Annales Mathematicae et Informaticae
56 (2022) pp. 47–57
DOI: https://doi.org/10.33039/ami.2022.12.013
URL: https://ami.uni-eszterhazy.hu

Unbounding discrete oriented polytopes

Mátyás Kiglics*, Gábor Valasek, Csaba Bálint*

Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary
kiglics@caesar.elte.hu
valasek@inf.elte.hu
csabix@inf.elte.hu

Abstract. We propose an efficient algorithm to compute k-sided unbounding discrete oriented polytopes (k-UDOPs) in arbitrary dimensions. These convex polytopes are constructed for a fixed set of directions and a given center point. The interior of k-UDOPs does not intersect the scene geometry. We discuss several types of general geometric queries on these constructs, such as intersection with rays, and provide an empirical investigation on the limit of these shapes as the number of sides increases. In the 2D case, we extend our construction to planar shapes enclosed by arbitrary parametric boundaries with known derivative bounds.

Keywords: computer graphics, computational geometry, collision avoidance

AMS Subject Classification: 68U05

1. Introduction

Bounding volumes are ubiquitous in various computing venues, such as computer graphics [6], collision detection [2, 3, 7, 9], and geometric information systems.

A $B \subset \mathbb{R}^D$ volume is a bounding volume of a $G \subset \mathbb{R}^D$ geometry if $G \subset B$ holds. This property facilitates quick filtering of geometries, in other words, we only execute a query on G if it is successful on B. For example, if a line does not intersect B, it cannot intersect G.

The more efficiently a query is carried out on B, the more performance may be gained by using culling based on bounding volumes. However, B has to be a

*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.

Submitted: November 6, 2022
Accepted: December 3, 2022
Published online: December 28, 2022

sufficiently close approximation to the shape of G to avoid an excessive amount of false positives. In practice, bounding volumes are also organized into hierarchies [6], a construct that only depends on the structure of the initial bounding volumes, not the geometries they contain.

Typical realization of bounding volumes are axis-aligned bounding boxes (AABBs) and oriented bounding boxes (OBBs). Geometric queries, for example, ray-surface intersection and collision detection against other similar volumes, are trivially resolved on these at the expense of their relatively low capacity for adapting to the shape and orientation of their enclosed geometries. These properties are improved by generalizing bounding boxes to the k-sided bounding discrete oriented polytope (k-DOP). The k-DOPs are defined as the intersection of k half-spaces; as such, they are convex. As k increases, the bounding volume can better adapt to the source geometry. However, it also incurs additional processing costs upon filtering geometries, as we must process more faces. In this sense, the choice of k is a trade-off between adaptivity and query complexity.

Unbounding volumes are complements to bounding volumes. They enclose empty spaces such that none of their interior points intersect any geometry. The most prominent example of such a construct is Hart's sphere tracing [5] algorithm that infers unbounding spheres from signed distance values to accelerate ray tracing.

In collision detection and path planning, these unbounding volumes can be used to reject geometries that cannot intersect a given entity. In this case, an unbounding geometry with a smaller volume generates fewer candidates in the filtering pass; thus, it also decreases the number of false positives.

Section 2 describes an efficient algorithm to compute k-sided unbounding convex oriented polytopes, or k-UDOPs, for a wide range of geometry types. The construction runs in $\Theta(kN)$ complexity for N objects and is generalized to higher dimensions. Our algorithm relies on the capability to compute the distance of the discussed geometries to hyperplanes. Section 3 enumerates several simple geometric representations and how to compute these distances on them. In particular, Section 3.5 describes a method to infer a conservative k-UDOP for shapes with arbitrary parametric boundaries in the plane, given known bounds on their derivatives, and in Section 4, we describe some alogorithms applied on k-UDOPs. In Section 5, we demonstrate that the k-UDOP converges to a polygon empirically as the number of sides increases. Finally, we demonstrate the results of our proposed algorithm on various plane geometries in Section 5.

