Chapter 7

Experiments

In this chapter, we present several experiments obtained with the algorithms described in the previous chapter. We first compute the affine erosion of some polygonal curves for different values of the area parameter, and check the affine invariance of the algorithm. We also show the effects of the affine discretization of the computed curves. Then, we compute the affine scale space of these curves by iterating the affine erosion (plus dilation) on them. We compare the results obtained with the exact algorithm to those obtained with the simplified algorithm based on the pseudo affine erosion.

7.1 Affine erosions

On the following experiments (Figure 7.1 to 7.6), the affine erosion $E_\sigma(\mathcal{C})$ of an initial curve $\mathcal{C}$ is represented for different values of the area parameter σ, actually taken in arithmetic progression. We begin with simple polygons and end with more complicated polygonal curves. It is important to notice that this representation is NOT the affine scale space of the initial curve $\mathcal{C}$, since the affine erosion operator is not iterated but simply computed for the same initial curve and increasing values of the area parameter. We shall compute later the corresponding affine scale spaces.

These figures can also be viewed as the level sets of an "affine distance" function $\boldsymbol{x} \mapsto d(\boldsymbol{x},\mathcal{C})$. For any point $\boldsymbol{x}$ lying inside a closed curve $\mathcal{C}$, $d(\boldsymbol{x},\mathcal{C})$ can be defined as the smallest area of a positive chord set of $\mathcal{C}$ enclosing $\boldsymbol{x}$, i.e.

$$d(\boldsymbol{x},\mathcal{C}) = \inf_{K \in \mathcal{K}^+(\mathcal{C}),\ \boldsymbol{x} \in K} \text{area}\,(K).$$

In particular, we have $d(\boldsymbol{x},\mathcal{C}) = 0$ if and only if $\boldsymbol{x} \in \mathcal{C}$, and

$$E_\sigma(\mathcal{C}) = \{\boldsymbol{x} \in \mathcal{I}(\mathcal{C}),\ d(\boldsymbol{x},\mathcal{C}) = \sigma\}.$$

To give an example, computing the 67 iterations of Figure 7.2 takes 0.3 second (CPU time) on a HP 735/125 station.

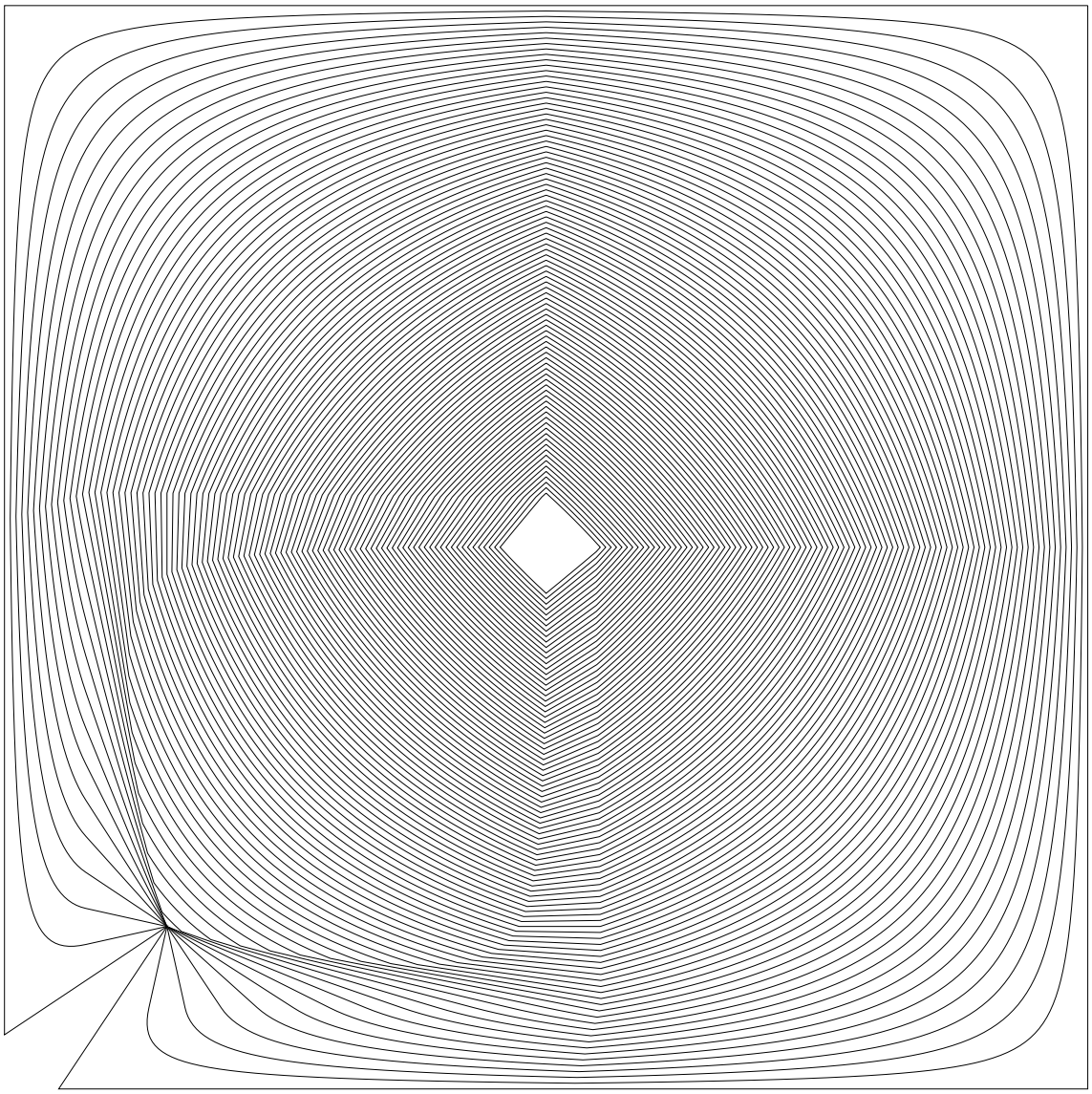

Figure 7.1: Affine erosions (modified square)

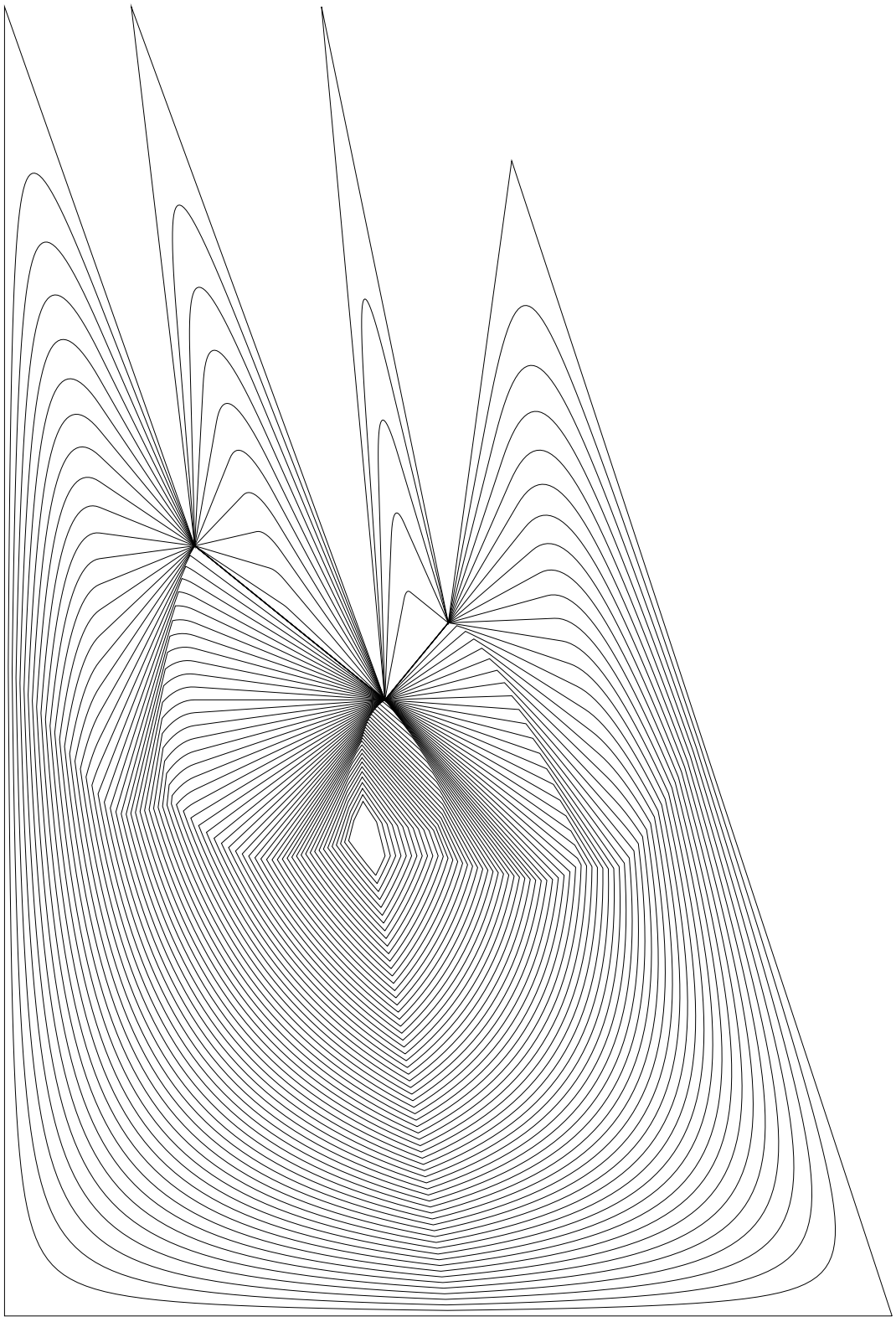

Figure 7.2: Affine erosions (teeth polygon)

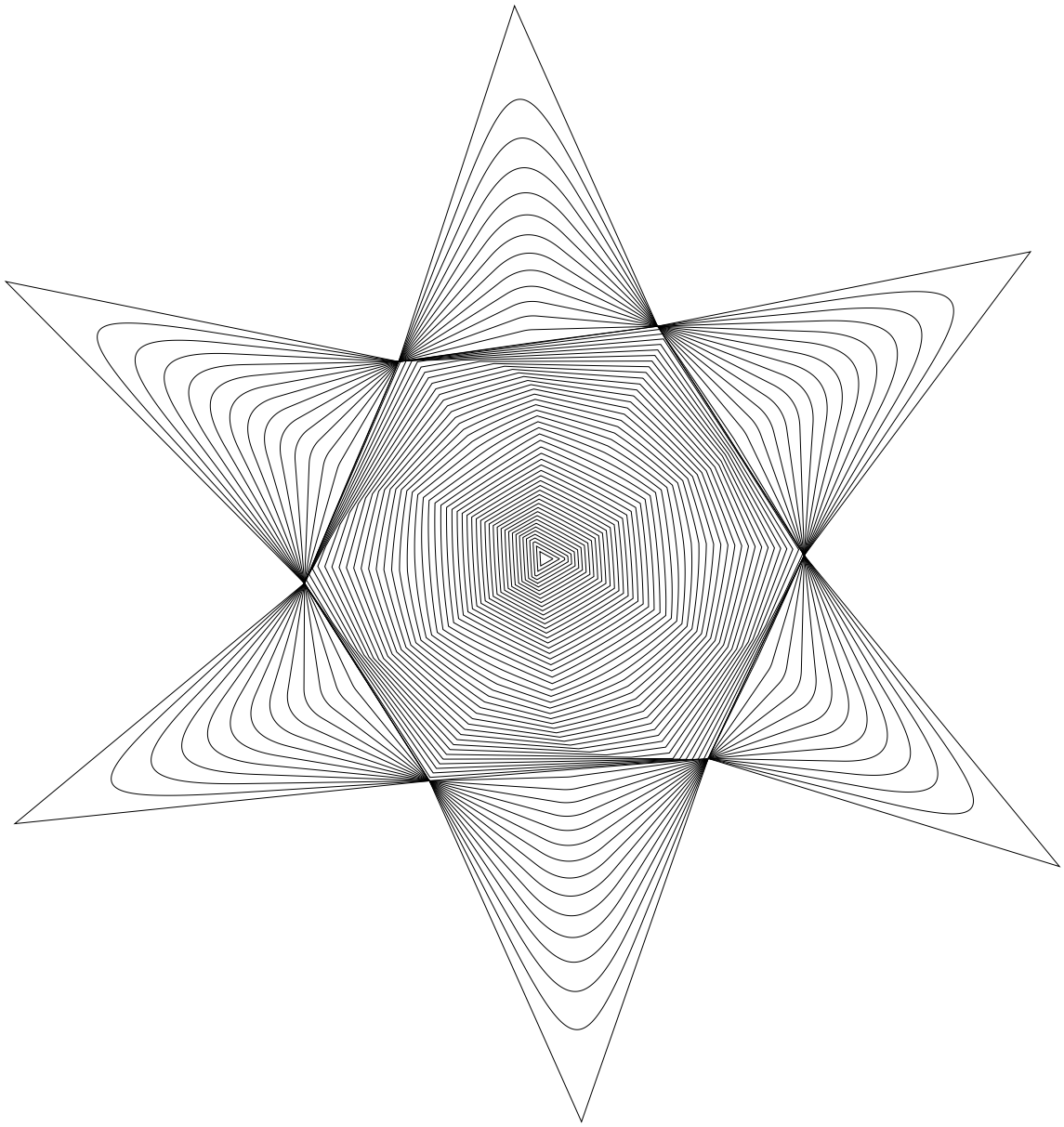

Figure 7.3: Affine erosions (non-symmetric star)

