Computation of the ϕ-Descriptor in the Case of 2D Vector Objects

Jason Kemp, Tyler Laforet and Pascal Matsakis
School of Computer Science, University of Guelph, Stone Rd E, Guelph, Canada

Keywords: Image Descriptors, Relative Position Descriptors, ϕ-Descriptor, Spatial Relationships, Vector Data.

Abstract: The spatial relations between objects, a part of everyday speech, are capable of being described within an image via a Relative Position Descriptor (RPD). The ϕ-descriptor, a recently introduced RPD, encapsulates more spatial information than other popular descriptors. However, only algorithms for determining the ϕ-descriptor of raster objects exist currently. In this paper, the first algorithm for the computation of the ϕ-descriptor in the case of 2D vector objects is introduced. The approach used is based on the concept of Points of Interest (which are points on the boundaries of the objects where elementary spatial relations change) and dividing the objects into regions according to their corresponding relationships. The capabilities of the algorithm have been tested and verified against an existing ϕ-descriptor algorithm for raster objects. The new algorithm is intended to show the versatility of the ϕ-descriptor.

1 INTRODUCTION

A Relative Position Descriptor (RPD) provides a quantitative description of the spatial position of two image objects with respect to each other (Matsakis et al., 2015). It can be used, for example, to give a quantitative answer to the question: “Is object A *to the left of* object B?”. The descriptor may indicate whether A is exactly, not at all, or to some degree to the left of B. There are many RPD algorithms designed for raster objects, i.e., sets of pixels (2D case) or voxels (3D case). There are significantly fewer RPD algorithms designed for vector objects, i.e., sets of vertices defining polygons (2D case) or polyhedra (3D case).

The force histogram (Matsakis and Wendling, 1999) might be the best known RPD and has found many applications. Algorithms exist to compute the force histogram in the case of 2D raster objects, 2D vector objects, 3D raster objects and 3D vector objects. In this paper, we focus on another, more recent RPD: the ϕ-descriptor (Matsakis et al., 2015) which encapsulates much more spatial relationship information than the force histogram, and appears to be much more powerful (Francis et al., 2018). Currently, there only exists an algorithm for calculating the ϕ-descriptor of 2D raster objects, and there is no algorithm for calculating the ϕ-descriptor of 2D vector objects.

Here, we introduce the first algorithm to calculate the ϕ-descriptor of 2D vector objects. Background information is provided in Section 2. The new algorithm is presented in Section 3 and tested in Section 4. Conclusions and future work are in Section 5.

2 BACKGROUND

Spatial relations can be categorized as directional, topological, and distance. Freeman (1975) was the first to suggest that spatial relations could be used to model relative position. He also suggested that a fuzzy system be used to allow for these relations to be given a degree of truth. Miyajima and Ralescu (1994) proposed that an RPD could be created to encapsulate spatial relationship information. An RPD provides a quantitative description of the spatial relations between two image objects. RPDs are visual descriptors like color, texture and shape descriptors. Symbol recognition (Santosh et al., 2012), robot navigation, geographic information systems, linguistic scene description, and medical imaging are some areas that benefit from RPDs (Naeem and Matsakis, 2015).

Existing work in RPDs is explored in Sections 2.1, 2.2, and 2.3, and methods to compare how similar two ϕ-descriptors are is explored in Section 2.4.

Kemp, J., Laforet, T. and Matsakis, P.
Computation of the -Descriptor in the Case of 2D Vector Objects.
DOI: 10.5220/0008984500600068
In *Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2020)*, pages 60-68
ISBN: 978-989-758-397-1; ISSN: 2184-4313

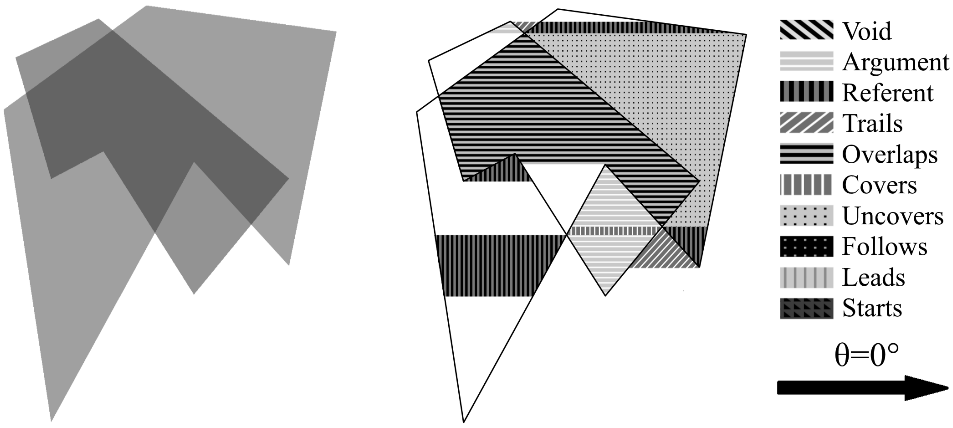

Figure 1: On the left, the original image with two intersecting objects. Areas where the objects overlap are shown in dark grey. On the right, that image divided into ϕ-regions in the horizontal left-to-right direction θ. Regions without a corresponding ϕ-group are left blank.