2. Unbounding k-DOP construction

2.1. Representation

A k-DOP, or a discrete oriented polytope with k sides in $2 \leq D \in \mathbb{N}$ dimensions, is defined by a center point $\boldsymbol{c} \in \mathbb{R}^D$, and a sequence of directions $\boldsymbol{v}_i \in \mathbb{R}^D$, $\|\boldsymbol{v}_i\|_2 = 1$ and distances $0 \leq h_i \in \mathbb{R}$, where $i = 1, \ldots, k$ and $k \geq D + 1$. The interior of the

k-DOP is given by

$$H_{\boldsymbol{c}} = \{\boldsymbol{x} \in \mathbb{R}^D \mid \forall i \in \{1, \ldots, k\} : (\boldsymbol{x} - \boldsymbol{c})^{\mathrm{T}} \cdot \boldsymbol{v}_i < h_i\}$$

For even k values, one may arrange the directions such that $\boldsymbol{v}_{2i} = -\boldsymbol{v}_{2i+1}$. This makes k-DOP queries more efficient, essentially halving the number of necessary evaluations. Note that sometimes the literature uses this convention, that is, a k-DOP refers to a $2k$ sided oriented convex polytope. In our constructs, k denotes the number of sides.

In this paper, we consider k and the $\boldsymbol{v}_i$ directions fixed and investigate the problem of finding the largest *unbounding* k-DOP around a point $\boldsymbol{c}$ that does not contain any point from a predefined set of geometries $A \subset \mathbb{R}^D$, that is $A \cap H_{\boldsymbol{c}} = \emptyset$. Note that $H_{\boldsymbol{c}}$ is convex, making intersection tests highly efficient. For example, Algorithm 2 is an $\Theta(k)$ algorithm for computing intersection with a ray.

2.2. Algorithm

We present a generalized method for constructing k-UDOPs for a fixed center point and directions. Let us consider a two-dimensional scene S, consisting of arbitrary geometric entities with a known evaluation of the distance-to-hyperplane query. Our method (Algorithm 1) is summarized as follows.

Algorithm 1 Constructing general k-UDOPs

Input: $\mathbf{c}$ center point, S set of geometries, $\mathbf{v}_i$ directions
Output: h_i distances
$h_i \leftarrow \infty$ ▷ $1 \leq i \leq k$
for all $g \in S$ **do**
  for all $\mathbf{v}_i$ **do**
    $d_i \leftarrow \min\{(\mathbf{p} - \mathbf{c})^{\mathrm{T}} \cdot \mathbf{v}_i \mid \mathbf{p} \in g\}$ ▷ distance of g from line containing $\mathbf{c}$
  end for
  if $h_i > d_i \quad \forall i \in \{j \mid d_j > 0\}$ **then** ▷ g is inside
    $m \leftarrow$ index of $\max(d_1, d_2, \ldots, d_k)$
    $h_m \leftarrow d_m$
  end if
end for

For every shape $g \in S$, we need to calculate the signed d_i distances, that is, the dot product of the $\boldsymbol{v}_i$ directions and $\boldsymbol{p} - \boldsymbol{c}$ vectors, where $\boldsymbol{p}$ is the point of g with the smallest Euclidean distance from the hyperplane defined by $\boldsymbol{c}$ and $\boldsymbol{v}_i$. The calculations of these distances are detailed in Section 3. Using these distances, we can separate the directions for which g overlaps the k-UDOP, and if that is the case for at least one $\boldsymbol{v}_i$, we overwrite h_m with the largest distance d_m. After iterating through every g shape, the $h_1, h_2, \ldots, h_k$ distances represent a single k-UDOP that does not overlap with any g. The construction method is visualized in Fig. 1, and some example scenes are presented in Fig. 2 and Fig. 3.