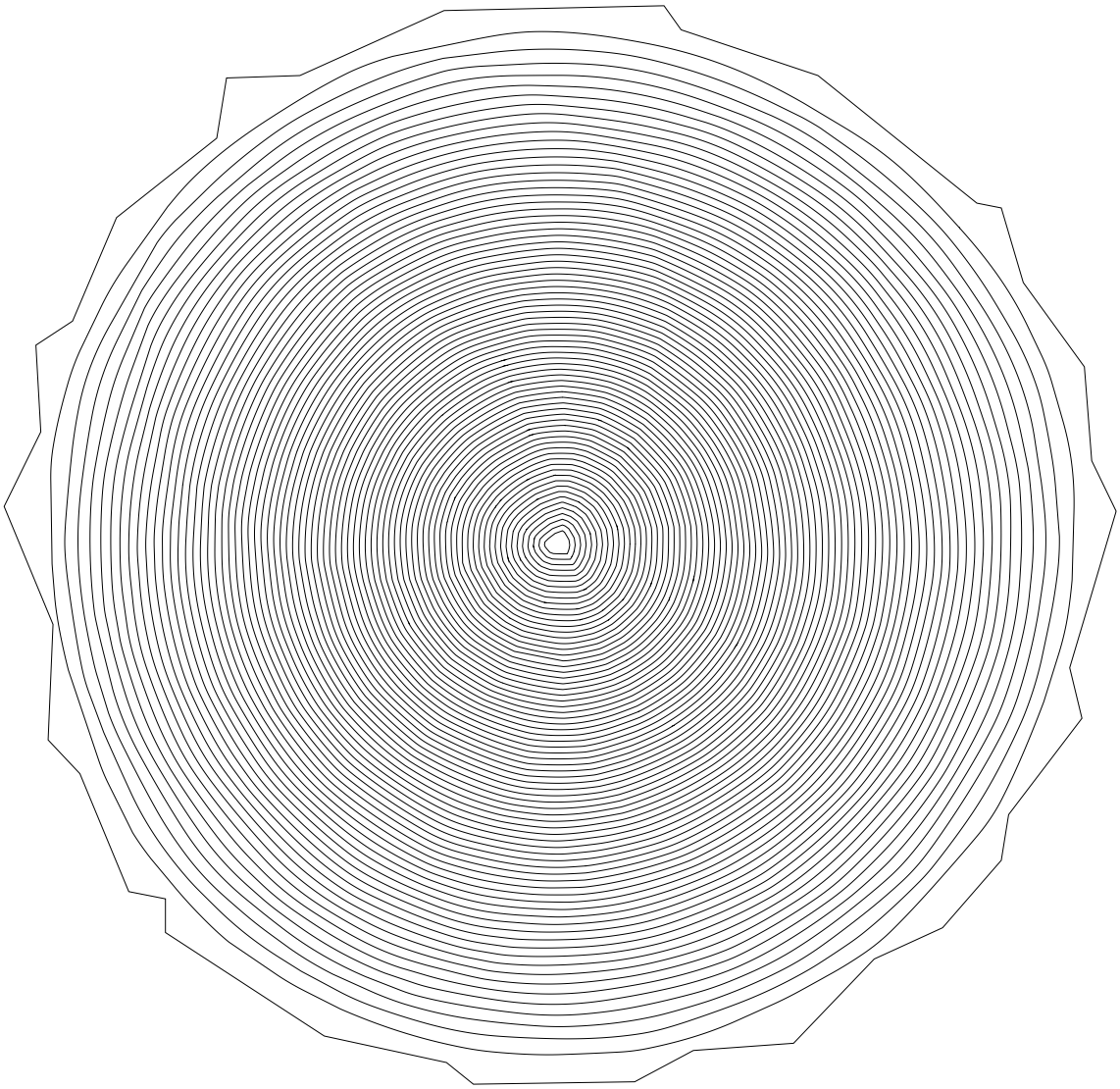

Figure 7.4: Affine erosions (rough circle)

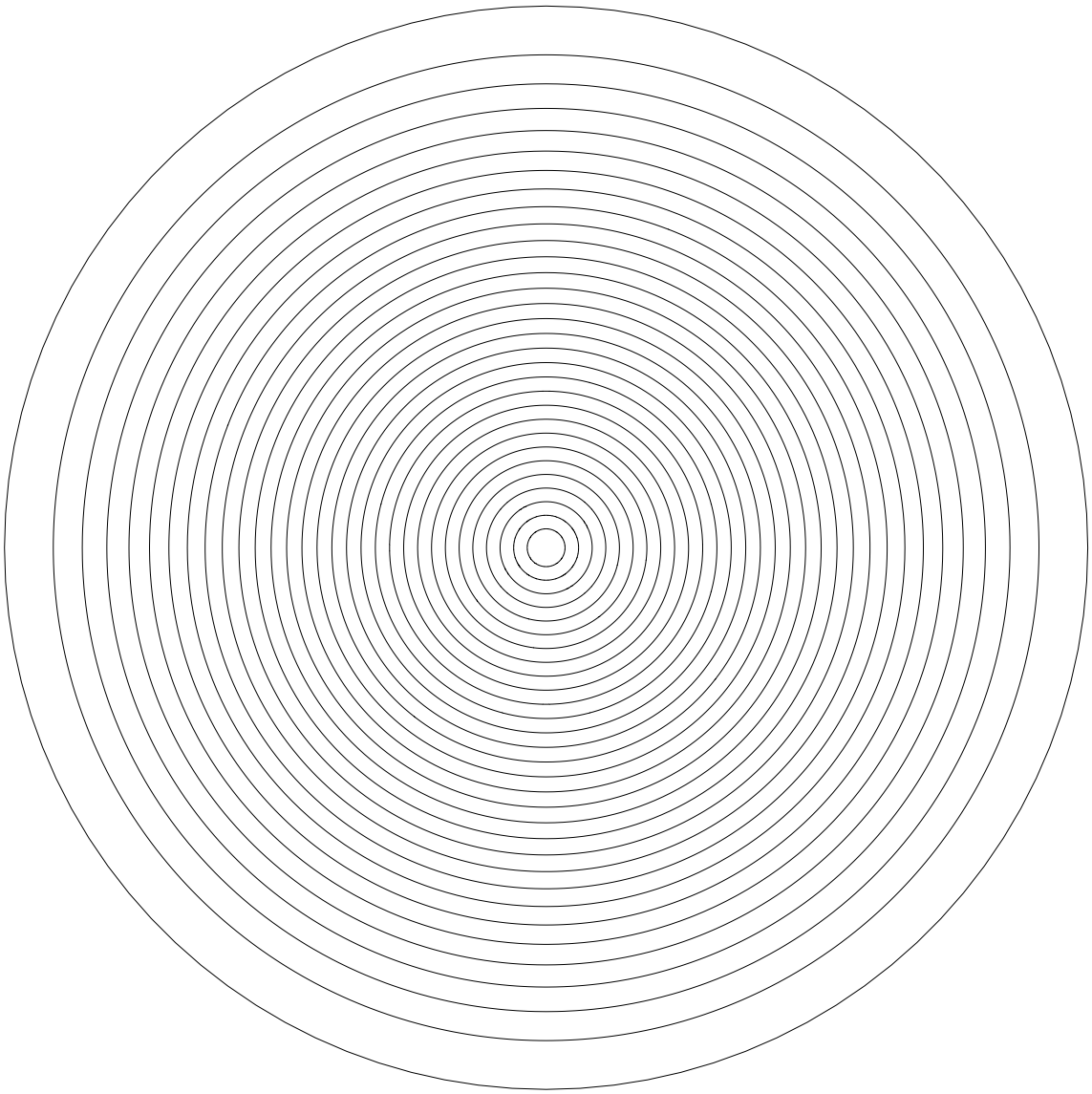

Figure 7.5: Affine erosions (exact circle)

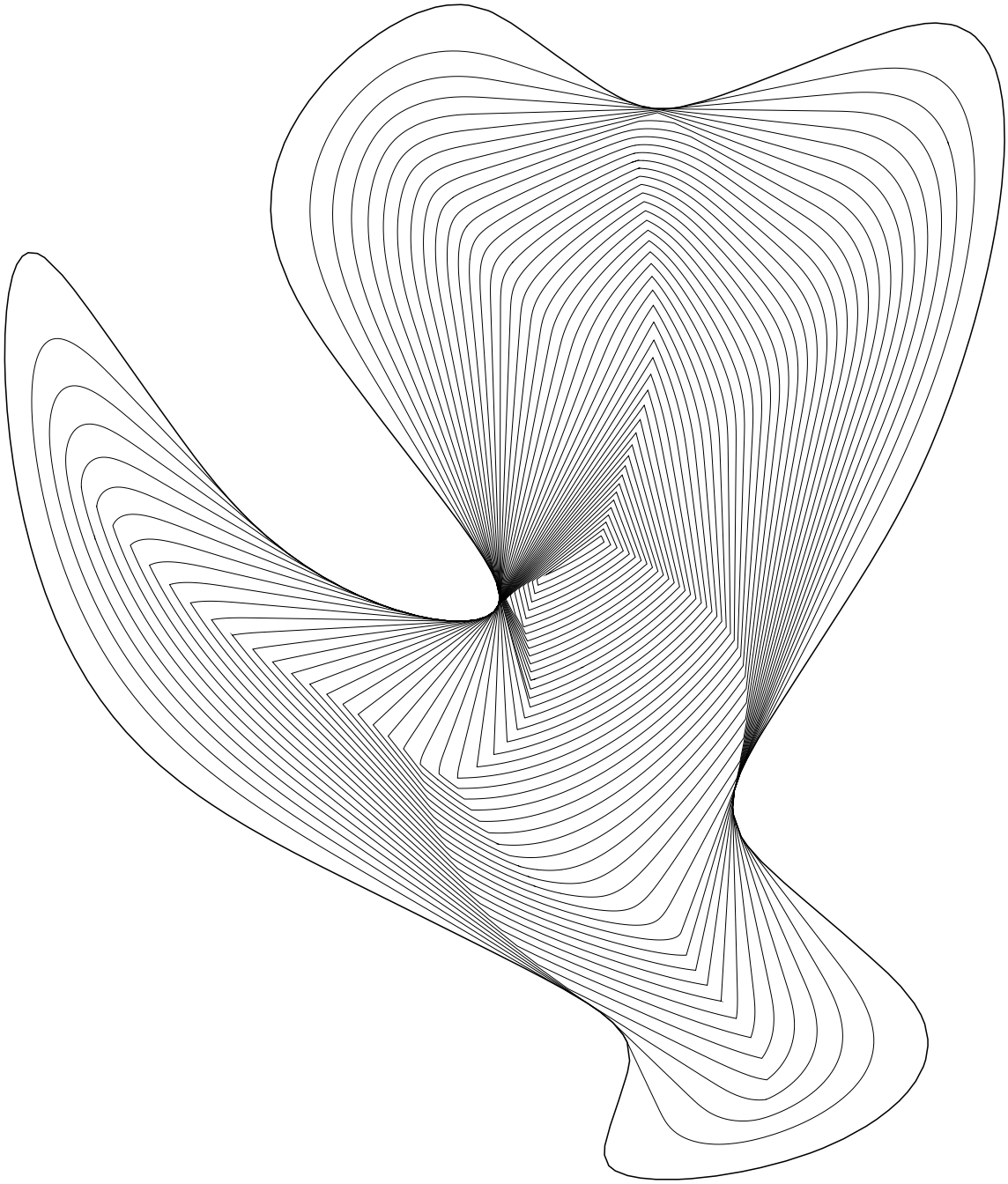

Figure 7.6: Affine erosions (regular curve)

7.1.1 Discretization

The next experiments (Figures 7.7 to 7.9) highlight the affine invariance property of the discretization process we described in the previous chapter. The affine erosion of some of the previous curves is computed for increasing values of the area parameter, and with a rather large sub-sampling area step in order the discretization to be easily seen. Notice how the sampling adapts to the resulting curve.

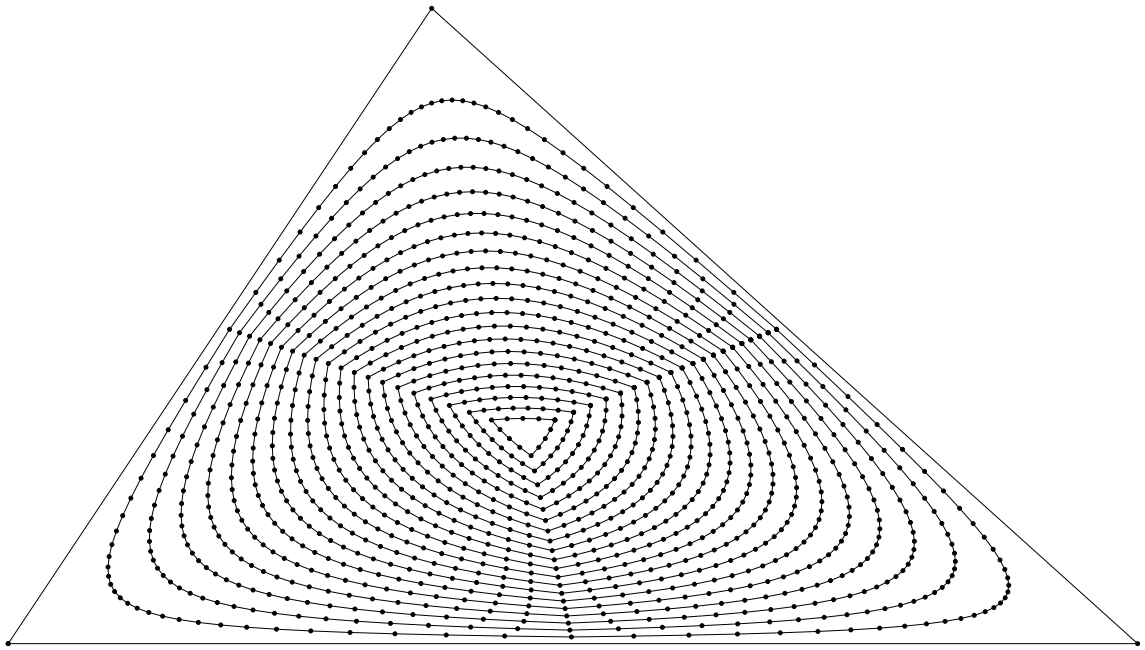

Figure 7.7: Discretized affine erosions (triangle)

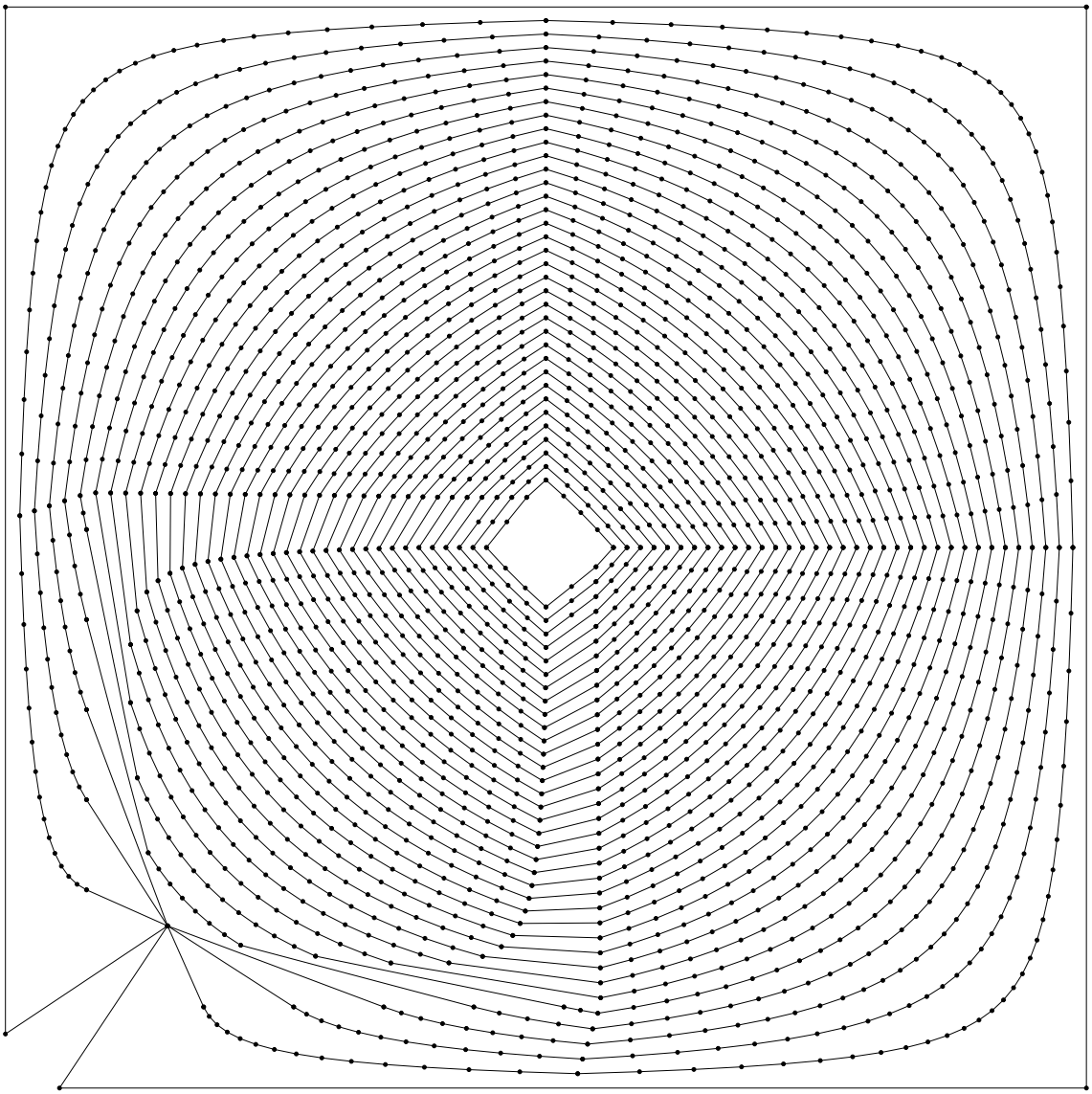

Figure 7.8: Discretized affine erosions (modified square)