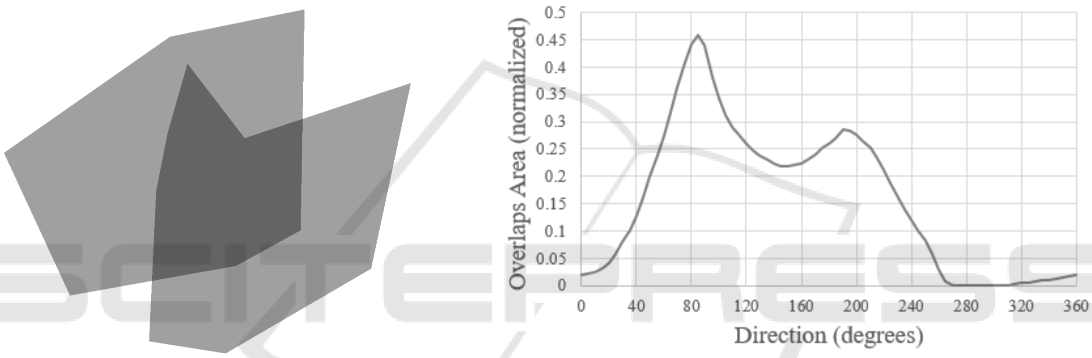

Figure 2: On the left, an example pair of objects. Areas where the objects overlap are shown in dark grey. On the right, a histogram of the area corresponding to the *overlaps* relation between the two objects in all directions. Area has been normalized to a percentage of the Region of Interaction, as explained in Sections 2.3.2 and 4.

2.1 Existing RPDs

Many RPDs exist, and have built upon each other over time. The angle histogram might be the first true RPD. It is able to model directional relations such as “to the right of”, “above” etc, (Keller and Wang, 1995) (Keller and Wang, 1996) (Krishnapuram et al., 1993) (Miyajima and Ralescu, 1994). The force histogram generalizes the angle histogram, and also improves upon it (Matsakis and Wendling, 1999). While the angle and force histograms can model directional relations, there are others such as the R Histogram and the Allen Histogram that can model distance or topological relations (Naeem and Matsakis, 2015). The recently introduced ϕ-descriptor (Matsakis et al., 2015) is the first RPD able to model a variety of directional, topological, and distance relations (Francis et al., 2018). In other words, it can describe a larger set of relations than any other existing RPD. A more in-depth review of RPDs is available in (Naeem and Matsakis, 2015).

2.2 The Force Histogram

The force histogram sees objects as physical plates and determines the sums of forces (like gravitational forces) exerted by the particles of one object on the particles of another object. A sum is calculated for each direction in space. There are algorithms to calculate the force histogram in the case of 2D vector objects (Recoskie et al., 2012), as well as 3D vector objects (Reed et al., 2014). In the 2D case, horizontal lines partition the objects into trapezoidal pieces. The areas of these pieces are determined and used to calculate the value of the descriptor in the horizontal direction. The objects are then rotated to the next direction in a direction set, and the process is repeated for each direction. In the 3D case, planes are used to intersect the 3D objects into 2D objects, and the 2D algorithm is applied to the resulting objects.

2.3 The Phi-descriptor

For each direction in a direction set, the ϕ-descriptor algorithm divides the objects into subregions (ϕ-regions). Each ϕ-region corresponds to an elementary spatial relationship in that direction. The area and height of each ϕ-region are determined, and the values that correspond to the same group of relationships (ϕ-group) are summed. The process is repeated in all directions to form the complete ϕ-descriptor (see Figures 1 and 2). The descriptor can be conceptualized as a set of histograms, one for each ϕ-group, with the directions as bins. The value in each bin is an area or height.

2.3.1 Phi-groups

An important aspect of this algorithm we refer to as ϕ-*groups*. These are groups of elementary spatial relations, each group sharing some characteristic described by the verb naming the group. For example, the *overlaps* group contains all relations where two objects overlap one another. The *trails* group contains all relations where there is a gap between the two objects, i.e. one object trails behind the other. Each group, along with the elementary spatial relations that they consist of, can be seen in Figure 3. One of the core elements of the algorithm is to find regions, ϕ-*regions*, that correspond to a single ϕ-group.