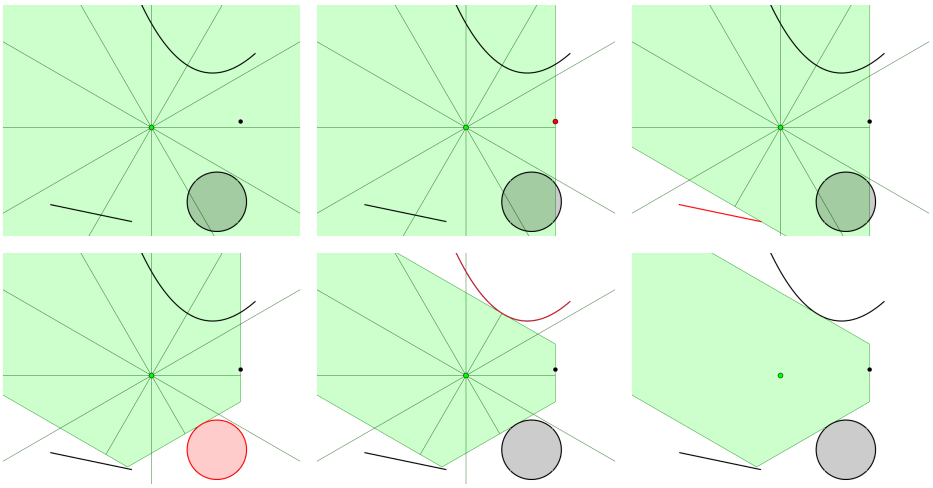


Figure 1. Construction of an unbunding 12-DOP polytope about a point (in green) to a scene containing a point, a line segment, a circle, and a quadratic Bézier curve. All h_i distances along $\boldsymbol{v}_i$ directions (thin lines) are initialized to $+\infty$ (top-left). Then at each iteration, we select a geometry (in red) and adjust the distances along all directions that have a positive dot product with the vector from the center point to the closest point of the selected geometry. The final k-UDOP have less sides than the number of directions it have started with (bottom-right).

This algorithm is linear in the number of entities for a fixed k, so its complexity is $\Theta(Nk)$, where $|S| = N$. The algorithm can be applied in higher dimensions, assuming we can evaluate the necessary signed distances.

3. Distance computations

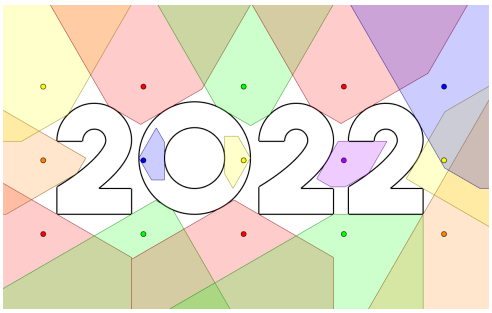
This section summarizes signed distance computations between a hyperplane and elementary geometric shapes.

For a fixed $\boldsymbol{x}_0$ and $\boldsymbol{v}_i$ direction, let $d(\boldsymbol{x})$ denote the signed distance between $\boldsymbol{x}$ and the hyperplane passing through $\boldsymbol{x}_0$ with normal $\boldsymbol{v}_i$, that is, $d(\boldsymbol{x}) = (\boldsymbol{x}-\boldsymbol{x}_0)^{\mathrm{T}} \cdot \boldsymbol{v}_i$, $\|\boldsymbol{v}_i\|_2 = 1$.

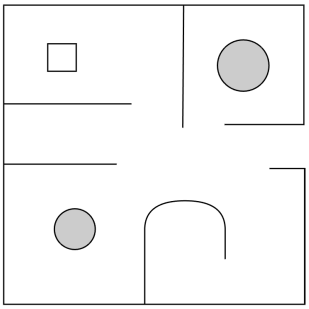
Let there be given an $\boldsymbol{x}_0$ region center and a unit normal $\boldsymbol{v}_j$ and let L_j denote the hyperplane that passes through $\boldsymbol{x}_0$ with normal $\boldsymbol{v}_j$, that is, $L_j = \{\boldsymbol{x} \in \mathbb{R}^D \mid (\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \cdot \boldsymbol{v}_j\}$.

