between 1 and 4, we display the number of generated triangles (#tris), the SDF generation time in seconds (runtime), and the percentage of the multiply covered areas.

As the number of line segments increases, the redundancy ratio decreases. The ratio is generally higher for text and smaller for figures because text tends to have long line segments with region bounds overlapping with more regions. On the other hand, our algorithm creates bounding regions with minimal overlaps under uniform curve rectification. For example, vertex and edge regions of regular polygons do not overlap.

We compared the Delaunay optimization to the brute force approach with $\binom{2N}{2}$ polygon cuts. On average, the Delaunay neighbor approach was 6.11 times faster and up to 24.7 times faster for the deer glyph. For longer inputs, like the Lorem text, the brute force approach did not finish in a reasonable time frame. However, since the brute force method considers every intersection, we compared the overlap ratio to that of the Delaunay optimized version. The worst case among 60 test glyphs was the single letter S, which increased from the 1.2% brute force overlap ratio to the 2.1% overlap ratio, an 89% increase, but the overlap was still small. Moreover, the median ratio increase was only 1.3% from brute force to Delaunay neighbors.

We benchmarked our GPU implementation by rasterizing the SDF region triangulations of glyphs. We positioned each such that they covered the entire screen at 1920×1080 resolution. Render times on an NVIDIA 2080 were 0.06 milliseconds for all test glyphs except for the Lorem text shown in Figure 8. With various Bézier rectification levels on the Lorem text, we confirmed that render times are proportional to the total number of triangles. Increasing the number of Bézier line divisions to four increased render time from

0.06 to 0.22 milliseconds on the NVIDIA 2080 GPU. On an AMD Radeon RX 5700 GPU, as we increased the rectification level from one to four, the render time increased from 0.12 to 0.32 milliseconds.

7 CONCLUSIONS

We presented an algorithm to compute the exact SDF of polygonal shapes, including polygons with holes. The runtime of the initial Matlab implementation made it viable as a proxy for computing an SDF partitioning of shapes bounded by parametric curves, such as the ones found in TrueType fonts. We demonstrated that GPU accelerated rasterization could efficiently render the conservative SDF region bounds in real-time.

The generated SDF regions suggest that expanding the current solution by incorporating circular regions, i.e., by the SDF of a circle, we could make this representation more concise for fonts and other shapes and offsets of polygons. Merging regions to generate approximate SDF regions within the error threshold, expanding it to set-theoretic operations, and optimized generation using bounding volumes or space partitioning is subject to future research.

ACKNOWLEDGEMENTS

EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies – The Project is supported by the Hungarian Government and co-financed by the European Social Fund. Supported by the ÚNKP-21-3 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

We would like to thank Visual Concepts for providing the AMD GPU used in the tests.

REFERENCES

[1] Aaltonen, S.: GPU-based clay simulation and ray-tracing tech in Claybook, Game Developers Conference 2018. In Game Developers Conference. San Francisco, CA, 2018.

[2] Aichholzer, O.; Aigner, W.; Aurenhammer, F.; Hackl, T.; Jüttler, B.; Rabl, M.: Medial axis computation for planar free-form shapes. Computer-Aided Design, 41(5), 339–349, 2009. ISSN 0010-4485. http://doi.org/10.1016/j.cad.2008.08.008.

[3] Bán, R.; Valasek, G.: First Order Signed Distance Fields. In A. Wilkie; F. Banterle, eds., Eurographics 2020 - Short Papers. The Eurographics Association, 2020. ISBN 978-3-03868-101-4. ISSN 1017-4656. http://doi.org/10.2312/egs.20201011.

[4] Brunton, A.; Rmaileh, L.A.: Displaced signed distance fields for additive manufacturing. ACM Trans. Graph., 40(4), 2021. ISSN 0730-0301. http://doi.org/10.1145/3450626.3459827.

[5] Cohen-Or, D.; Solomovic, A.; Levin, D.: Three-dimensional distance field metamorphosis. ACM Trans. Graph., 17(2), 116–141, 1998. ISSN 0730-0301. http://doi.org/10.1145/274363.274366.

[6] Frisken, S.F.; Perry, R.N.; Rockwood, A.P.; Jones, T.R.: Adaptively sampled distance fields: A general representation of shape for computer graphics. In 27th Annual Conf. on Computer Graphics and Interactive Techniques, SIGGRAPH, 249–254. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000. ISBN 1-58113-208-5. http://doi.org/10.1145/344779.344899.