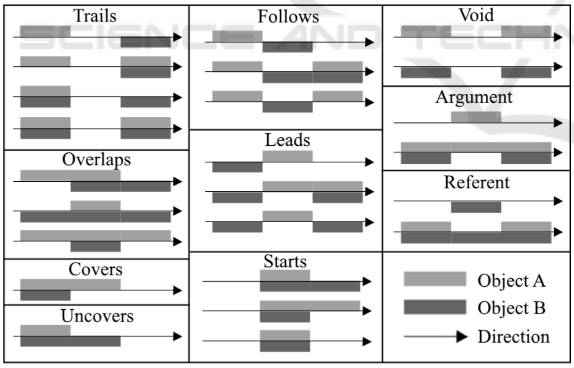

Figure 3: Elementary spatial relationships in a given direction and their corresponding ϕ-groups. Each spatial relationship is represented by horizontal slices of two objects *A* and *B*. Slices for different objects have been split apart for clarity; areas where object *A* is shown stacked on object *B* represent areas where two objects overlap.

2.3.2 ϕ-Descriptor

The descriptor can be seen as a matrix of numbers, a column for each direction and a pair of rows for each ϕ-group, as seen in Table 1. The pair of rows contain areas and heights. Each element of the matrix is the sum of the areas or heights of all ϕ-regions sharing a ϕ-group in a single direction. For example, if there are two regions with the *trails* ϕ-group in direction θ then the cells for *trails* in direction θ would contain one for the sum of the areas, and one for the sum of the heights. In addition to the ϕ-groups, the first two rows of the descriptor show the area and height of the Region of Interaction in each direction (see Section 3.1). These two values are the sum of the areas and heights, respectively, of all ϕ-regions in the direction.

Table 1: An example of part of a ϕ-descriptor. 72 directions were tested for this example. Each row represents an elementary spatial relation (or the Region of Interaction) and the corresponding sums of areas and heights for each tested direction. For example, the sum of the areas of the ϕ-regions corresponding to the *trails* ϕ-group in direction 5° is 10461 (shown below in bolded font).

ϕ-Group \ θ		0°	5°	...
RoI	area	310009	311138	...
	height	516.681	533.891	...
Void	area	404	234	...
	height	202	36.137	...
Arg	area	7769	7819	...
	height	75.062	70.710	...
Ref	area	7863	9200	...
	height	129.966	120.720	...
Trails	area	7481	**10461**	...
	height	124.683	129.641	...
...	...	...	...	...

2.4 Similarity Measures

Two similarity measures, *MinMax* and *SubAdd*, are used to compare the similarity of ϕ-descriptors. These measures have previously been used and detailed in (Matsakis, 2016) to compare ϕ-descriptors of pairs of raster objects. They each require that the descriptors use the same direction set, and that all values are non-negative. The values from one descriptor are compared against their counterpart in the other descriptor; the pair of values $d1(\theta)$ and $d2(\theta)$ would refer to the same element in each descriptor (ex. *trails* area in direction θ).

2.4.1 MinMax Similarity

The *MinMax* of the two descriptors is essentially the ratio between the sum of the minima and the sum of the maxima the pairs of values:

$$\frac{\sum min(d1(\theta), d2(\theta))}{\sum max(d1(\theta), d2(\theta))} \tag{1}$$

The *MinMax* similarity is 1.0 if both descriptors are identical, and lower values indicate less similarity.

2.4.2 SubAdd Similarity

The *SubAdd* of the two descriptors is essentially the ratio between the sum of the absolute differences and the sum of the sums of the pairs of values:

$$1 - \frac{\sum |d1(\theta) - d2(\theta)|}{\sum |d1(\theta) + d2(\theta)|} \quad (2)$$

The *SubAdd* similarity is 1.0 if both descriptors are identical, and lower values indicate less similarity.

3 ALGORITHM

The ϕ-descriptor is built upon the principle that pairs of objects can be divided into regions (ϕ-regions), for each direction in a direction set, and these regions correspond to specific sets of elementary spatial relations (ϕ-groups). The primary goal then of any ϕ-descriptor-based algorithm is to find an appropriate way to create these regions and identify the spatial relations they correspond to. Using the areas and heights of these ϕ-regions, the descriptor can be built in one direction; to complete the ϕ-descriptor, the process is repeated in all directions. The goal of this new algorithm is to calculate the ϕ-descriptor for a pair of vector objects; this process is summarized in Algorithm 1.