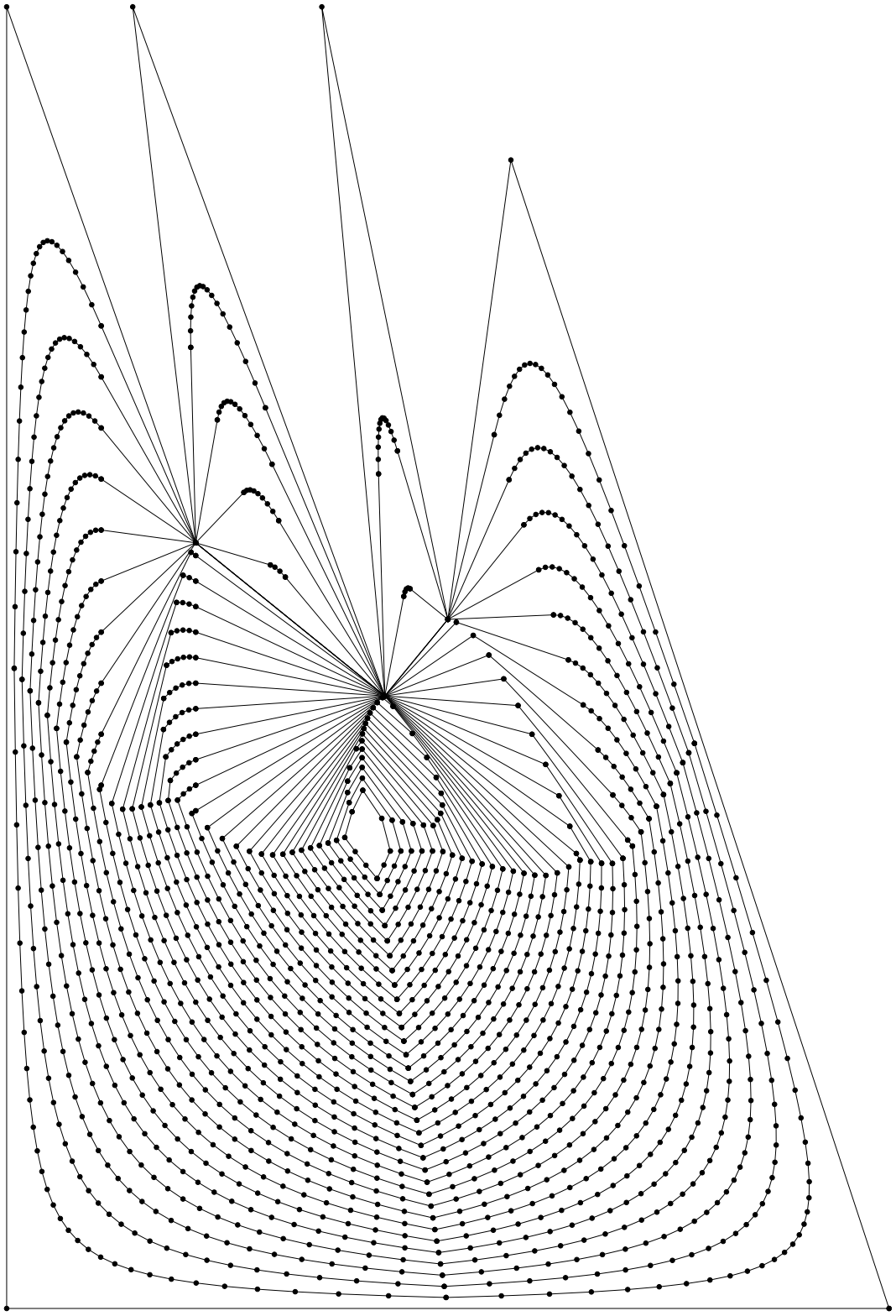

Figure 7.9: Discretized affine erosions (teeth polygon)

7.1.2 Affine Invariance

We now check the affine invariance of the exact algorithm described in the previous chapter. We apply an affine transformation to the initial curve of Figure 7.2 and then compute the affine erosion for the same values of the area parameter (Figure 7.10). The inverse affine transformation being applied (Figure 7.11), we check that we obtain the same result as Figure 7.2.

We use the same method to check that the discretization is affine invariant too (Figure 7.13 to be compared to Figure 7.9).

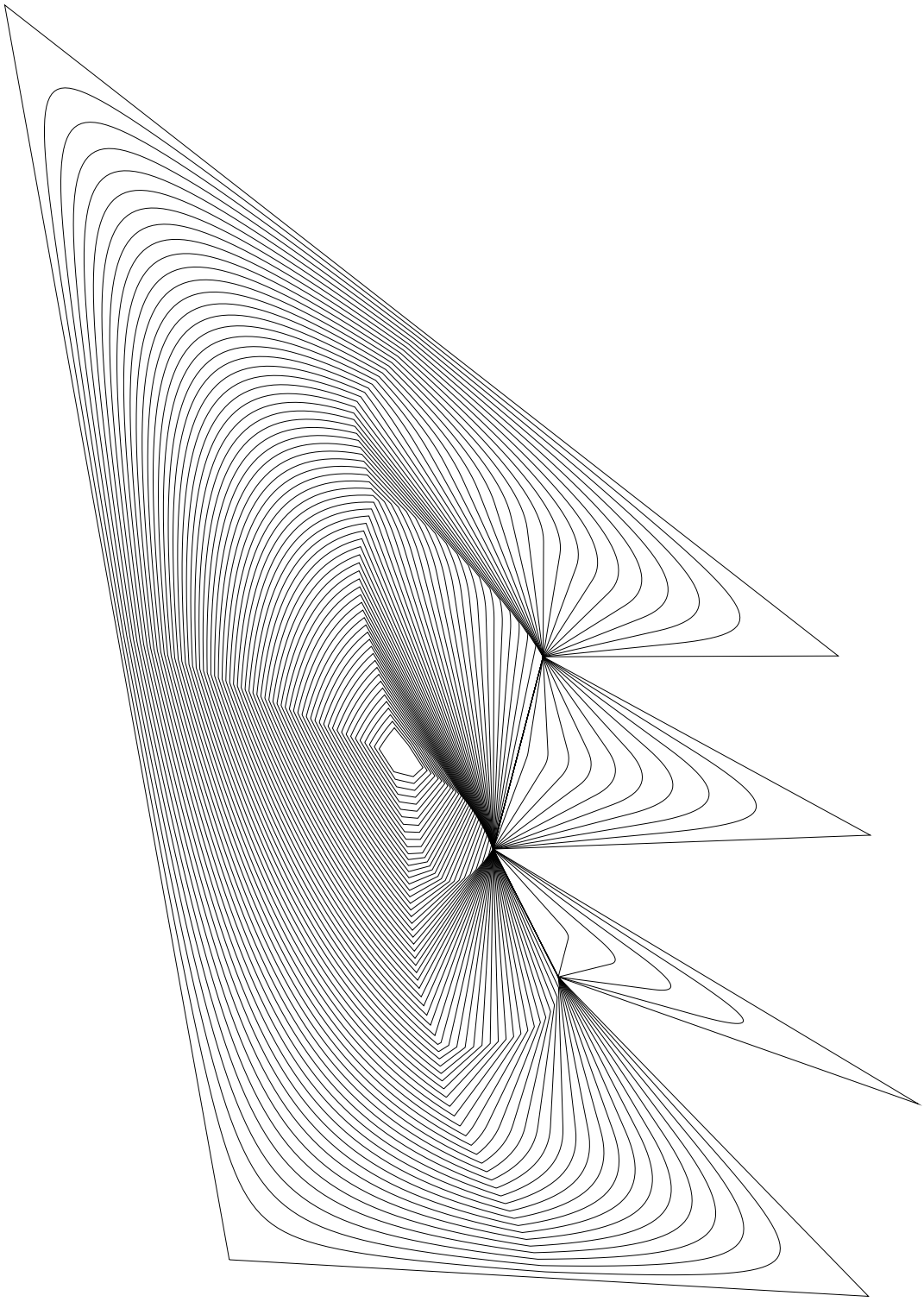

Figure 7.10: Affine erosions (distorted teeth polygon)

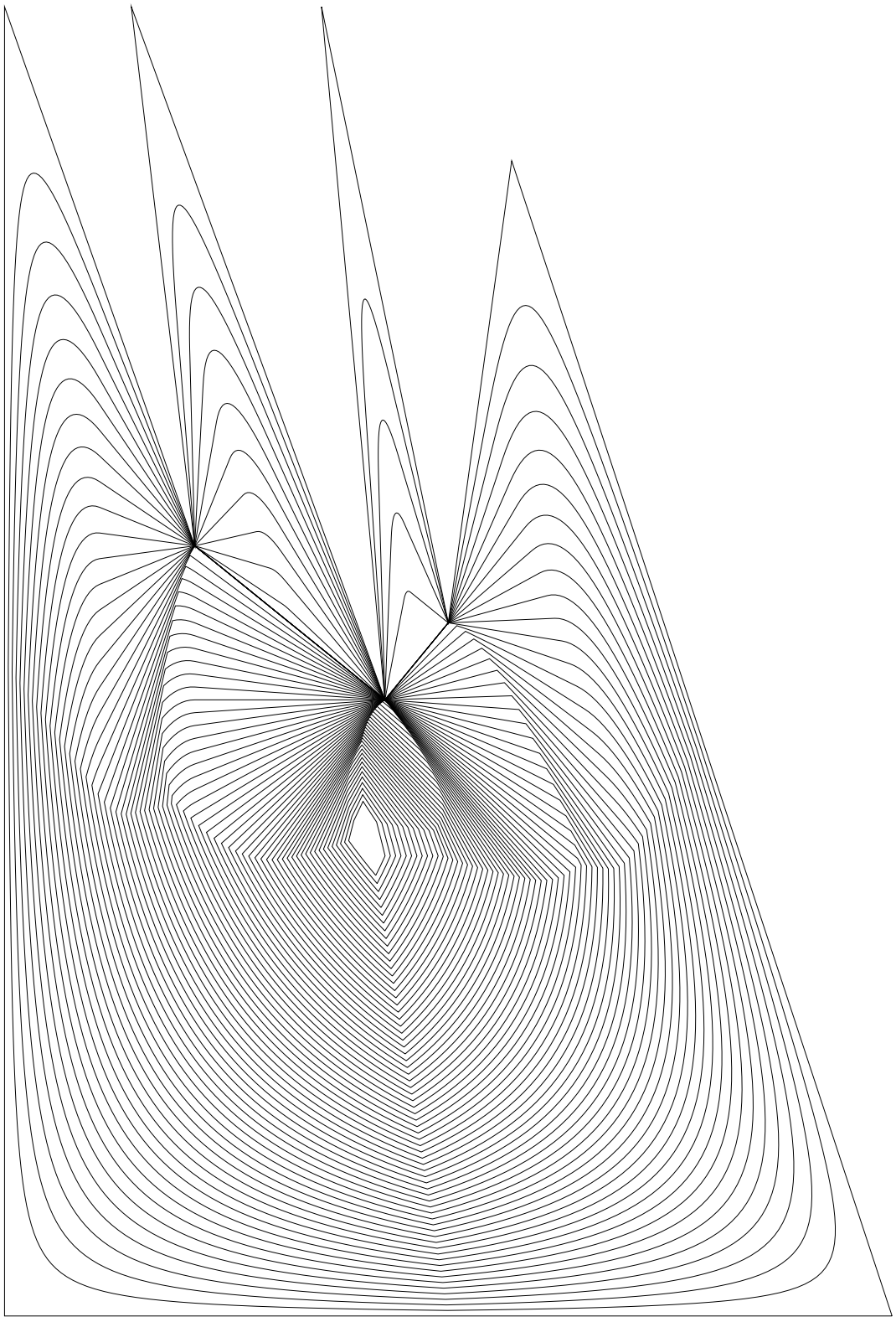

Figure 7.11: Inverse affine transformation of the previous figure
According to the theory, we obtain the same result as on Figure 7.2.