3.1. Points

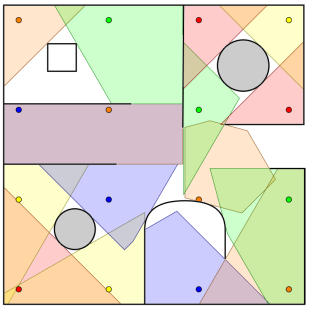
The distance from a point $\boldsymbol{x}$ is computed as $(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \cdot \boldsymbol{v}_j$.



(a) Unbounding 8-DOPs fitted to a text with a TrueType font. The boundary curves are composed of linear and quadratic polynomial segments.



(b) A stylized floor-plan composed of line segments, quadratic Bézier curves and circles.



(c) Fitting unbounding 12-DOPs to the floorplan of Fig. 2b.

Figure 2. Test scenes used in our deterministic tests. Figures 2a and 2c illustrate a sparse 3×5 and 4×4 grid of center points and the corresponding 15 and 16 k-UDOPs of the respective scenes.

3.2. Line segments and polygons

A line segment between $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^D$ is parametrized as $\boldsymbol{p}(t) = \boldsymbol{a} + t(\boldsymbol{b} - \boldsymbol{a})$, $t \in [0, 1]$. The smallest distance between L_j and $\boldsymbol{p}(t)$ is then either at their intersection point at $t = -\frac{(\boldsymbol{a}-\boldsymbol{x}_0)^{\mathrm{T}} \cdot \boldsymbol{v}_j}{(\boldsymbol{b}-\boldsymbol{a})^{\mathrm{T}} \cdot \boldsymbol{v}_j}$, if $\boldsymbol{a} \neq \boldsymbol{b}$ and $t \in [0, 1]$, or the smallest of $|d(\boldsymbol{a})|$ and $|d(\boldsymbol{b})|$.

The distance of an n-sided polygon is resolved by taking the smallest distance between L_j and the polygon edges.

3.3. Bézier curves

Let $\boldsymbol{b}(t) = \sum_{i=0}^{n} \boldsymbol{b}_i B_i^n(t)$, $t \in [0, 1]$ denote a degree n Bézier curve, where $\boldsymbol{b}_i \in \mathbb{R}^D$, $i = 1, \ldots, n$ are control points and $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$ are the Bernstein polynomials.

The smallest distance is either realized at a $t^* \in [0, 1]$ parameter or at one of

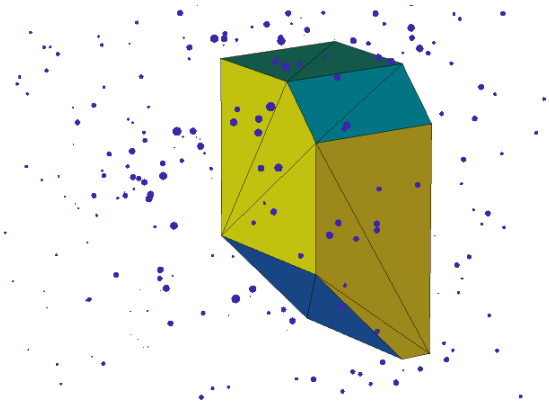


Figure 3. A 3D generalization of Algorithm 1 to a point cloud.

the curve endpoints $\boldsymbol{b}_0$ or $\boldsymbol{b}_n$. Since

$$\begin{aligned} d(\boldsymbol{b}(t)) &= (\boldsymbol{b}(t) - \boldsymbol{x})^{\mathrm{T}} \cdot \boldsymbol{v}_j \\ &= \left(\sum_{i=0}^{n} B_i^n(t) \boldsymbol{b}_i - \sum_{i=0}^{n} B_i^n(t) \boldsymbol{x} \right)^{\mathrm{T}} \cdot \boldsymbol{v}_j \\ &= \sum_{i=0}^{n} B_i^n(t) \underbrace{(\boldsymbol{b}_i - \boldsymbol{x})^{\mathrm{T}} \cdot \boldsymbol{v}_j}_{d_i} = \sum_{i=0}^{n} B_i^n(t) d_i \ , \end{aligned}$$

the parameters of the closest points on the curve satisfy

$$\partial_t d(\boldsymbol{b}(t)) = n \sum_{i=0}^{n-1} B_i^{n-1}(t) \Delta d_i = 0 \ ,$$

using the notation $\Delta^k d_i = \Delta^{k-1} d_{i+1} - \Delta^{k-1} d_i$, $k \geq 1$, $i = 0, \ldots, n-k$ and the convention $\Delta^0 d_i = d_i$.