Algorithm 1: Vector ϕ-Descriptor Computation.

```
for all θ ∈ direction set do
    Rotate original objects by −θ.
    Find vertices of the inputted objects where elementary spatial relations change, referred to as Points of Interest (PoI).
    for all PoI do {Create Rays}
        Cast horizontal "rays" through the PoI.
        Determine the points where the ray intersects with the objects (endpoints).
        Determine PoI ϕ-groups for directions "left-to-right" and "right-to-left".
        Add edges between endpoints closest to PoI.
    end for
    Divide the objects into ϕ-regions based on the boundaries of the original objects that intersect the rays.
    Calculate areas and heights of ϕ-regions.
    Save them to descriptor.
end for
```

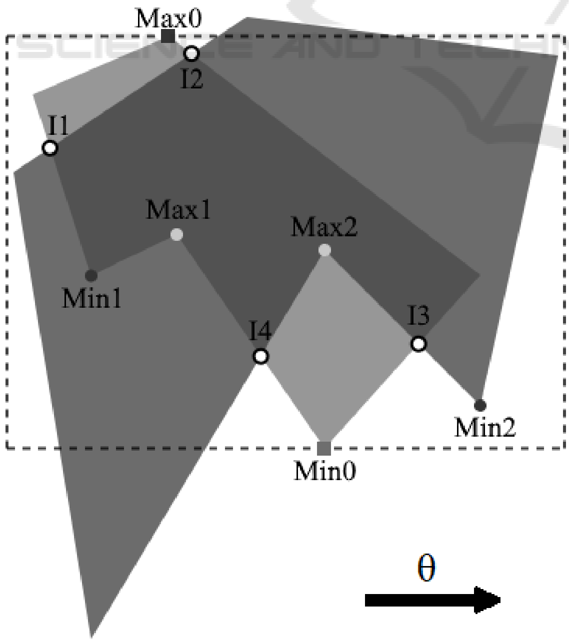

Figure 4: This image shows the Region of Interaction, as a dotted box, in direction θ. The PoI are shown: *Max0* (black square), *Min0* (grey square), Intersections (white circle with black border), Local Minima (black circle), and Local Maxima (grey circle).

3.1 Points of Interest

We focus here on the "left-to-right" direction (or direction θ=0°). For any other direction θ, rotate the objects by −θ beforehand. To find the ϕ-regions we must look at what would make one portion of the objects change to a different elementary spatial relationship. The points in which these changes occur we call Points of Interest (PoI) (see Figure 4) and are confined to the region of interaction, the space which both objects share in direction θ=0°. The top-most and bottom-most points of the region of interaction are both PoI, as well as any local extrema within that region, and any point where the boundaries of the objects intersect.

The *Region of Interaction* is defined by two vertices, *Max0* and *Min0* (the top-most and bottom-most points). The *extrema* are vertices with both neighbours either above (minima), or below (maxima) the vertex. We consider that a pair of vertices can form an *extrema pair* if both vertices share a *Y* value, they are neighbours of each other, and their remaining two neighbours are both either above (minima), or below (maxima) the pair. After a rotation the set of PoI which correspond to these two types may change (i.e. a minima may no longer be a minima after a rotation), and thus they must be recalculated for each direction.

The *intersections* are points where an edge from one object intersects an edge from the other. While the other types of PoI may change after each rotation, the intersections remain the same throughout the algorithm. For this reason, they are calculated only once in a pre-processing step.

3.2 Rays

At each PoI we pass a horizontal ”ray” and determine the locations in which it intersects the edges of the two objects. We call these intersection points *endpoints*. We sort the endpoints left to right, and by counting how many endpoints are on either side of an endpoint we can determine whether it lies on object *A*, *B*, *both* or *neither*; this is the endpoint’s *source*.

Edges are added between the endpoints along the ray. These edges serve to divide the objects into ϕ-regions based on the changing elementary spatial relationships (as defined by the PoI). The endpoints along the top-most and bottom-most rays are all connected by edges, while intersection and extrema rays only have edges added directly adjacent to PoI. An example of the process to create rays is shown in Figure 5 by creating two rays from local minima.

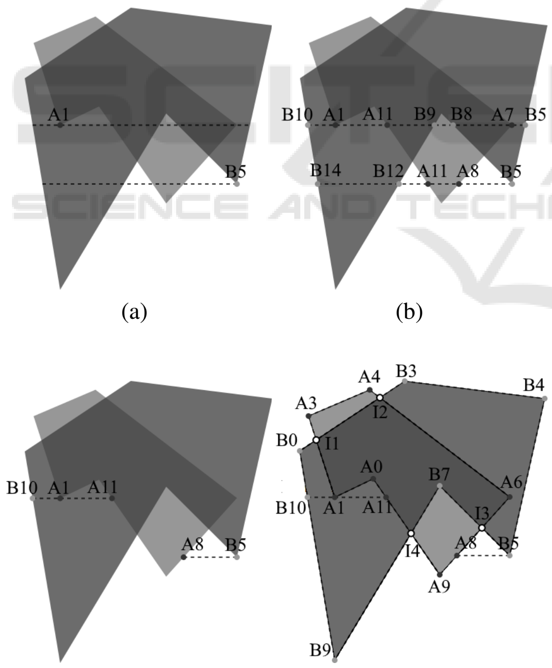

Figure 5: An example creating two rays from local minima. a) Local minima A1 and B5 along with the rays passing through. b) All ray intersections located. c) The edges and endpoints formed by the ray are determined. d) The new edges (dotted-lines) and endpoints added to the original objects.