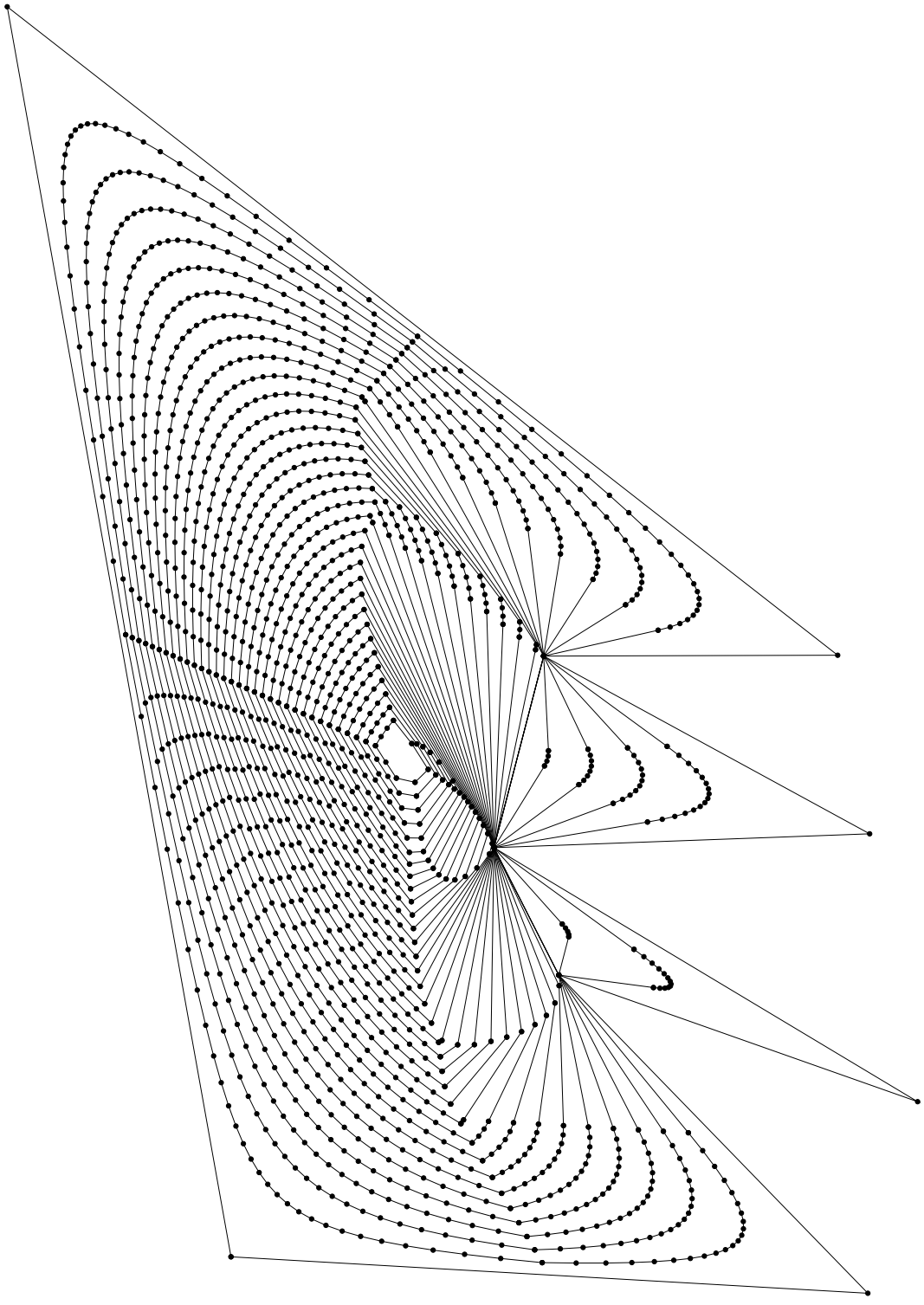

Figure 7.12: Discretized affine erosions (distorted teeth polygon)

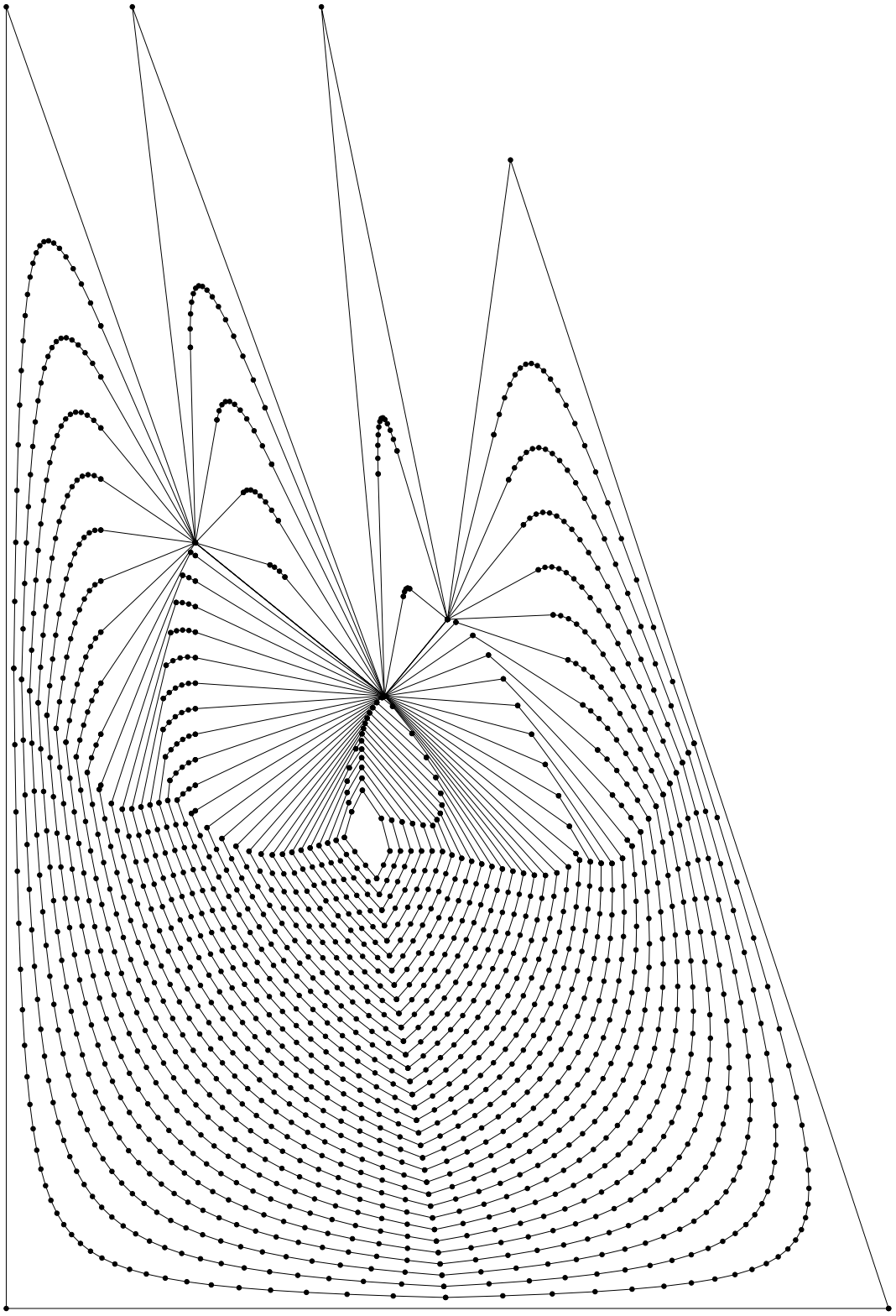

Figure 7.13: Inverse affine transformation of the previous figure

7.2 Affine scale spaces

7.2.1 Exact algorithm

The following experiments simulate the affine scale space on non-convex polygonal curves, as obtained by iterating the exact algorithm . Each curve corresponds to one iteration of the affine erosion plus dilation, computed using the exact algorithm described in the previous section. As predicted by the theory, the curves collapse in a "elliptically shaped" point (see [67]).

Computing the 29 iterations of Figure 7.18 takes 6 minutes (CPU time) on a HP 735/125 station. The number of sampled points reaches 700 for some iterations and the number of computed curves (hyperbolae and segments) attains 1600.

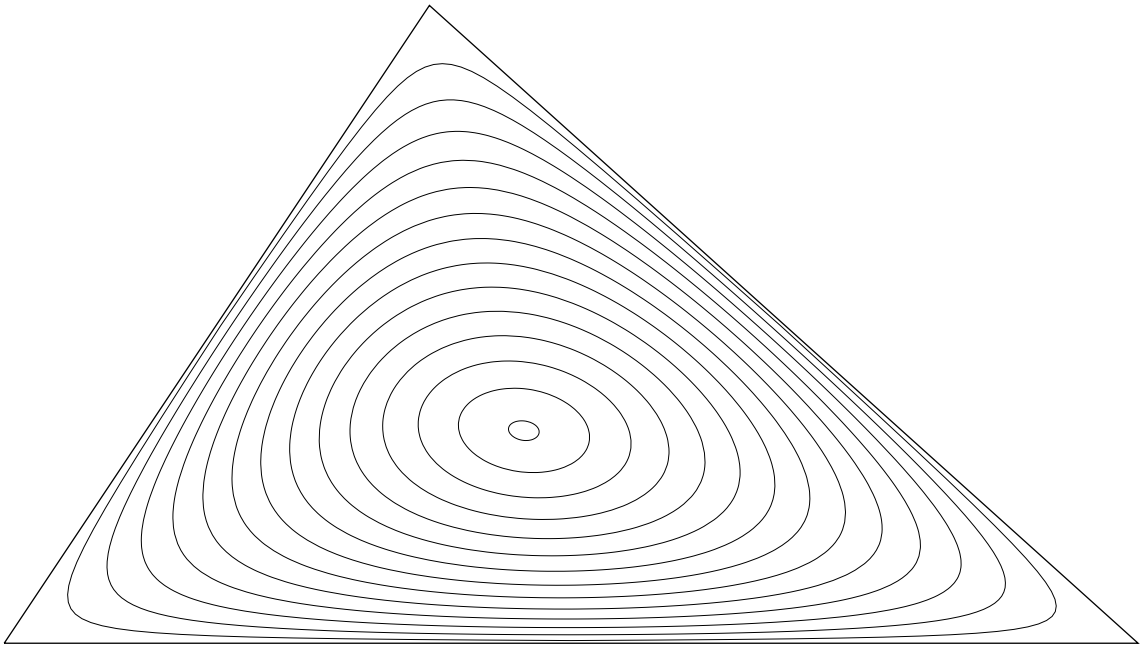

Figure 7.14: Affine scale space (triangle)

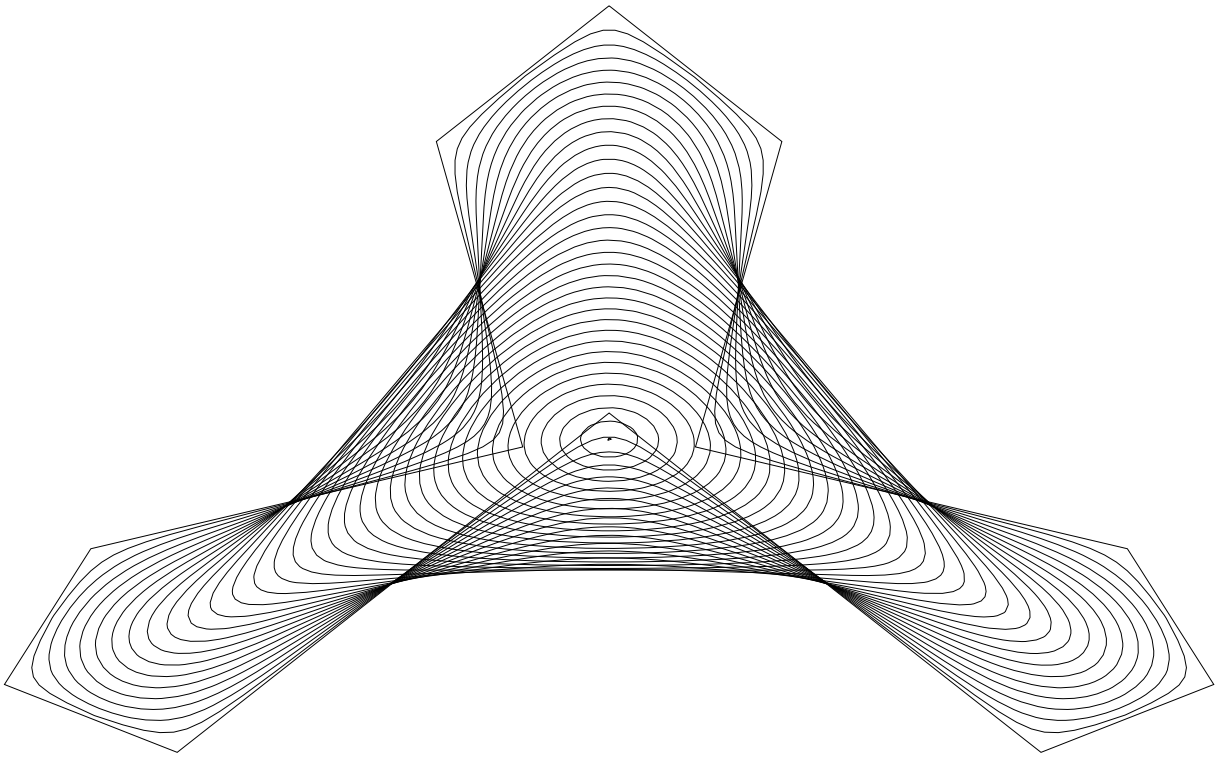

Figure 7.15: Affine scale space (clover polygon)

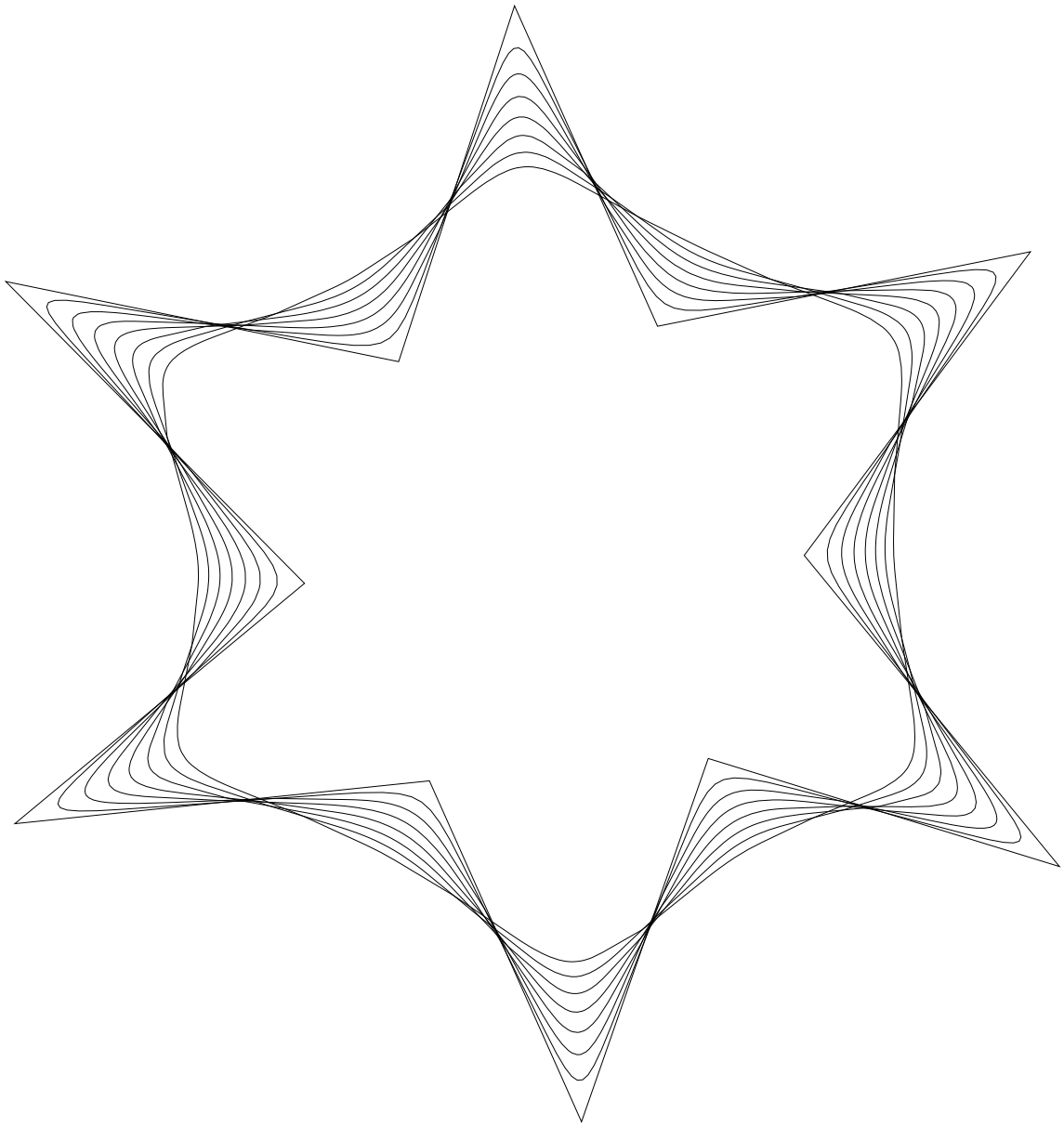

Figure 7.16: Affine scale space (non-symmetric star)