In case of quadratic Bézier curves, the solution is

$$t = -\frac{\Delta d_0}{\Delta^2 d_0} \ ,$$

as long as $\Delta^2 d_0 \neq 0$ and t is in $[0, 1]$ interval. The closest distance is then realized either at $\boldsymbol{b}_0$, $\boldsymbol{b}_2$, or $\boldsymbol{b}(t)$ between the line and the Bézier curve.

For cubic Bézier curves, we can use the Bernstein form of the quadratic formula to obtain the two roots as

$$t_{1,2} = \frac{-\Delta d_0 \pm \sqrt{d_1^2 - d_0 d_2}}{\Delta^2 d_0} \ .$$

The minimum distance is then at either one of the roots that lie in $[0, 1]$ or at one of the endpoints.

Note that the convex hull property of Bézier curves [4] allows us to approximate the exact distance to $\boldsymbol{b}(t)$.

3.4. Spheres

The signed distance between L_j and a sphere with center $\boldsymbol{c} \in \mathbb{R}^D$ and radius $r > 0$ is $(\boldsymbol{c} - \boldsymbol{x})^{\mathrm{T}} \cdot \boldsymbol{v}_j - r$.

3.5. Shapes with continuous parametric boundaries

Let us consider the plane only and address the case of shapes that have sufficiently many times continuously differentiable boundaries, parametrized by some $\boldsymbol{p}(t) : [a, b] \to \mathbb{R}^2$ mapping. We devise conservative bounds on the distance between the line L_j and $\boldsymbol{p}(t)$ given a bound on the magnitude of the appropriate derivatives of $\boldsymbol{p}(t)$.

First, we construct a piecewise polynomial approximation to the boundary to achieve this. Afterward, we compute the distance of L_j to these polynomial boundary approximations, as shown in Section 3.3. Finally, using the error term of the particular approximating polynomial, we decrease the computed distance.

Geometrically, the last step uses a distance lower bound to the offset of the polynomial approximation. The key insight is that we do not explicitly represent the offset of the parametric boundary; it is sufficient to apply it in distance space [1].

Let us consider the case of order k Hermite interpolation. Let $\boldsymbol{h}_k(t)$ denote the Hermite polynomial that interpolates $\boldsymbol{p}^{(l)}(t_k)$, $l = 0, \ldots, k$ at prescribed knots t_k, where $\boldsymbol{p}^{(l)}(t)$ denotes the l-th derivative at t. Then

$$\boldsymbol{p}(t) - \boldsymbol{h}_k(t) = \frac{\boldsymbol{p}^{(k+1)}(\xi)}{(k+1)!}(t - t_k)^{k+1}(t_{k+1} - t)^{k+1}$$

holds for some $\xi \in (t_k, t_{k+1})$. If $M > 0$ is a bound on $\|\boldsymbol{p}^{(k+1)}\|_\infty$, then the right hand side of

$$\|\boldsymbol{p} - \boldsymbol{h}_k(t)\|_\infty \leq \underbrace{\frac{M}{(k+1)!}\left|\frac{t_{k+1} - t_k}{2}\right|^{2k}}_{E_k}$$

provides the maximum deviation between the polynomial approximation and the original shape. Computing the Hermite interpolation in Bernstein basis is trivial [4], and subtracting $\sqrt{2} \cdot E_k$ from the distance between L_j and the polynomial approximation gives a conservative bound on the distance between L_j and the segment of $\boldsymbol{p}(t)$ between t_k and t_{k+1}.