3.3 ϕ-Regions

While creating the rays, the *sources* of the endpoints were determined, and edges added along the ray. By looking at the sources of an endpoint and its two neighbouring endpoints, we can determine the ϕ-group of the center endpoint. For example, if a point has the *source A*, and the neighbouring *sources* belong to *neither* object, we can know (Figure 3) that the endpoint corresponds to the ϕ-group *Argument*. Once we have identified the ϕ-groups, and all rays have been created, we can begin forming the ϕ-regions. To accomplish this we traverse the edges above each point on a ray in a counter-clockwise direction; the ϕ-region is the smallest cycle above and to the right that contains the starting endpoint on the ray. As we traverse the region, we remove the directed edges that form it; this reduces computation costs for traversing subsequent ϕ-regions.

3.4 ϕ-Region Areas and Heights

The ϕ-descriptor requires the size of each ϕ-region. The area of the ϕ-region is used for the size, as well as the height of the ϕ-region. Algorithm 2 shows the process to find the area of a polygon (ϕ-region) (Page, 2011). To calculate the height of the ϕ-region, take the difference of the maximum and minimum *Y* values of all points in the ϕ-region. After the sizes of all ϕ-regions are determined for a direction θ, we look at each ϕ-region and add the appropriate areas and heights to the θ and θ+180° columns at the two rows associated with the ϕ-groups of the region.

Algorithm 2: Find ϕ-Region Area.

```
numPoints = number of points in polygon[].
area = 0
j = numPoints -1
for i = 0 to i = numPoints do
    area+ = (polygon[j].x + polygon[i].x) * (polygon[j].y − polygon[i].y)
    j = i
end for
area = area / 2
```

The ϕ-descriptor contains the areas and heights of ϕ-regions in multiple directions. The number of directions k must be an even number greater than zero, i.e., $k = 2j$ where $j \in \mathbb{Z}, j \geq 1$. This is because directions are processed in pairs; when a ϕ-group in direction θ is determined the ϕ-group for direction θ+180° is easily determined as well.

The objects are rotated about the centroid of the two objects. We require the centroid of the two objects such that they can rotate around the same origin; this will preserve their relative position. This centroid is calculated once at the start of the algorithm and used for each rotation. The x-coordinate of the centroid, for example, is the average of the x-coordinates of all vertices from both objects.

In addition to rotating, we remove all remaining points and edges we added when creating the rays. We want to return the objects to their original state, i.e., with only the initial vertices, object intersections, and edges on the boundaries of the objects between these vertices and intersections.

Algorithm 3: Rotate Object.

$centroid = (mean(Xvalues), mean(Yvalues))$
for all Point p in object **do**
  $\Delta x = p.x - centroid.x$
  $\Delta y = p.y - centroid.y$
  $p.x = centroid.x + \Delta x * cos\theta - \Delta y * sin\theta$
  $p.y = centroid.y + \Delta x * sin\theta + \Delta y * cos\theta$
end for

4 EXPERIMENTS

The testing plan to validate the new algorithm presented in this paper can be broken into five steps:

1. Generate random vector object pairs.
2. Calculate ϕ-descriptors of vector object pairs using the new algorithm.
3. Convert vector object pairs to raster object pairs.
4. Calculate ϕ-descriptor of raster object pairs using the existing algorithm.
5. Compare descriptors and processing time.

It is expected that a vector object pair and the corresponding rasterized object pair should produce near identical descriptors, and the higher the resolution of the rasterized objects the higher the similarity between the two descriptors. Note, however, that the descriptors must first be "normalized" using the size of the Region of Interaction mentioned in Section 2.3.2. For each direction, each area in the descriptor is divided by the area of the Region of Interaction, and each height is divided by the height of the Region of Interaction. In the end, the normalized descriptors contain percentages, rather than actual size data; for example, "*trails* area in direction θ accounts for 2.1% of the Region of Interaction area". When comparing the descriptors we ask the following question, "Does this region account for the same percentage in both descriptors?"

The random generation of vector object pairs is briefly discussed in Section 4.1, the results of the experiments are presented in Section 4.2, and the limitations with the algorithm are stated in Section 4.3.

4.1 Test Data

Random vector objects pairs were generated using Ounsworth's algorithm (Ounsworth, 2012). A number of random directions is selected. A point is placed at a random distance in each direction, maintaining a clockwise progression. The algorithm contains some variables to have some control over the output, such as the number of directions, the center of the polygon, the "average" distance from the center for the points, the "spikiness" (which determines how much variance there is from the average distance) and the "irregularity" (which controls how even the direction set is). Two examples of randomly generated object pairs are shown in Figure 6.