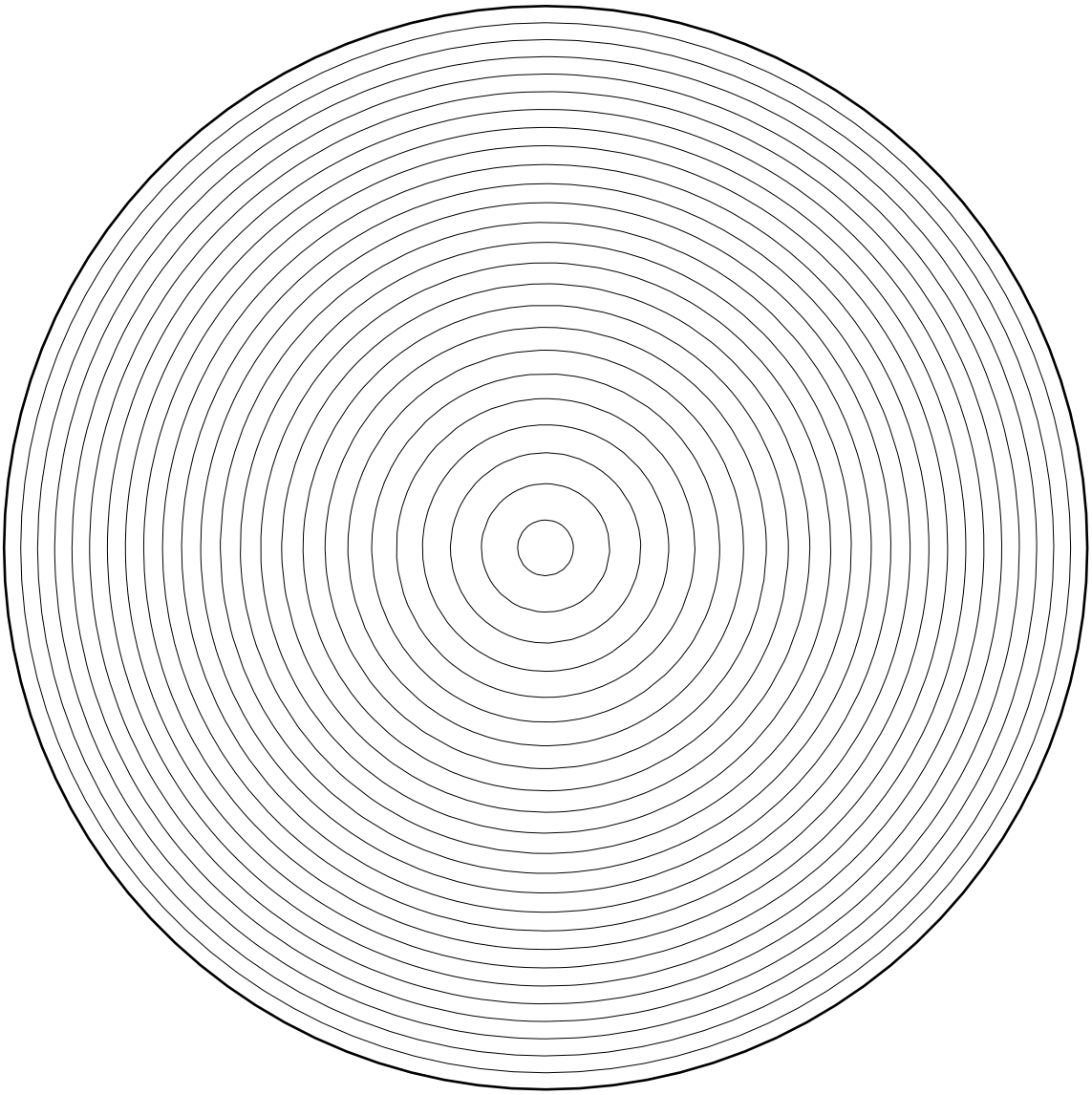

Figure 7.17: Affine scale space (exact circle)

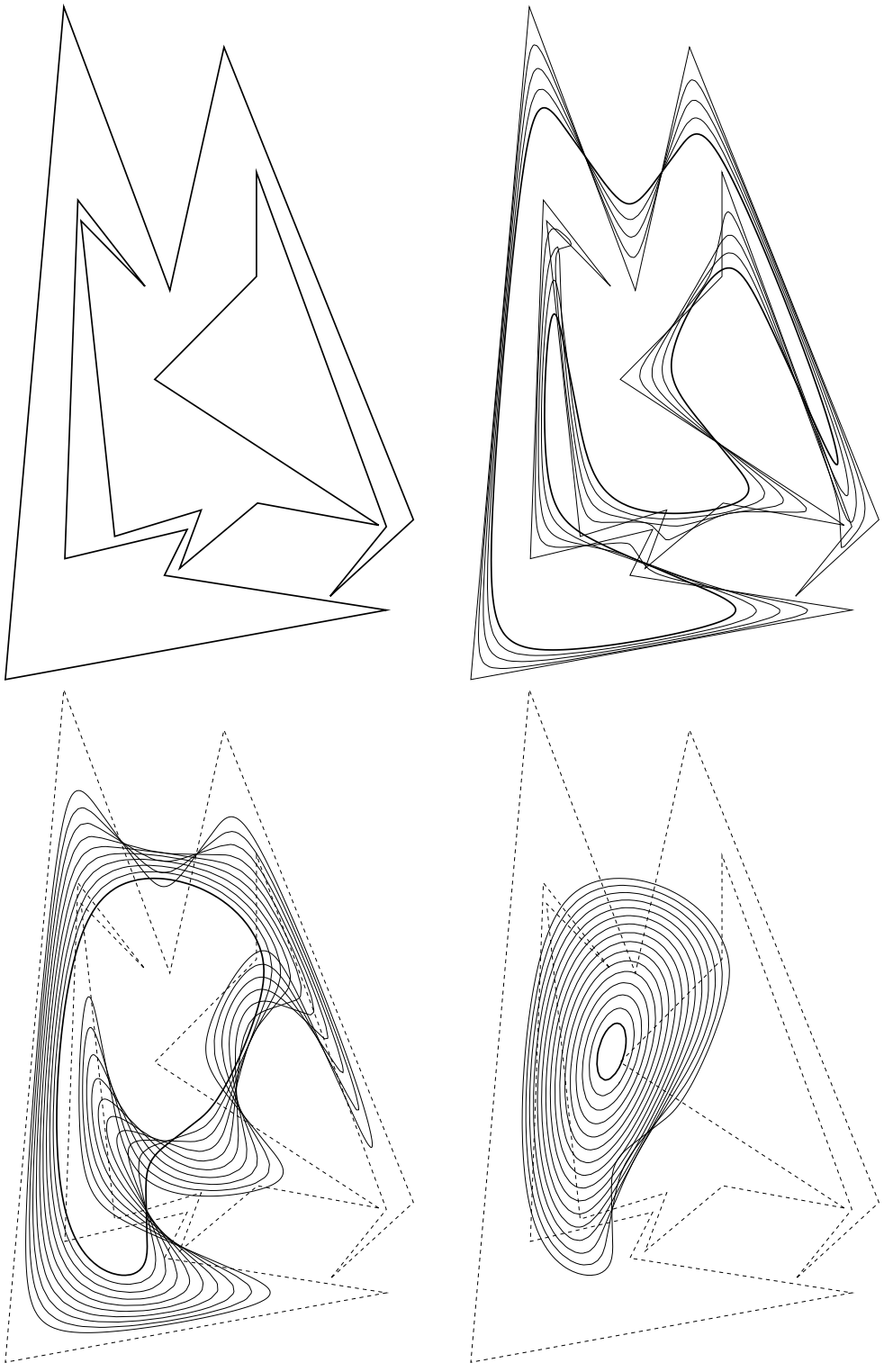

Figure 7.18: Affine scale space (weird polygon) — computation time : 6 minutes

7.2.2 Simplified algorithm

First, we check that the simplified algorithm give similar results to the exact one for the previous "weird" polygon : Figure 7.19 is quite similar to Figure 7.18, while the computation time is reduced to 7 seconds (instead of 6 minutes for the exact algorithm).

Then, we compute the affine scale space of large curves (about 4000 vertices and 1800 convex components for the initial curve represented on Figure 7.27). Notice the fine precision of Figure 7.28, which is impossible to attain with Sethian's algorithm for a reasonable amount of time and memory. For the "whale" polygon (Figures 7.23 to 7.26), the almost auto-intersections of the initial curve would probably cause any finite difference algorithm to fail, because the topological structure of the initial curve is very instable under a pixel discretization.

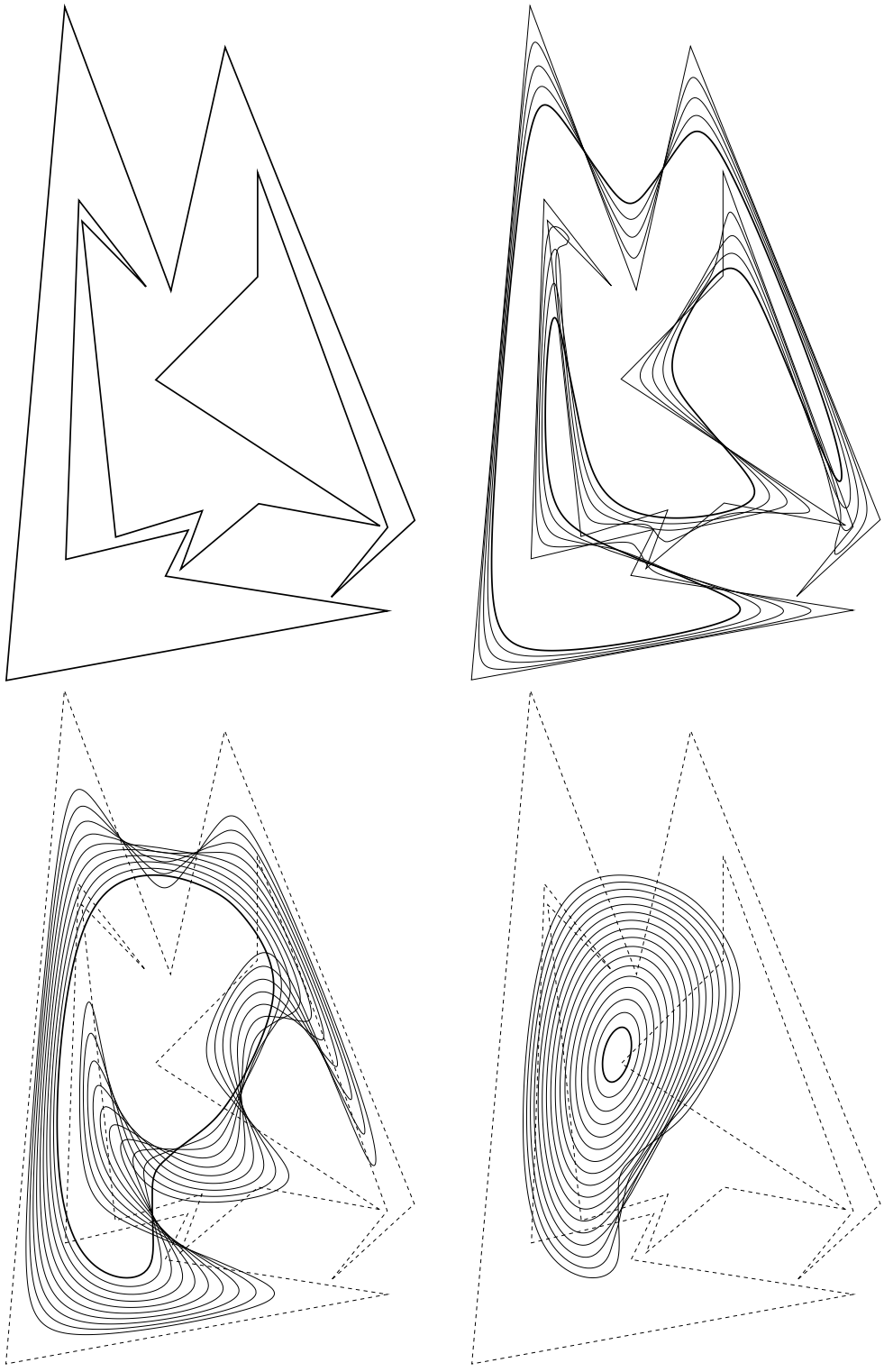

Figure 7.19: Affine scale space using the simplified algorithm (weird polygon) — computation time : 7 seconds