[7] Green, C.: Improved alpha-tested magnification for vector textures and special effects. In ACM SIGGRAPH 2007 Courses, SIGGRAPH '07, 9–18. ACM, New York, NY, USA, 2007. ISBN 978-1-4503-1823-5. http://doi.org/10.1145/1281500.1281665.

[8] Held, M.: Vroni: An engineering approach to the reliable and efficient computation of voronoi diagrams of points and line segments. Comput. Geom. Theory Appl., 18(2), 95–123, 2001. ISSN 0925-7721. http://doi.org/10.1016/S0925-7721(01)00003-7.

[9] Huang, J.; Li, Y.; Crawfis, R.; Lu, S.C.; Liou, S.Y.: A complete distance field representation. In Proceedings of the Conference on Visualization '01, VIS '01, 247–254. IEEE Computer Society, USA, 2001. ISBN 078037200X.

[10] Kallweit, S.; Clarberg, P.; Kolb, C.; Davidovič, T.; Yao, K.H.; Foley, T.; He, Y.; Wu, L.; Chen, L.; Akenine-Möller, T.; Wyman, C.; Crassin, C.; Benty, N.: The Falcor rendering framework, 2022. https://github.com/NVIDIAGameWorks/Falcor.

[11] Koschier, D.; Deul, C.; Bender, J.: Hierarchical Hp-adaptive signed distance fields. SCA '16, 189–198. Eurographics Association, Goslar, DEU, 2016. ISBN 9783905674613. http://doi.org/10.5555/2982818.2982844.

[12] Macklin, M.; Erleben, K.; Müller, M.; Chentanez, N.; Jeschke, S.; Corse, Z.: Local optimization for robust signed distance field collision. 3(1), 2020. http://doi.org/10.1145/3384538.

[13] Park, J.J.; Florence, P.; Straub, J.; Newcombe, R.; Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation, 2019.

[14] Sanchez, M.; Fryazinov, O.; Pasko, A.: Efficient evaluation of continuous signed distance to a polygonal mesh. In Proceedings of the 28th Spring Conference on Computer Graphics, SCCG '12, 101–108. Association for Computing Machinery, New York, NY, USA, 2012. ISBN 9781450319775. http://doi.org/10.1145/2448531.2448544.

[15] Shamos, M.I.: Geometric complexity. In Proceedings of the Seventh Annual ACM Symposium on Theory of Computing, STOC '75, 224–233. Association for Computing Machinery, New York, NY, USA, 1975. ISBN 9781450374194. http://doi.org/10.1145/800116.803772.

[16] Song, X.; Jüttler, B.; Poteaux, A.: Hierarchical spline approximation of the signed distance function. In 2010 Shape Modeling International Conference, 241–245, 2010. http://doi.org/10.1109/SMI.2010.18.

[17] Sud, A.; Govindaraju, N.; Gayle, R.; Manocha, D.: Interactive 3D distance field computation using linear factorization. I3D '06, 117–124. Association for Computing Machinery, New York, NY, USA, 2006. ISBN 159593295X. http://doi.org/10.1145/1111411.1111432.

[18] Tagliasacchi, A.; Delame, T.; Spagnuolo, M.; Amenta, N.; Telea, A.: 3d skeletons: A state-of-the-art report. Computer Graphics Forum, 35(2), 573–597, 2016. http://doi.org/https://doi.org/10.1111/cgf.12865.

[19] Takikawa, T.; Litalien, J.; Yin, K.; Kreis, K.; Loop, C.; Nowrouzezahrai, D.; Jacobson, A.; McGuire, M.; Fidler, S.: Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021, 11358–11367. Computer Vision Foundation / IEEE, 2021. https://openaccess.thecvf.com/content/CVPR2021/html/Takikawa_Neural_Geometric_Level_of_Detail_Real-Time_Rendering_With_Implicit_3D_CVPR_2021_paper.html.

[20] Wright, D.: Dynamic occlusion with signed distance fields. In Advances in Real-Time Rendering in Games. Epic Games (Unreal Engine), SIGGRAPH, 2015.

[21] Wu, J.; Kobbelt, L.: Piecewise linear approximation of signed distance fields. In T. Ertl, ed., 8th International Fall Workshop on Vision, Modeling, and Visualization, VMV 2003, München, Germany, November 19-21, 2003, 513–520. Aka GmbH, 2003.

[22] Yap, C.K.: An $O(n \log n)$ algorithm for the voronoi diagram of a set of simple curve segments. Discrete Comput. Geom., 2(4), 365–393, 1987. ISSN 0179-5376. http://doi.org/10.1007/BF02187890.