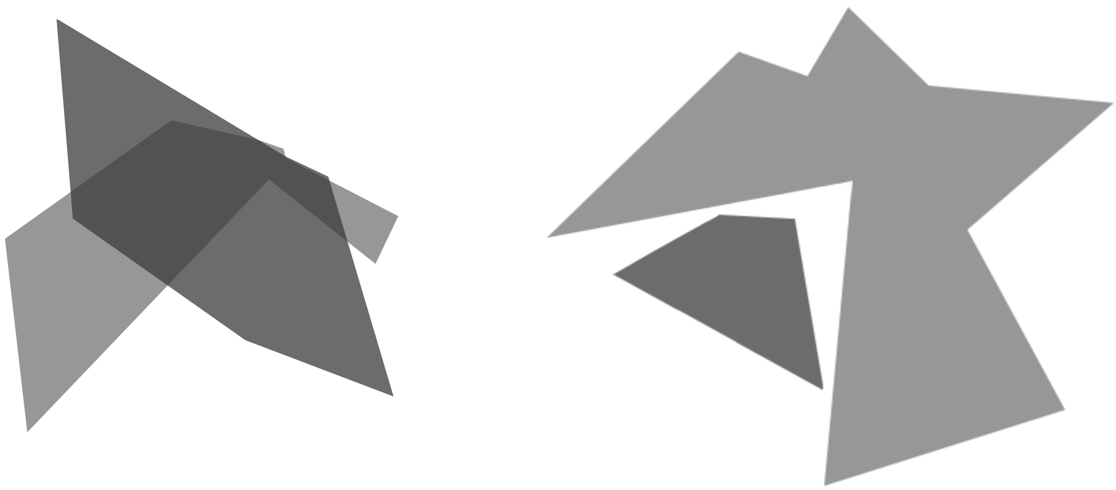

Figure 6: Two example object pairs created with the polygon generating algorithm.

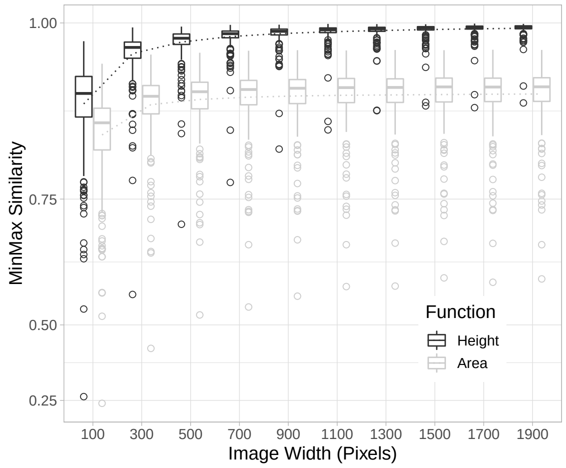

Figure 7: Results of the *MinMax* similarity tests over various resolutions. 300 object pairs were tested at each resolution. As the resolution increases, the similarity improves.

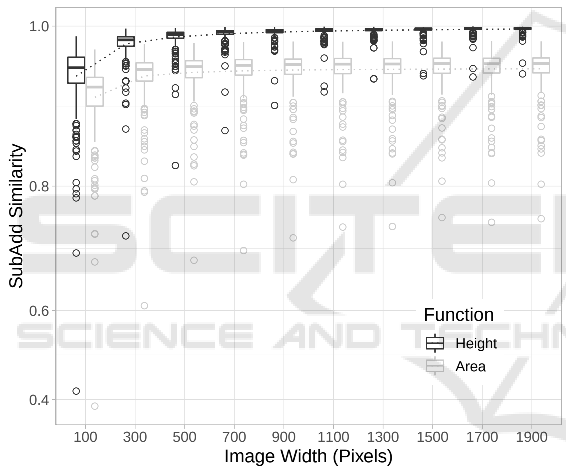

Figure 8: Results of the *SubAdd* similarity tests over various resolutions. 300 object pairs were tested at each resolution. As the resolution increases, the similarity improves.

4.2 Results

Three hundred vector object pairs were created, converted to raster pairs with various resolutions, and tested against both the raster and vector ϕ-descriptor algorithms. Figures 7 and 8 show the *MinMax* and *SubAdd* similarity results. The similarities tend to improve as the resolution of the raster objects increases.