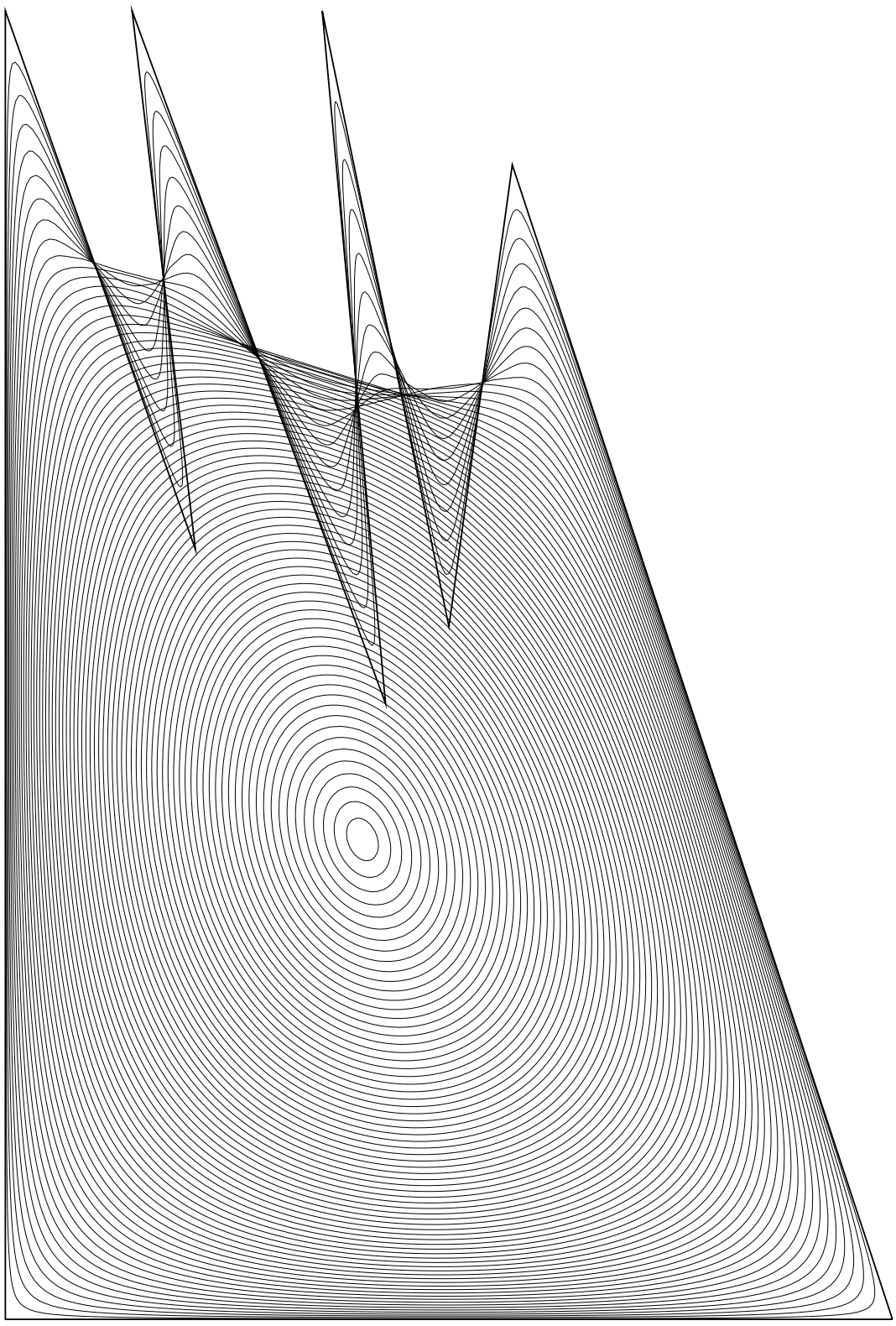

Figure 7.20: Affine scale space using the simplified algorithm (teeth polygon)

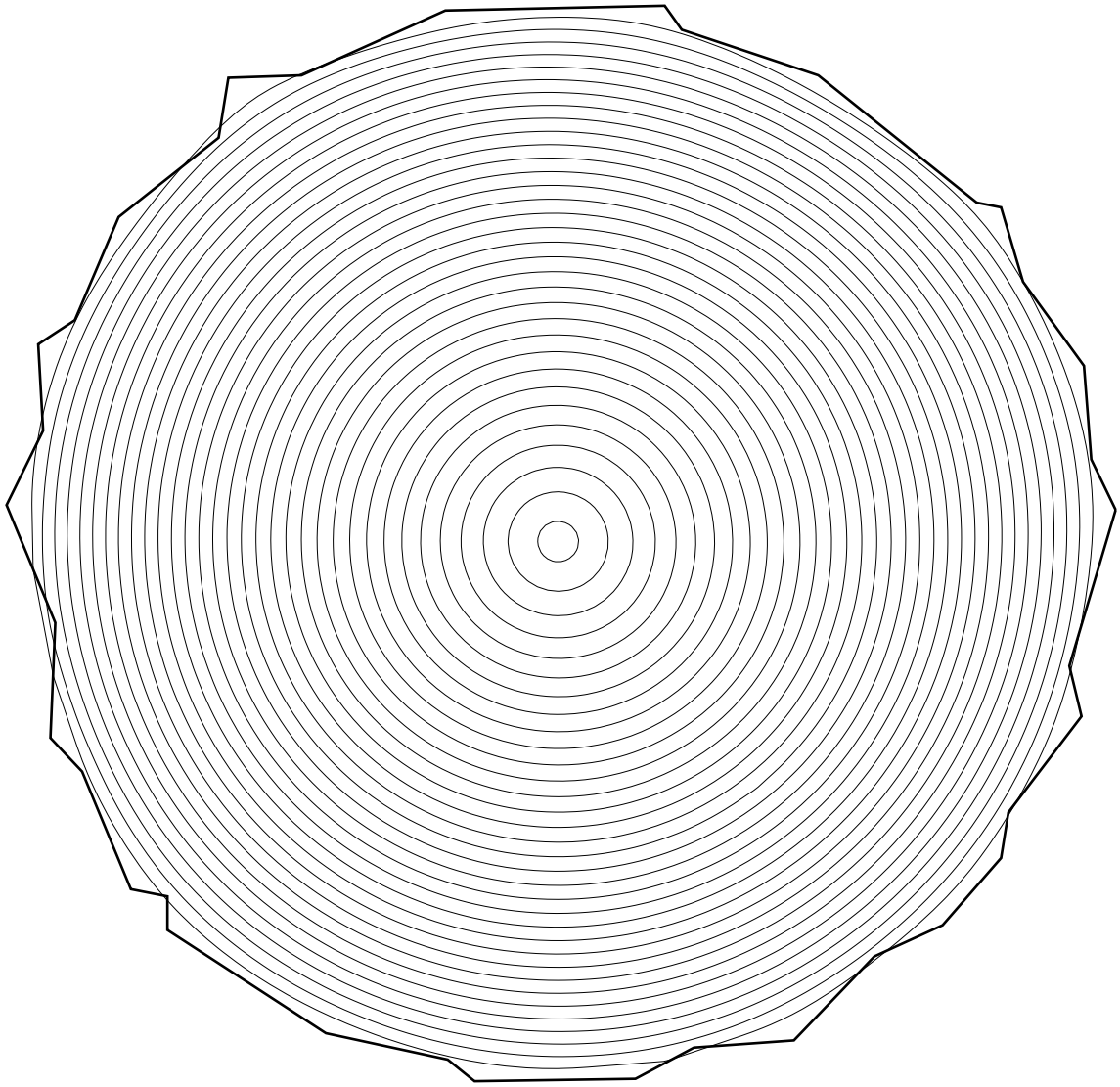

Figure 7.21: Affine scale space (rough circle)

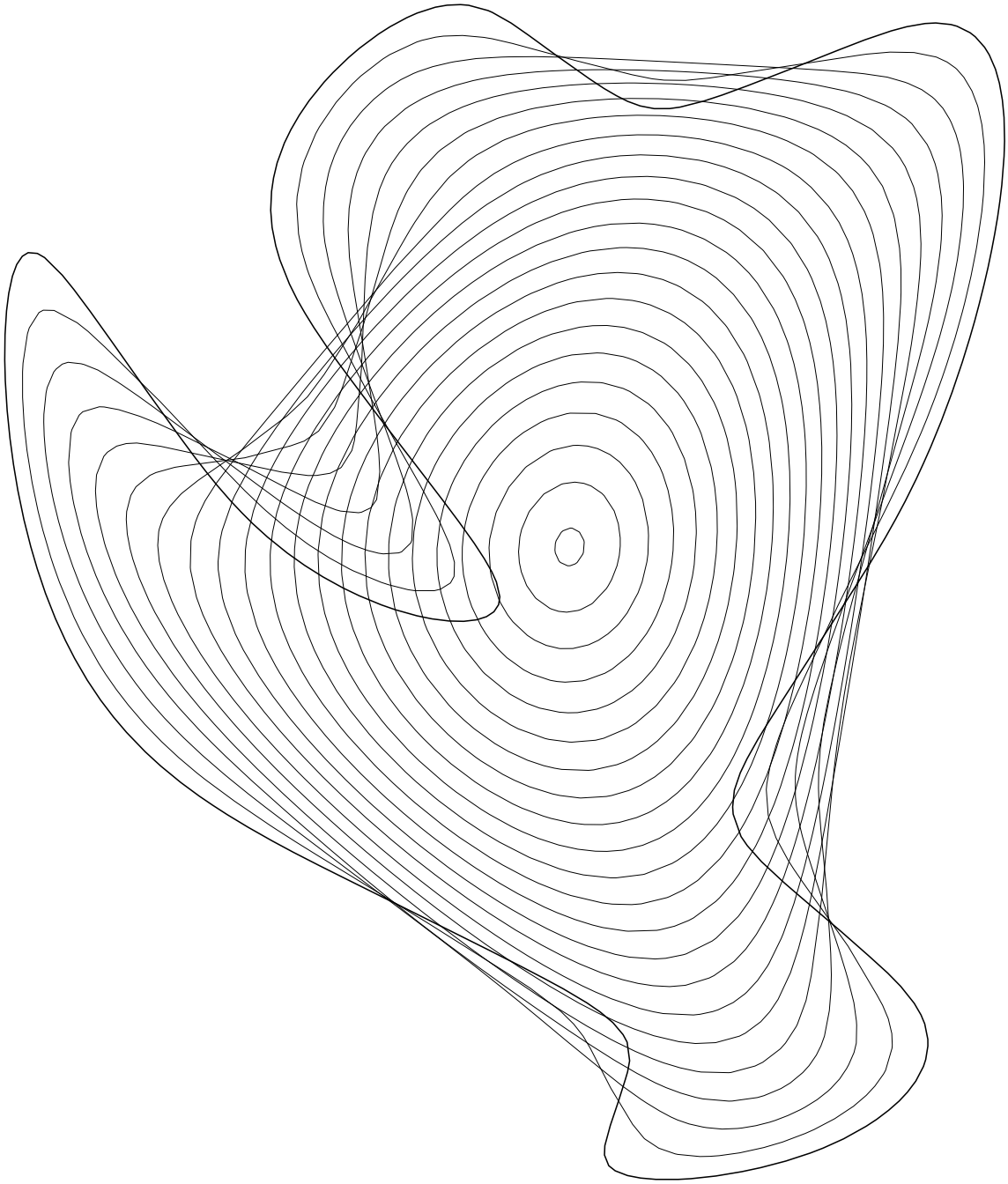

Figure 7.22: Affine scale space using the simplified algorithm (regular curve). The computation time is only 0.9 second, and the algorithm is stable despite the coarse quantization of curves we used here.

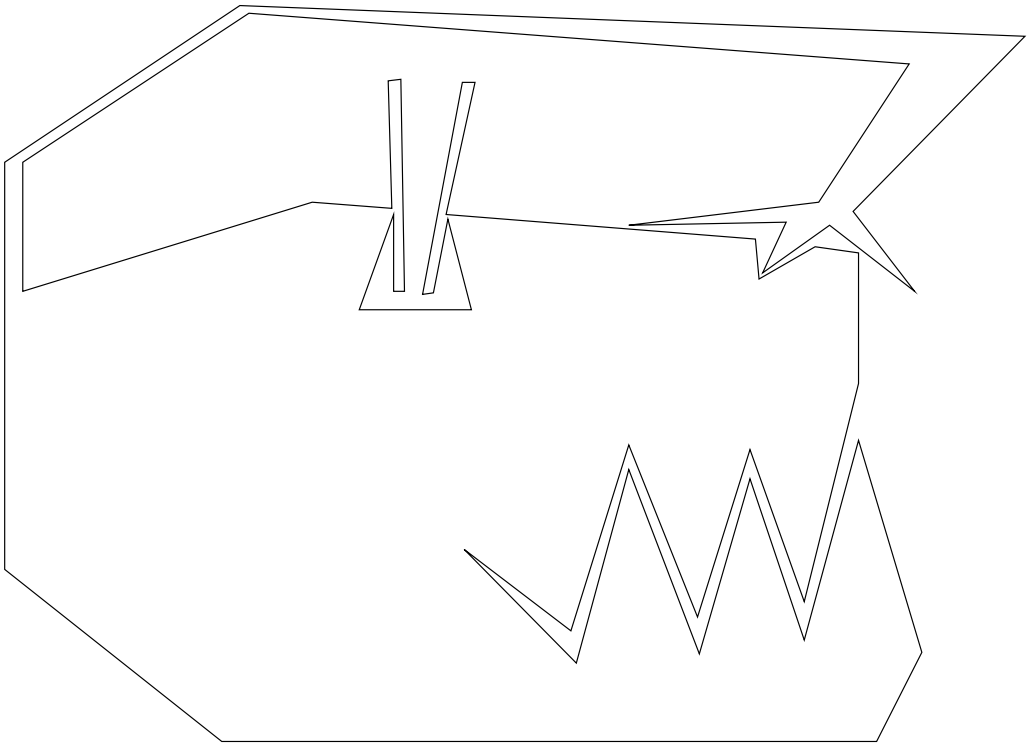

Figure 7.23: whale : initial curve (t=0)

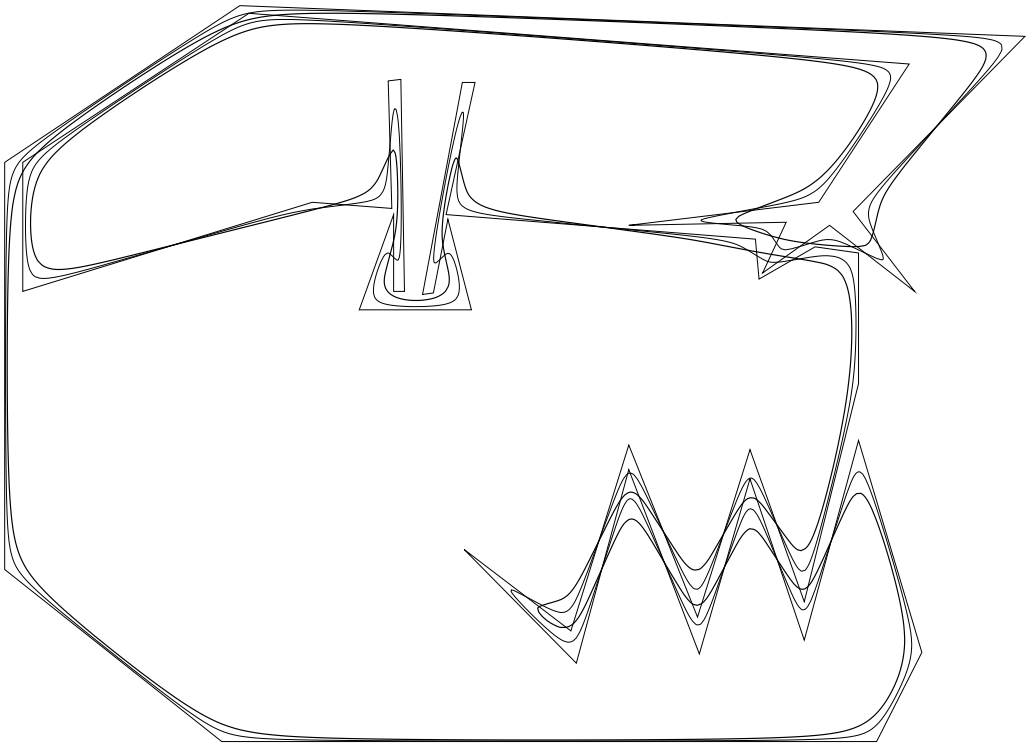

Figure 7.24: whale : filtered curve (t=100, 200)

