Preparatory Modelling Week, September 7–14, 2015, Sofia, Bulgaria

Group 1

Circular Arc Approximation of Pointwise Curves for Use in the NC Programming

Dimitar Fidanov

Faculty of Physics, Plovdiv University
dimitarf@gmail.com

Maria Paskova

Faculty of Mathematics and Informatics, Sofia University
maria.paskova3@gmail.com

Roni Angelov

Faculty of Mathematics and Informatics, Sofia University
randomwalkss@gmail.com

Velislav Bodurov

Faculty of Mathematics and Informatics, Sofia University
vellislav@gmail.com

Instructor: **Dragomir Aleksov**

Faculty of Mathematics and Informatics, Sofia University
dragomira@fmi.uni-sofia.bg

Abstract. We have outlined an idea for the solution to a problem concerning arc approximation of pointwise defined curves. The main steps used in the algorithm are described in detail in Section 1.2 and then some results are obtained first on an artificially created toy set of points defining a curve and then on a real data set coming from industry. Some of the concepts used in the text are dealt with in the appendix.

1.1 Introduction

We consider a numerical control cutting machine that thermally cuts hard materials. Due to some limitations of the machine and its software, cutting can only be done in straight line segments and circular arcs. Another problem that needs to be accounted for is the possible damage that can be inflicted to the material being cut when the velocity of the tool is not at an optimal constant level. Too slow velocity will lead to melting and thus damaging the item irreparably. Too fast velocity on the other hand will lead to constant interruptions of the process and is therefore undesirable as well.
As an input we receive N points in the Cartesian plane $\{(x_i, y_i)\}_{i=1}^{N}$ that outline the contour of the item being cut and considering the constraints mentioned in the last paragraph we should develop an algorithm that approximates the real contour with line segments and circular arcs so that the Hausdorff distance between the points and the new outline is minimized.
Since the problem where the curve contains straight line segments is well studied we shall focus solely on the problem of approximation only by circular arcs. From now on we can assume that the input does not contain three or more consecutive points that lie on the same straight line (that is to mean with respect to the limited precision computers can work with).
The output of the algorithm should consist of ordered quadruplets

$$((u_1, v_1), (u_2, v_2), (u_c, v_c), \varepsilon)\,,$$

where

- (u_1, v_1) and (u_2, v_2) are respectively the starting and ending point of an arc;
- (u_c, v_c) is the center of the circle from which the arc is taken;
- ε is a flag variable taking two values (for example -1 and 1) which distinguishes between the two possible arcs defined by the first three elements of the quadruplet;

Below we describe our approach to the problem. A different idea can be found in [1].

1.2 Description of the Algorithm

We shall assume that the input point cloud comes equipped with a topology (i.e. ordering of the points). The proposed algorithm works in three steps which we will describe in detail below.

1.2.1 First Step of the Algorithm

Our considerations start with the simple observation that any three points in the Cartesian plane define a unique circle passing through them. Using some knowledge of analytic geometry one can easily find the center and radius of such circle. If the three points have coordinates $(x_1, y_1), (x_2, y_2)$ and (x_3, y_3) we get

$$x_c = \frac{\begin{vmatrix} x_1^2 + y_1^2 & y_1 & 1 \\ x_2^2 + y_2^2 & y_2 & 1 \\ x_3^2 + y_3^2 & y_3 & 1 \end{vmatrix}}{2 \cdot \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}},$$

$$y_c = \frac{\begin{vmatrix} x_1 & x_1^2 + y_1^2 & 1 \\ x_2 & x_2^2 + y_2^2 & 1 \\ x_3 & x_3^2 + y_3^2 & 1 \end{vmatrix}}{2 \cdot \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}},$$

$$r = \sqrt{(x_1 - x_c)^2 + (y_1 - y_c)^2}$$

for the coordinates of the center and the radius of the circle.
Then we proceed as follows: Take the first three points from the input and draw the unique circle passing through them. Repeat this procedure for the second, third and fourth points respectively. The result will look like that on Figure 1.1. The next thing we do is to look at the angle between the tangent vectors of the two circles at one of their common points. Due to the way the algorithm is implemented we will always choose the second common point as is depicted in the figure.

If this angle is "sufficiently small" (to be made precise in the third step of the algorithm) we can conclude that the part of the boundary defined by all four points can be approximated well with a single circular arc.