4. Queries on *k*-UDOPs

4.1. Converting *k*-UDOPs to polytope mesh

To compute the vertices of the D-dimensional polytope, we have to find the intersection of all possible combinations of D planes. This produces $\binom{k}{D}$ vertices that could be part of the polytope, so we have to filter out those that are not.

Given two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^D$, we can compute the $\boldsymbol{v} = \alpha \cdot \boldsymbol{a} + \beta \cdot \boldsymbol{b} \in \mathbb{R}^D$ vector that has perpendicular difference vectors to $\boldsymbol{a}$ and $\boldsymbol{b}$, with the following deltoid [8] formula:

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \frac{1}{\boldsymbol{a}^{\mathrm{T}}\boldsymbol{a} \cdot \boldsymbol{b}^{\mathrm{T}}\boldsymbol{b} - \boldsymbol{a}^{\mathrm{T}}\boldsymbol{b} \cdot \boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}} \begin{bmatrix}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b} \cdot (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{a} - \boldsymbol{a}^{\mathrm{T}}\boldsymbol{b})\\ \boldsymbol{a}^{\mathrm{T}}\boldsymbol{a} \cdot (\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b} - \boldsymbol{a}^{\mathrm{T}}\boldsymbol{b})\end{bmatrix}, \tag{4.1}$$

where the vertex is obtained with $\boldsymbol{v} = \alpha \cdot \boldsymbol{a} + \beta \cdot \boldsymbol{b}$.

Equation (4.1) allows the computation of multiple intersections in parallel. Applying the formula $D-1$ times to D directions yields a single vertex, which is then tested against the boundary of the k-DOP. Thus, in general, the algorithm is slow $O(k^{D+1})$. In two dimensions, these steps can be simplified to be $O(k^2)$ because directions have a circular ordering.

Once the vertices are computed, the connectivity information of the vertices may be computed by running a D-dimensional convex hull algorithm.

Algorithm 2 Ray and k-DOP intersection

Input: $\mathbf{c}$ center, $\mathbf{v}_i$ directions, h_i distances, $\boldsymbol{p}_0 + t\boldsymbol{d}$ ray
Output: t_1, t_2 intersection parameters
$t_1 \leftarrow -\infty$, $t_2 \leftarrow +\infty$
for all $\mathbf{v}_i$ **do**
  $t \leftarrow \dfrac{(\boldsymbol{p}_0 - \boldsymbol{c})^{\mathrm{T}} \cdot \boldsymbol{v}_i}{\boldsymbol{d}^{\mathrm{T}}\boldsymbol{v}_i}$; ▷ Intersect with each plane
  if $\boldsymbol{d}^{\mathrm{T}}\boldsymbol{v}_i < 0$ **then** ▷ Is plane back-facing?
    $t_1 \leftarrow \max(t_1, t)$ ▷ Keep furthest back-facing
  else
    $t_2 \leftarrow \min(t_2, t)$ ▷ Keep closest front-facing
  end if
end for
If $t_1 < t_2$ then there is an intersection

4.2. Ray intersection

Intersecting a k-UDOP with a ray in two dimensions can be reformulated as a ray-convex polygon intersection problem once the k-UDOP is converted to a polygon, as shown in Section 4.1. Even though the subsequent intersection computation may be carried out in $\mathcal{O}(\log k)$ time, it does not generalize to higher dimensions

and involves a quadratic time conversion. Pre-computing the polygons mitigates the latter; however, it may double the storage for each k-UDOP.

Instead, a linear time ray intersection algorithm may be formulated directly on our k-UDOP representation, shown in Algorithm 2.

The main idea is to divide the half planes into two groups: front-facing ($\boldsymbol{d}^{\mathrm{T}}\boldsymbol{v}_i > 0$), and back-facing ($\boldsymbol{d}^{\mathrm{T}}\boldsymbol{v}_i < 0$). We have to find the smallest t solution amongst the front-facing (t_2) and the largest t solution for the back-facing (t_1) planes. If, and only if, the k-DOP is intersected, then $t_1 < t_2$ and hence for any $t \in [t_1, t_2]$, the segment $\boldsymbol{x} = \boldsymbol{p}_0 + t\boldsymbol{d}$ is within the k-DOP.