The graphs of the results show some outliers for pairs of descriptors that are particularly dissimilar. These results are from images that have very small regions, as shown in Figures 9 and 10. When rasterizing ϕ-regions that are very thin, portions of the objects may fit between rows or columns of pixels; the resulting raster image may not show any pixels for these

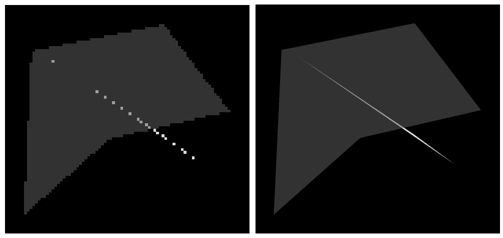

Figure 9: These are two raster images of object pair 73. The image on the left has a width of 100 pixels, and the image on the right 1,100. The image on the left shows gaps in the thin object, which was too thin to be correctly rasterized. It therefore produced a very low similarity score.

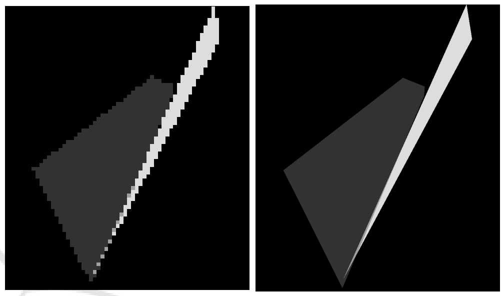

Figure 10: These are two raster images of object pair 43. The image on the left has a width of 100 pixels, and the right 1,100. There are multiple locations in the left image where a pixel from object A is directly adjacent to a pixel from object B. However, this never actually occurs in the vector image, as there is either an overlap or a space between the object. The left image therefore produced a very low similarity score.

regions at all, as shown in Figure 9. When objects have edges that are very close, the raster image may have pixels from the two objects directly adjacent, while the vector image shows the objects to overlap, or have a gap, as shown in Figure 10. It is also important to note that we are not comparing the original descriptors, but rather the normalized descriptors, as explained in Section 4. If we look at Figure 9 it is apparent that the region of interaction will change depending on what pixels are detected for each object.

While these outliers show poor similarity, this does not indicate that either algorithm is producing incorrect results. Both algorithms are giving the correct results for the input data provided.

When analysing the time required to calculate the ϕ-descriptor we must consider the number of vertices in the vector object. Figure 11 shows the processing time for 100 randomly generated object pairs, with random numbers of vertices (the total number of vertices of the pairs ranges from $12-181$). We can see that the number of vertices has a large impact on the processing time; Figure 12 shows the object pair that required the most processing time, and had the most vertices.

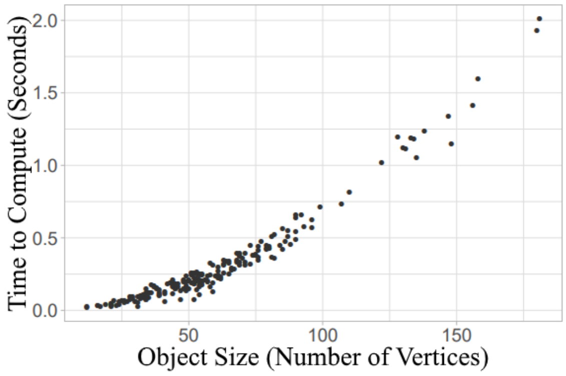

Figure 11: The processing times of 100 vector object pairs. The number of vertices refers to the sum of number of vertices of both objects in each pair. It is clear that a larger number of vertices increases the processing time of the ϕ-descriptor computation for vector objects.

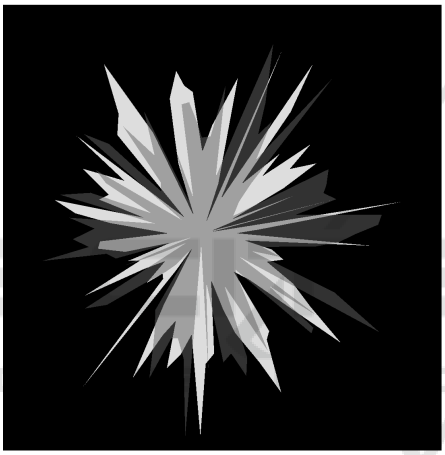

Figure 12: The vector object pair that required the most processing time. There are a total of 181 vertices between the two objects.

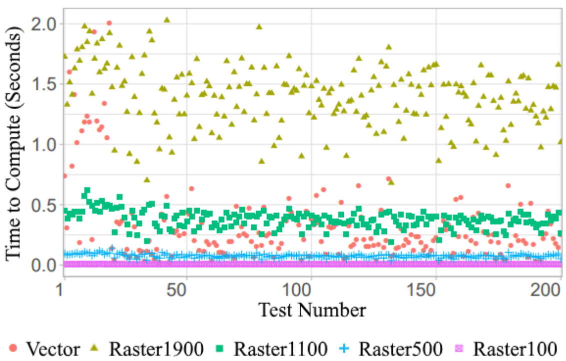

Figure 13: A comparison of the processing times of 100 pairs of objects. We consider vector objects, and raster objects with a pixel width of $100, 500, 1100$, and 1900.