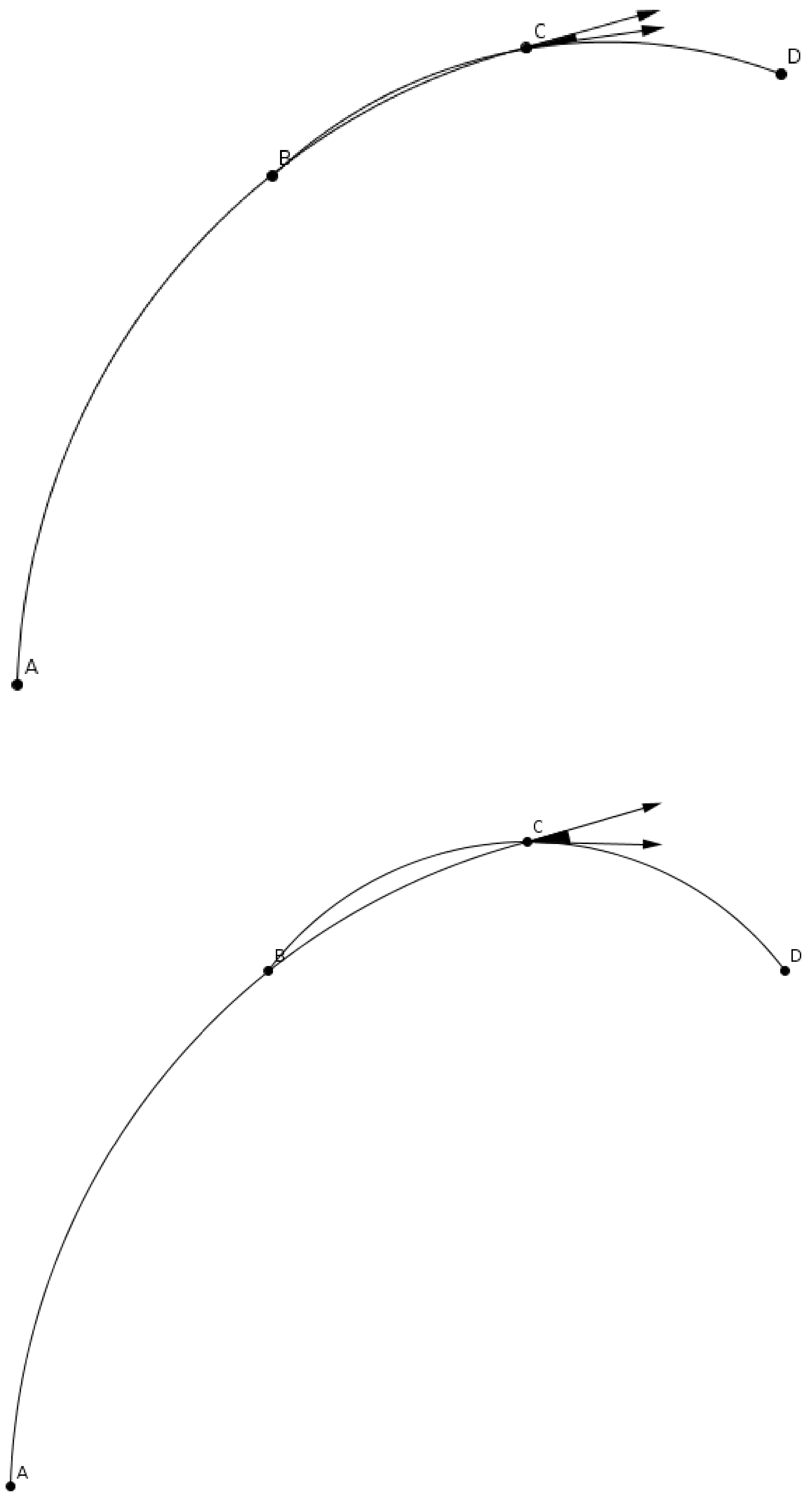

Figure 1.1: Results after the first step of the algorithm.