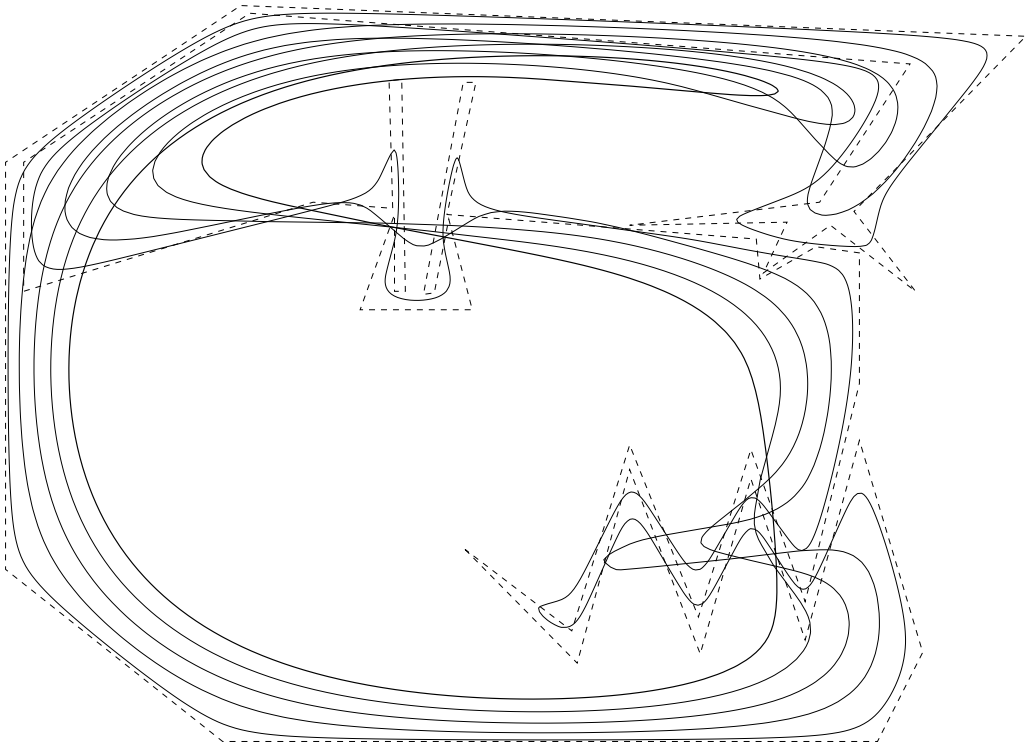

Figure 7.25: whale : filtered curve (t=1200, 2200, 3200, 4200)

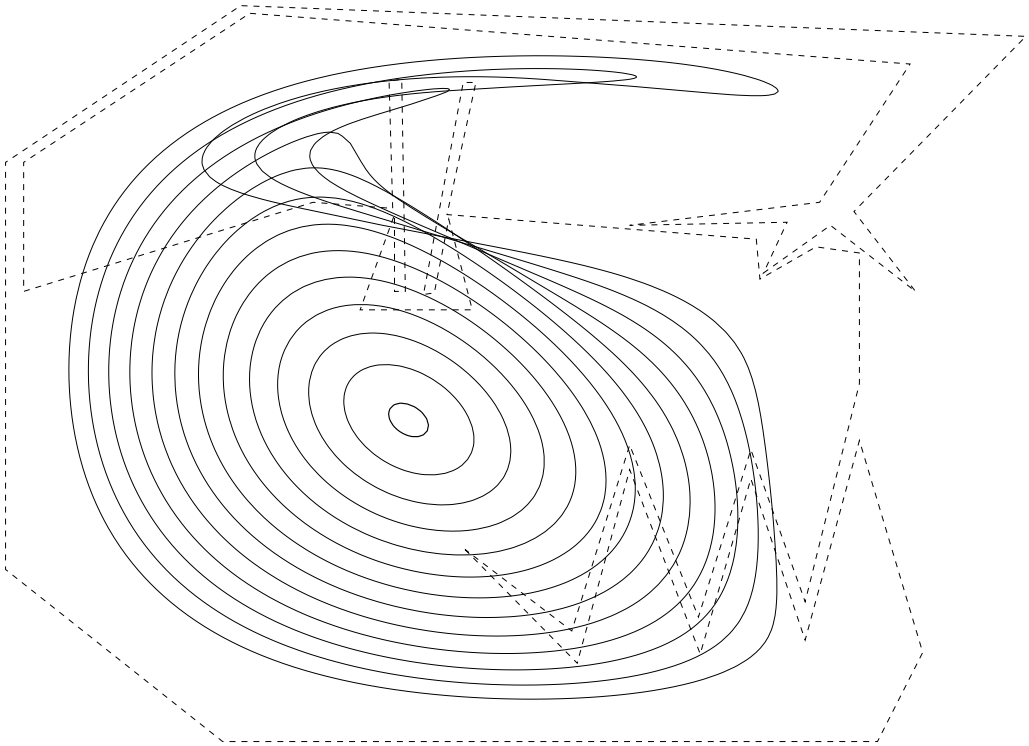

Figure 7.26: whale : filtered curve (t=5200, 6200, ..., 16200)

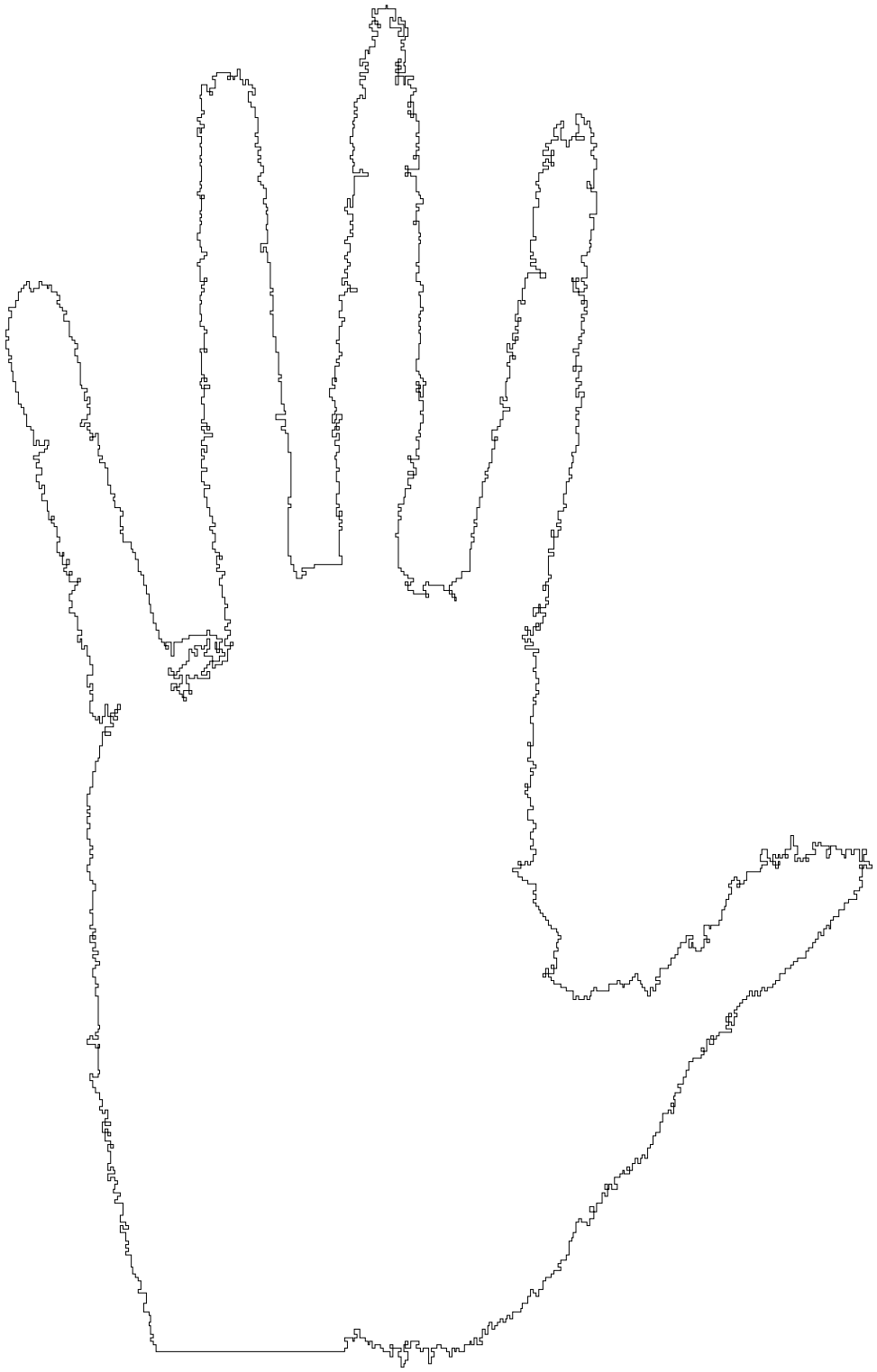

Figure 7.27: hand : initial curve (t=0)

Figure 7.28: hand : filtered curve (t=1)

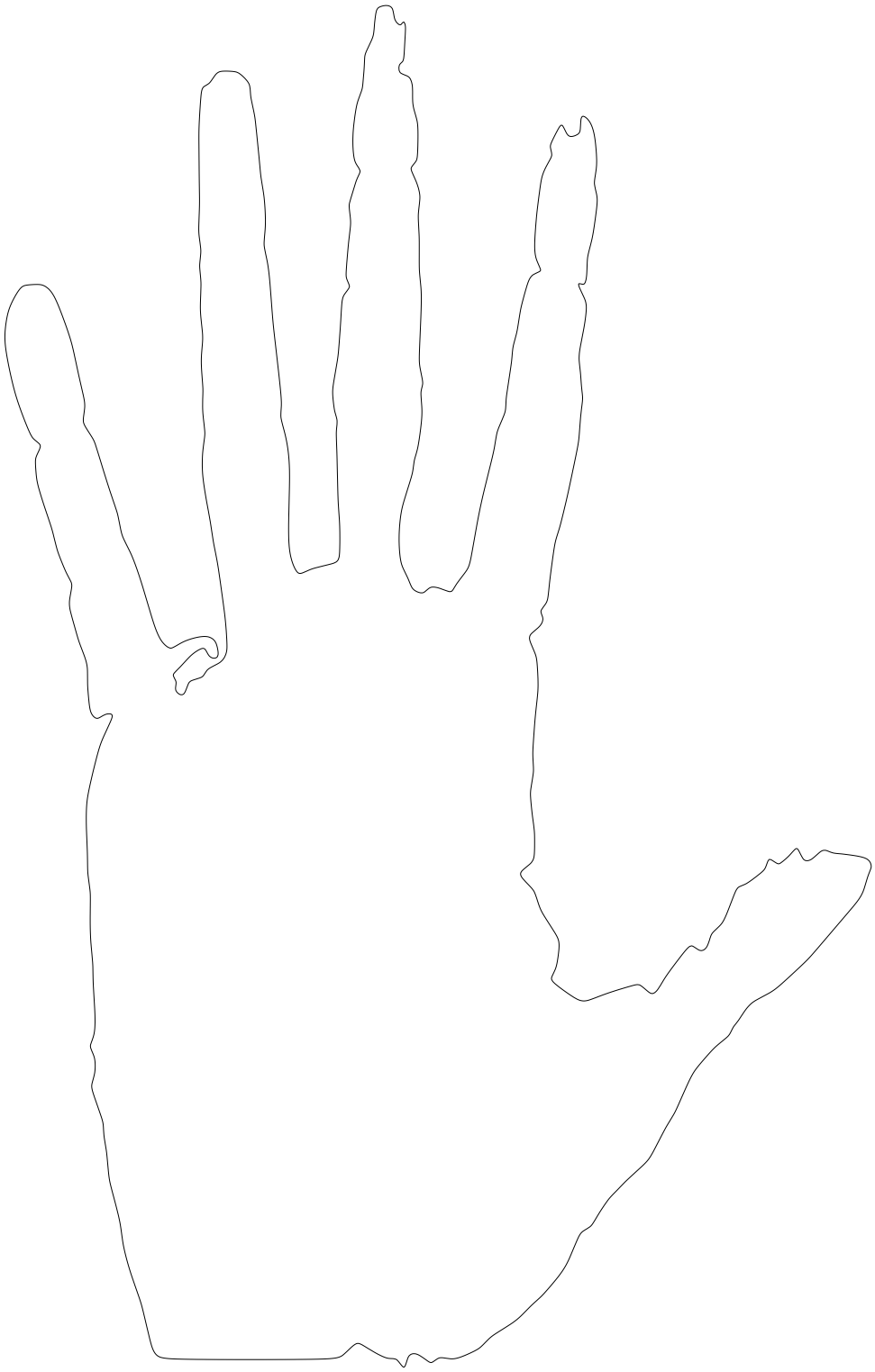

Figure 7.29: hand : filtered curve (t=8)