Figure 13 compares the processing time of these same 100 vector object pairs to their raster counterparts. We examine the raster objects at $100, 500, 1100$, and 1900 pixels in width. We see the computation time for the ϕ-descriptor of raster objects is very consistent for a given object size, and as expected the more pixels in the raster object, the more processing time is required. When the raster objects are between 500 and 1100 pixels in width, and the vector object pair has less than 100 vertices, the ϕ-descriptors are often processed in roughly the same time range.

4.3 Current Limitations

There are some limitations with the algorithm in regard to the current implementation that should be noted. In particular, the handling of objects with holes or made of multiple connected components has not been implemented. It should be noted that these limitations are not a fault of the algorithm itself, but rather is a fault of how it was implemented. Work is currently being done to eliminate these limitations and to test robustness of the algorithm against these cases.

5 CONCLUSION

We have introduced the first algorithm to calculate the ϕ-descriptor in the case of 2D vector objects. Our motivation was to show that the ϕ-descriptor is a capable successor to a well-known Relative Position Descriptor (the force histogram), and is equally versatile. The algorithm was tested against 300 randomly generated pairs of 2D vector objects. Each pair had the 2D vector ϕ-descriptor computed using the new algorithm, the corresponding 2D raster ϕ-descriptor was computed as well, and the two descriptors were compared. Our results have shown that in most cases the vector and raster descriptors were very similar as expected, and the higher the resolution of the rasterized objects the higher the similarity. Cases with poor similarity were found to have very thin regions that could not be properly rasterized, i.e., the low similarity scores were due to data loss in the conversion, not a fault of either algorithm. Potential applications of this work are in areas where a powerful RPD is required to encapsulate a great amount of spatial relationship information between 2D vector objects (e.g., human-robot communication and geographical information systems).

Work is currently being done to eliminate the limitations discussed in Section 4.3. Future versions of this algorithm are planned to be capable of handling objects with holes or made of multiple connected components. This algorithm also has the potential to be expanded to the case of 3D vector objects.

REFERENCES

Francis, J., Rahbarnia, F., and Matsakis, P. (2018). Fuzzy nlg system for extensive verbal description of relative positions. *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8.

Freeman, J. (1975). The modelling of spatial relations. *Computer graphics and image processing*, 4(2):156–171.

Keller, J. M. and Wang, X. (1995). Comparison of spatial relation definitions in computer vision. In *Proceedings of 3rd International Symposium on Uncertainty Modeling and Analysis and Annual Conference of the North American Fuzzy Information Processing Society*, pages 679–684. IEEE.

Keller, J. M. and Wang, X. (1996). Learning spatial relationships in computer vision. In *Proceedings of IEEE 5th International Fuzzy Systems*, volume 1, pages 118–124. IEEE.

Krishnapuram, R., Keller, J. M., and Ma, Y. (1993). Quantitative analysis of properties and spatial relations of fuzzy image regions. *IEEE Transactions on fuzzy systems*, 1(3):222–233.

Matsakis, P. (2016). Affine properties of the relative position phi-descriptor. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 1941–1946.

Matsakis, P., Naeem, M., and Rahbarnia, F. (2015). Introducing the phi-descriptor - a most versatile relative position descriptor. In *ICPRAM*, pages 87–98.

Matsakis, P. and Wendling, L. (1999). A new way to represent the relative position between areal objects. *IEEE Transactions on pattern analysis and machine intelligence*, 21(7):634–643.

Miyajima, K. and Ralescu, A. (1994). Spatial organization in 2d segmented images: representation and recognition of primitive spatial relations. *Fuzzy Sets and Systems*, 65(2-3):225–236.

Naeem, M. and Matsakis, P. (2015). Relative position descriptors - a review. *ICPRAM 2015 - 4th International Conference on Pattern Recognition Applications and Methods, Proceedings*, 1:286–295.

Ounsworth, M. (2012). Algorithm to generate random 2d polygon: https://stackoverflow.com/questions/8997099/algorithm-to-generate-random-2d-polygon.

Page, J. D. (2011). Algorithm to find the area of a polygon: https://www.mathopenref.com/coordpolygonarea2.html.

Recoskie, D., Xu, T., and Matsakis, P. (2012). A general algorithm for calculating force histograms using vector data. In *ICPRAM (1)*, pages 86–92.

Reed, J., Naeem, M., and Matsakis, P. (2014). A first algorithm to calculate force histograms in the case of 3D vector objects. pages 104–112.

Santosh, K., Lamiroy, B., and Wendling, L. (2012). Symbol recognition using spatial relations. *Pattern Recognition Letters*, 33:331–341.