1.2.2 Second Step of the Algorithm

If the situation described in the previous paragraph is indeed the case we get to the second step of our algorithm which is aimed at finding a suitable circular arc that best approximates the boundary defined by the four points. This can be defined more generally as the following optimization problem:

$$\inf_{a,b,r}\{\sup_{i=1,2,3,4}\{|\sqrt{(x_i-a)^2+(y_i-b)^2}-r|\}\},$$

where (a,b) is the center of the arc and r is its radius.
The problem though seems too general to tackle directly so we decided to simplify it by reducing the degrees of freedom. We did this by forcing the circular arc to pass through the first and the last point of the point cloud currently under consideration (again - throughout the text we assume that the points are equipped with an ordering of some sort). Then our problem reduces to:

$$\inf_{r}\{\sup_{i=1,2,3,4}\{|\sqrt{(x_i-\alpha)^2+(y_i-\beta)^2}-r|\}\},$$

where $\alpha=\alpha(x_1,y_1,x_4,y_4,r)$ and $\beta=\beta(x_1,y_1,x_4,y_4,r)$ are functions of r only (note that x_1,y_1,x_4 and y_4 are held constant).
Yet again the problem is simplified in the following manner: We do not solve it explicitly but rather approach the solution in an approximate way. As is shown in Figure 1.2 (in the more general case) we can construct two arcs that enclose all points under consideration in the area formed by them. If their centers are respectively P and Q, the center of the optimal arc is sure to lie on the line segment PQ (which by elementary geometric considerations is a part of the perpendicular bisector of the segment AB).
Now we move from P to Q (the direction doesn't matter actually) along $\vec{PQ}$ with a predefined step $\delta=\rho|\vec{AB}|$, where ρ is some constant of proportionality (the idea is to somehow relate the step size to the length of AB so we don't exhaust $\vec{PQ}$ too quickly). To be more specific we create arcs iteratively with centers $P+k\times\delta\vec{PQ}$, where k ranges from 0 to $\lceil\frac{1}{\delta}\rceil$. We settle for the arc which minimizes the Hausdorff distance from the point cloud to itself.
If we have found a suitable arc we return to Step 1 and consider it along with the arc defined by the last two points used as well as the next one in the list. Now we can consider the angle between the tangents at the (possibly) one common point (that's why we insisted on picking the last common point of the two arcs - in this implementation of the algorithm we are not guaranteed that there will be others in the more general case of n points) between the two arcs and if "sufficiently small" we can again go to Step 2.
If we come to a point where the magnitude of the angle no longer satisfies us we end up with one arc constructed and start the process all over again beginning from the endpoint of the arc just created.

1.2.3 Third Step of the Algorithm

This step of the algorithm will shed a light on the choice of a threshold for the angle used in Step 1.