4.3. Bounding k-DOP containment test

For collision detection, we would like to also know if a k-UDOP intersects with another k-DOP. For this, let the k-UDOP be defined by a $\boldsymbol{c}_1 \in \mathbb{R}^D$ center, $\boldsymbol{v}_i \in \mathbb{R}^D, \|\boldsymbol{v}_i\|_2 = 1$ directions and $h_i > 0$ distances, and the k-DOP be defined by the same $\boldsymbol{v}_i \in \mathbb{R}^D$ directions but with a $\boldsymbol{c}_2 \in \mathbb{R}^D$ center and $g_i >$ distances. Then, the k-DOP is inside the k-UDOP if the distance vector between the centers ($\boldsymbol{c}_2 - \boldsymbol{c}_1$) projected onto each $\boldsymbol{v}_i$ is less than the difference between the k-DOP distances ($h_i - g_i$). Algorithm 3 summarizes the above and allows efficient utilization of k-UDOP acceleration structures for collision detection tasks in any dimension.

Algorithm 3 Bounding k-DOP containment test

Input: $\boldsymbol{c}_1, \boldsymbol{c}_2$ centers, $\mathbf{v}_i$ common directions, h_i, g_i distances
Output: True only if first k-DOP contains the second
for all $\mathbf{v}_i$ **do**
  if $(\boldsymbol{c}_2 - \boldsymbol{c}_1)^{\mathrm{T}} \cdot \boldsymbol{v}_i \geq h_i - g_i$ **then**
    return ***false***
  end if
end for
return ***true***

5. Test results

We observed that the k-DOP does not always utilize all sides, so the generated polygon often has less than k number of edges. To find a reasonable choice of k, we generated random points around $\boldsymbol{c}$ from different distributions and measured various metrics of the resulting k-DOP. We tested 50 different scenes with k increasing from 3 to 300. The means of the various metrics are shown in Table 1.

We also noticed that as we increase k, the k-UDOP shape stabilizes, as if a fixed point solution was found. To verify this, we have taken the symmetric difference of the polygons of consecutive pairs of k-DOPs and measured the average difference of its area. Generally, the difference converged to zero; however, the area fluctuated even for large k values in a few cases. This seems to have been caused by empty

Table 1. Average values of different metrics measured from 50 different set of points for every k value between 3 and 300.

k	**Perimeter**	**Area**	**# of sides**	$\frac{\#\text{ of sides}}{k}$
3	3.217	0.911	3.00	100%
4	3.669	1.070	4.00	100%
5	3.896	1.194	4.42	88.4%
6	3.820	1.165	4.84	80.7%
7	4.018	1.242	4.96	70.9%
8	3.952	1.232	5.06	63.3%
9	3.906	1.264	5.14	57.1%
10	3.757	1.149	5.34	53.4%
11	3.875	1.232	5.70	51.8%
12	3.925	1.233	5.72	47.7%
⋮	⋮	⋮	⋮	⋮
298	4.033	1.279	15.74	5.28%
299	4.042	1.280	15.92	5.32%
300	4.024	1.266	15.86	5.29%

areas in-between the clusters of generated points, which placed at least one of the k-DOP sides very far from $\boldsymbol{c}$ for certain k values but was cut off in other cases. The results of the different tests are visualized in Fig. 4.