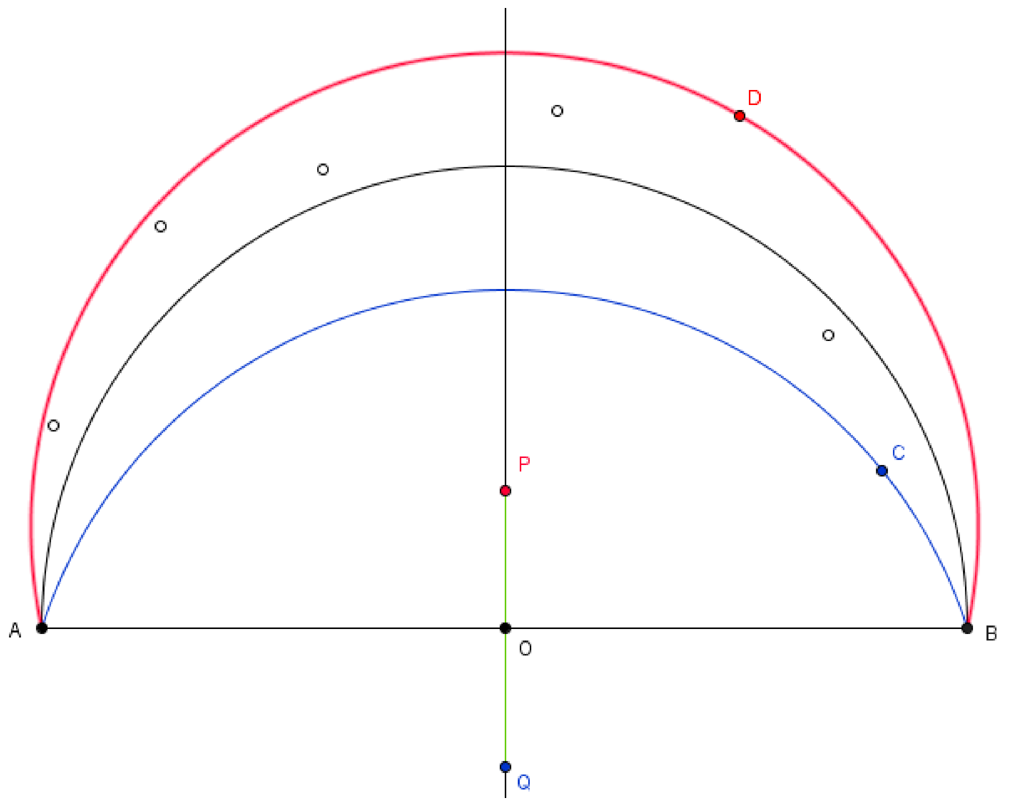

Figure 1.2: Second Step of the Algorithm

We fix a starting point for the angle $\varphi = 0.1\,rad$ for example and perform the algorithm just described. As a result we get a set of arcs forming some sort of a curved line (call it l) and compute the total error of the approximation as the maximum of the Hausdorff distances of the points to their respective arcs.
Now it's time to use some of the physical constraints imposed by the machine and in particular it's accuracy. Due to limitations of physical nature the device cuts with an accuracy of between $0.1\,mm$ and $0.01\,mm$ (that means it won't distinguish between points whose coordinates differ from the third digit after the decimal point on). For the curved line l we can easily check if the error exceeds $0.01mm$ and if it does we repeat the procedure with $\varphi_{new} := \frac{\varphi}{10}$. We do this until we force the error below $0.01\,mm$ and stop at the first value of φ to do so (this will end eventually since we can force the error to be zero by constructing an arc through every three consecutive points).

1.2.4 Additional Considerations

One of the potential problems that can arise when using the outlined procedure is the situation where after constructing our "last" arc we are left with only one point that is not yet used. So now we have just two points to construct an arc through (the endpoint of the previous one and our "free" point). Since this can't be done uniquely (and the first step of our algorithm will fail in this case) we

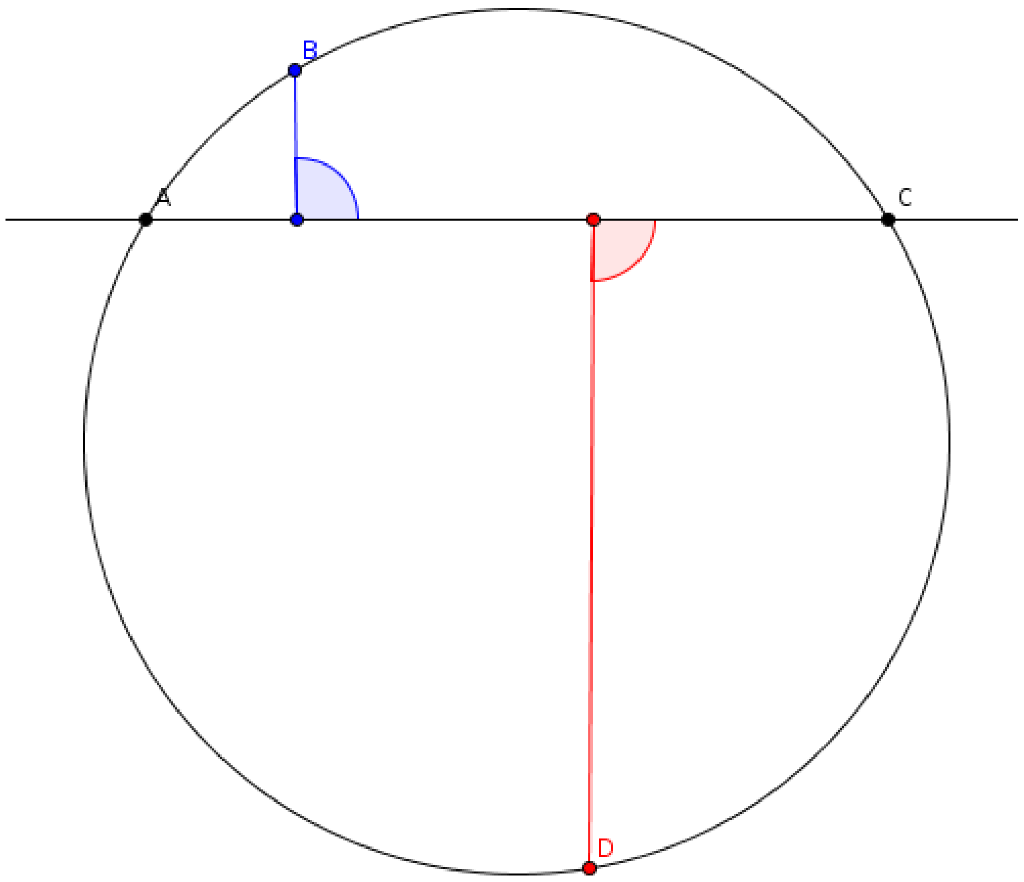

Figure 1.3: Cases for the Flag Variable

take the next-to-last point from the arc just drawn and construct the unique one through the last three points from the point cloud. Now we only consider the segment connecting the last two points and use it to finish up l.

In the introduction we mentioned that the output of the algorithm should include the flag variable ε. Now we explain how to go about and choose the proper value of ε. Suppose we find ourselves in a situation like that on Figure 1.3. We have an arc passing through the points A and C as well as a third point B or D. Intuitively we want to assign the value $\varepsilon = -1$ if our arc passes through (or near) B since the point B seems to lie on the "positive side" of the segment AC and $\varepsilon = 1$ if we have the arc passing through D. We can try to formalize that using the notion of oriented distance. Suppose the coordinates of A and C are respectively (x_1, y_1) and (x_2, y_2). Using that information we can write the equation of a line passing through them. It's exact form is

$$\alpha x + \beta y + \gamma = 0,$$

where $\alpha = y_1 - y_2, \beta = x_2 - x_1, \gamma = x_1 y_2 - x_2 y_1$.

The normal equation of this line has the form

$$\frac{\alpha x + \beta y + \gamma}{\sqrt{\alpha^2 + \beta^2}} = 0.$$

Now if we have the points $B(x_3, y_3)$ and $D(x_4, y_4)$ as shown in the figure above we will get the following results:

$$d(B, AC) = \frac{\alpha x_3 + \beta y_3 + \gamma}{\sqrt{\alpha^2 + \beta^2}} > 0$$

and

$$d(D, AC) = \frac{\alpha x_4 + \beta y_4 + \gamma}{\sqrt{\alpha^2 + \beta^2}} < 0.$$

Now in order to find the "correct" ε along with the two endpoints we choose one more point that was used to create the arc (call it P). Then

$$\varepsilon := -sign(d(P, AC)).$$

One further idea that we considered was to refine the methods used in the third step of the algorithm. That is if we find a value of φ that forces the error below 0.01 we can try to find another value that does the job by considering some values in the interval $(\varphi, 10 \times \varphi)$. The basic idea behind the improvement is that we may have skipped a bit larger value of φ as a result of our predetermined step size and as a direct consequence get more arcs than we should.

1.3 Results

1.3.1 Artificially Created Example

As a first test for our algorithm we created a toy set of 10 points for which we "knew" what the output should be. On Figure 1.4 we can see the set itself with the result depicted in Figure 1.5. As is evident from the figures, four arcs were used for the approximation. In this case we encounter the problem mentioned in the first part of subsection 1.2.4 and have dealt with it using the described procedure.

1.3.2 Real Data

On Figure 1.6 we can see the pointwise defined boundary of a part of an industrial item to be cut. The set consists of 200 points that are part of a parabola. For simplicity we have taken just the first 55 points from the set and applied the algorithm on them. The results are shown on Figure 1.7. The contour defined by the 55 points was approximated with just two arcs (one encompassing 33 of the points and the other - 23). In a table below are shown the exact errors of the approximation and as we can see they are below the threshold of $0.01\,mm$ we set in the beginning.

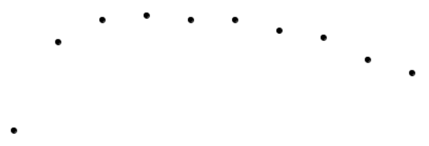

Figure 1.4: Toy Data Set

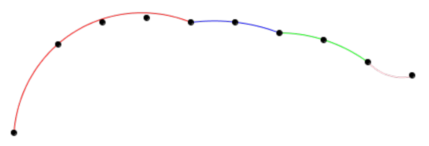

Figure 1.5: Results on the Toy Data Set

Arc	Error
Arc $\overset{\frown}{AC}$	0.0054
Arc $\overset{\frown}{CB}$	0.0041

The algorithm gives 12 arcs in total for the approximation of the whole contour defined by the full data set (not depicted here for reasons of readability).