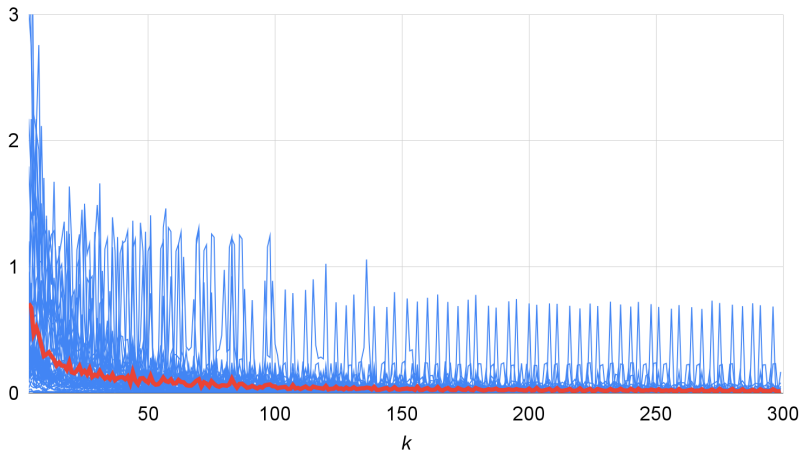


Figure 4. Measured convergence of consecutive polygons for 50 random cases (blue lines) and their average (red). This difference is measured in the area of the symmetric difference of the polygons generated from k and $k+1$-DOPs.

The convergence was expected, since if $\boldsymbol{w}_j$ are directions towards each object's closest point to $\boldsymbol{c}$, then drawing perpendicular lines through the footpoints to each $\boldsymbol{w}_j$, we obtain the convex polygon that the algorithm seems to approach. This is

because at least one of the directions $\boldsymbol{v}_i$ will generally get closer to each $\boldsymbol{w}_j$ direction, so the algorithm will choose a corresponding line that is almost perpendicular to $\boldsymbol{w}_j$.

6. Conclusions

We presented a simple and efficient algorithm to compute D-dimensional k-UDOPs for a prescribed position and a set of k fixed directions.

In the plane, we showed that the resulting convex polygon converges to a fixed shape empirically, whose number of effective sides stayed within 16, even for $k = 300$. As such, large k figures function more to orient the resulting k-UDOP.

Additionally, we presented conversion algorithms to polytope meshes in arbitrary dimensions for both bounding and k-UDOPs and direct ray intersection and bounding k-DOP containment tests.

References

[1] C. Bálint, G. Valasek, L. Gergó: *Operations on Signed Distance Functions*, Acta Cybernetica 24.1 (May 2019), pp. 17–28, doi: https://doi.org/10.14232/actacyb.24.1.2019.3, url: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/4004.

[2] S. Dinas, J. M. Bañón: *A literature review of bounding volumes hierarchy focused on collision detection*, Ingeniería y competitividad 17.1 (2015), pp. 49–62.

[3] C. Ericson: *Real-time collision detection*, CRC Press, 2004.

[4] G. Farin: *Curves and Surfaces for Computer Aided Geometric Design (3rd Ed.): A Practical Guide*, San Diego, CA, USA: Academic Press Professional, Inc., 1993, isbn: 0-12-249052-5.

[5] J. Hart: *Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces*, The Visual Computer 12 (June 1995), doi: https://doi.org/10.1007/s003710050084.

[6] T. L. Kay, J. T. Kajiya: *Ray Tracing Complex Scenes*, in: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '86, New York, NY, USA: Association for Computing Machinery, 1986, pp. 269–278, isbn: 0897911962, doi: https://doi.org/10.1145/15922.15916.

[7] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, K. Zikan: *Efficient collision detection using bounding volume hierarchies of k-DOPs*, IEEE Transactions on Visualization and Computer Graphics 4.1 (1998), pp. 21–36, doi: https://doi.org/10.1109/2945.675649.

[8] G. Valasek, C. Bálint, A. Leitereg: *Footvector Representation of Curves and Surfaces*, Acta Cybernetica (Aug. 2021), doi: https://doi.org/10.14232/actacyb.290145, url: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/4205.

[9] G. Zachmann: *Rapid collision detection by dynamically aligned DOP-trees*, in: Proceedings. IEEE 1998 Virtual Reality Annual International Symposium (Cat. No.98CB36180), 1998, pp. 90–97, doi: https://doi.org/10.1109/VRAIS.1998.658428.