1.4 Conclusion

As is evident by the experiments and described in the last section the algorithm seems to give a reasonable solution to the proposed problem. The results are comparable with those presented in [1].

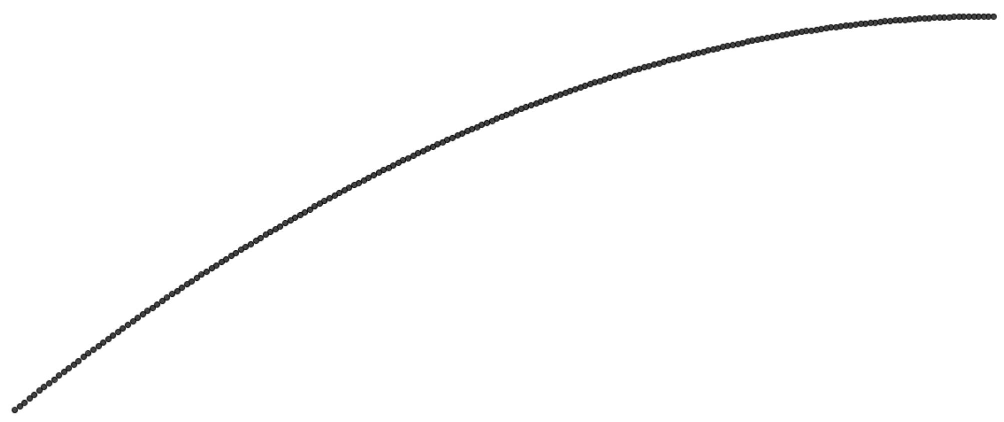

Figure 1.6: Real Data Set Consisting of 200 Points

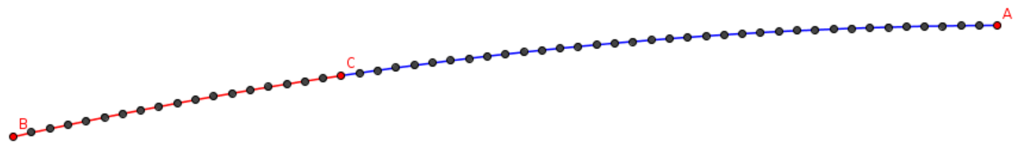

Figure 1.7: Results on a Part of the Real Data Set

Some directions in which the problem can be further developed are:

- Additional optimization of the algorithm
- Solving explicitly the optimization problems posed in subsection 1.2.2
- Testing the algorithm with a set of points representing a closed contour and thus fully automating the process

1.5 Appendix

1.5.1 Hausdorff Distance

The choice of error measure in our approach to the problem fell on the Hausdorff distance [1] and so a brief description of the idea behind it is due.
def. Let X and Y be two non-empty subsets of a metric space (M, d). We define their Hausdorff distance $d_H(X, Y)$ by

$$d_H(X,Y) = max\{sup_{x\in X} inf_{y\in Y} d(x,y), sup_{y\in Y} inf_{x\in X} d(x,y)\}.$$

It is used to measure how far two sets in a metric space are from each other. Informally speaking two sets are close in the Hausdorff distance if each point of

[1]More information on the concept can be found on Wikipedia

either set is close to some point of the other.
While the Hausdorff distance between two sets in a metric space can be infinite [2] when both X and Y are bounded it is guaranteed to be finite. In our case X is a finite set of points and Y is a circular arc (both are bounded, hence the Hausdorff distance will always be finite).

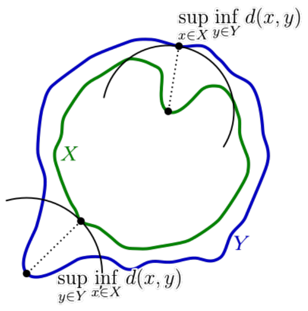
$\sup_{x\in X}\inf_{y\in Y} d(x,y)$
X
Y
$\sup_{y\in Y}\inf_{x\in X} d(x,y)$

Figure 1.8: Example of the Hausdorff Distance Between Two Sets

[2]For example when one set is unbounded and the other is bounded

Bibliography

[1] D. Aleksov et al., Circular Arc Spline Approximation of Pointwise Curves for use in NC Programming, 104th European Study Group with Industry: Problems & Final Reports, Demetra 2014, 94–101.