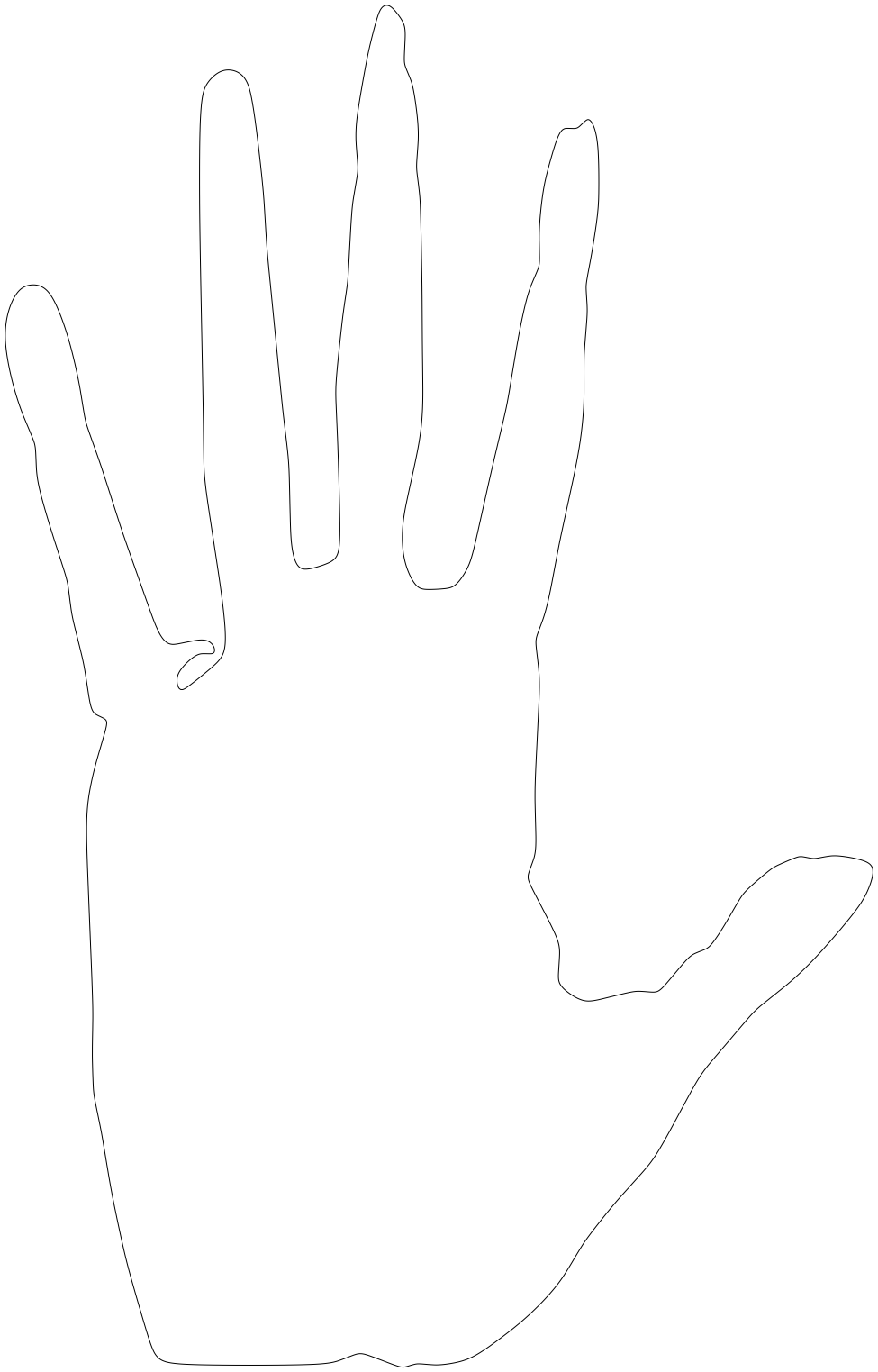

Figure 7.30: hand : filtered curve (t=20)

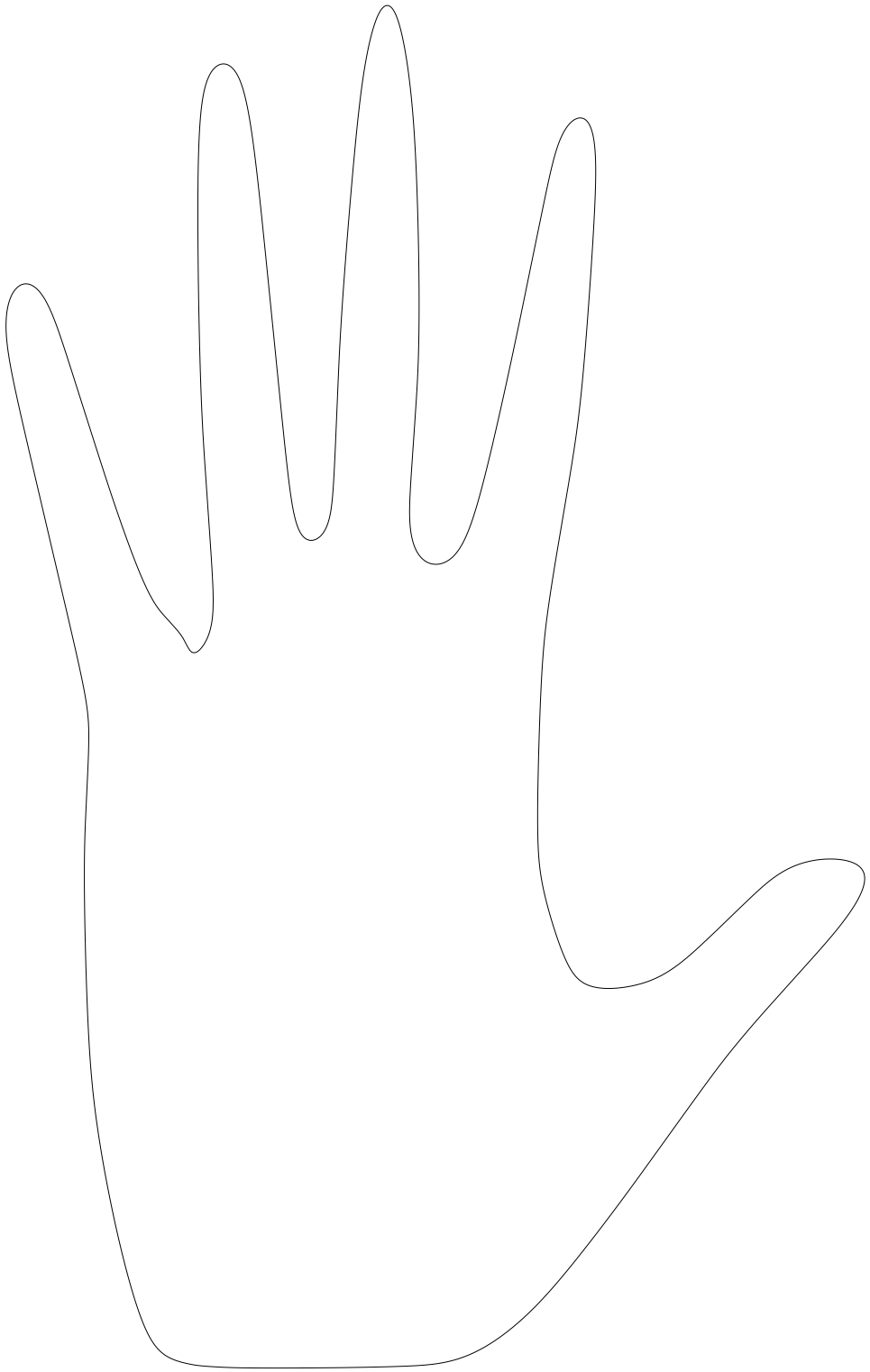

Figure 7.31: hand : filtered curve (t=200)

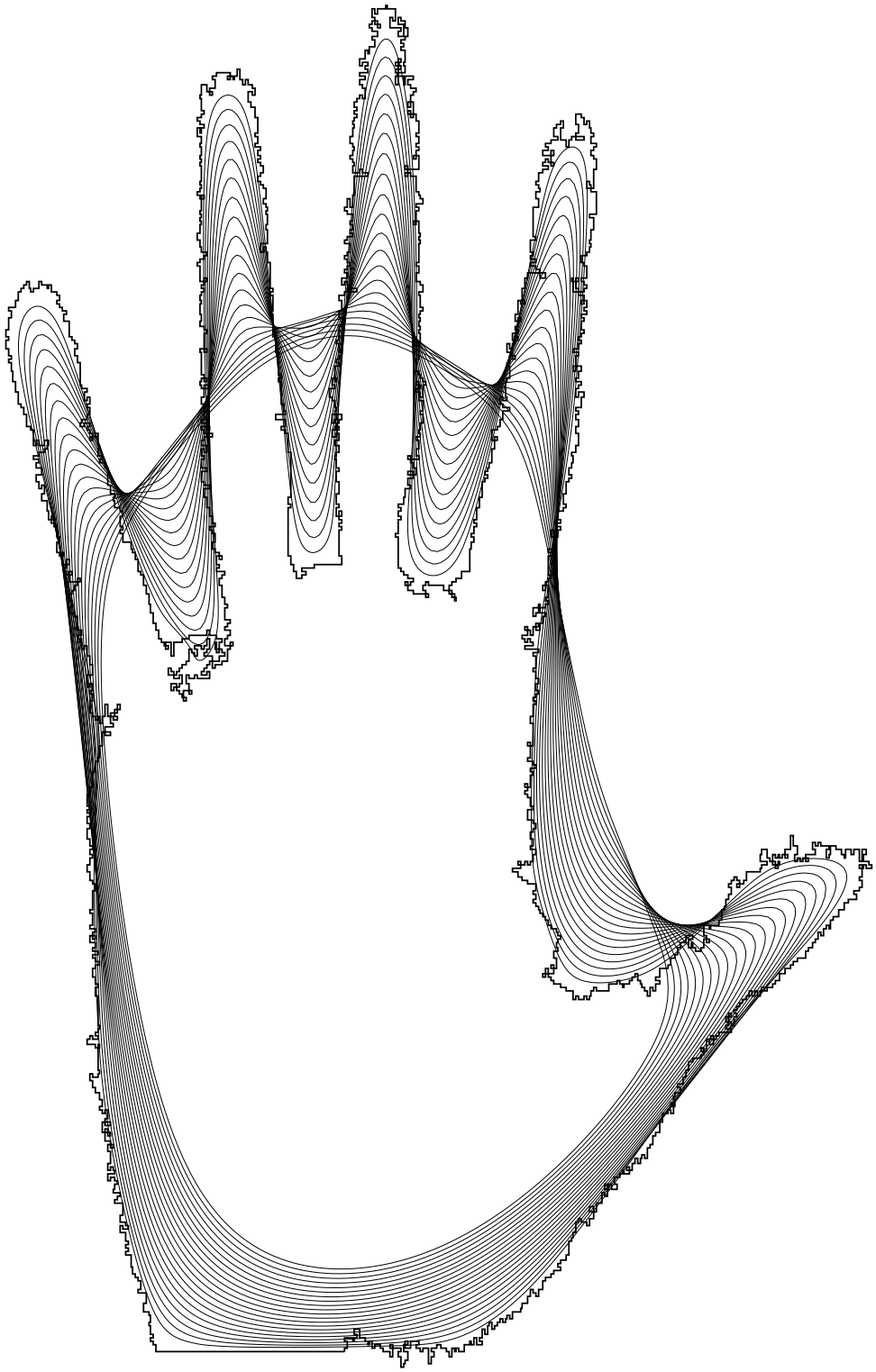

Figure 7.32: hand : filtered curve (t=300, 400, ..., 1300)

Figure 7.33: dog : initial curve (t=0)

Figure 7.34: dog : filtered curve (t=1)

Figure 7.35: dog : filtered curve (t=10)

Figure 7.36: dog : filtered curve (t=100)

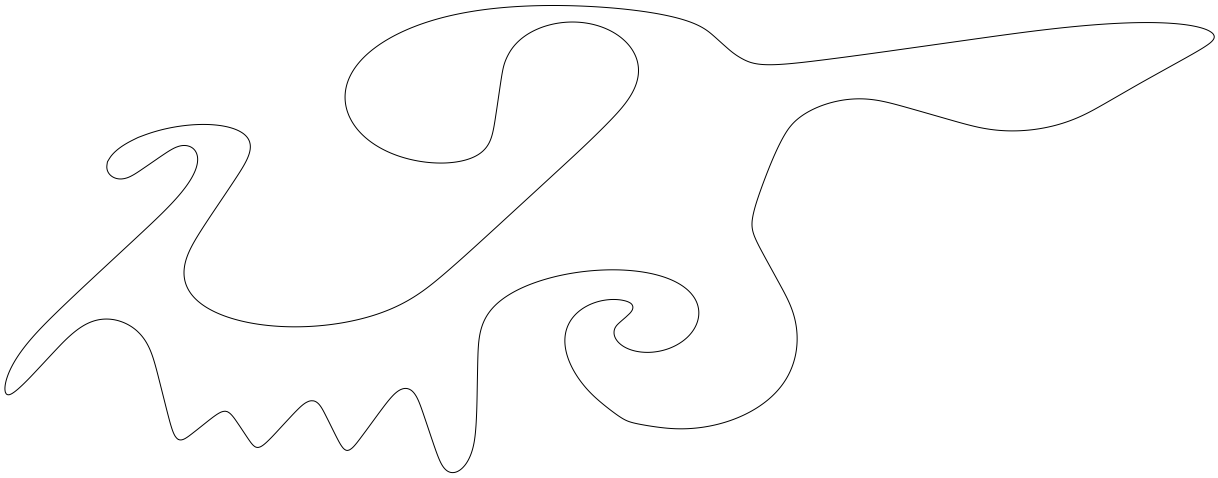

Figure 7.37: dog : filtered curve (t=1000)

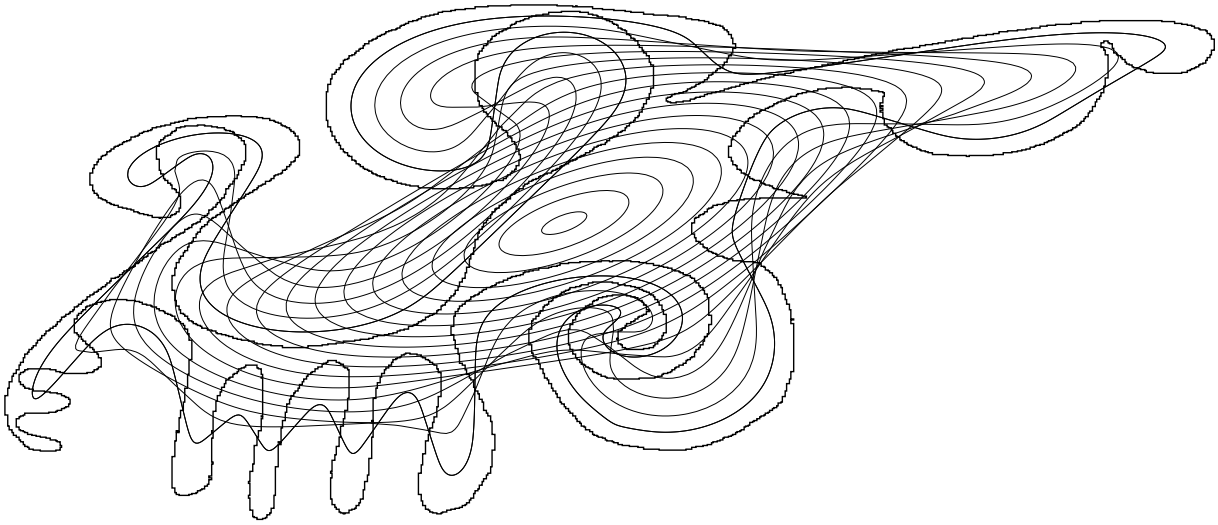

Figure 7.38: dog : filtered curve (t=1000, 2000, ..., 18000) — computation time : 2.5 seconds

7.3 Affine scale space of non-closed curves

Since the simplified algorithm allows to compute affine erosions of non-closed curves, it is possible to compute the affine scale space of a non-closed curve by iterating this operator. The need to consider the affine scale space for non-closed curves is explained in [21] : the affine scale space of non-closed curve can be defined thanks to a symmetrization-periodization process (Neumann condition) which makes the extremities fixed. If the two extremities are distinct, the asymptotic state is a segment. When they are not, a singularity may appear (see Figure 7.40).

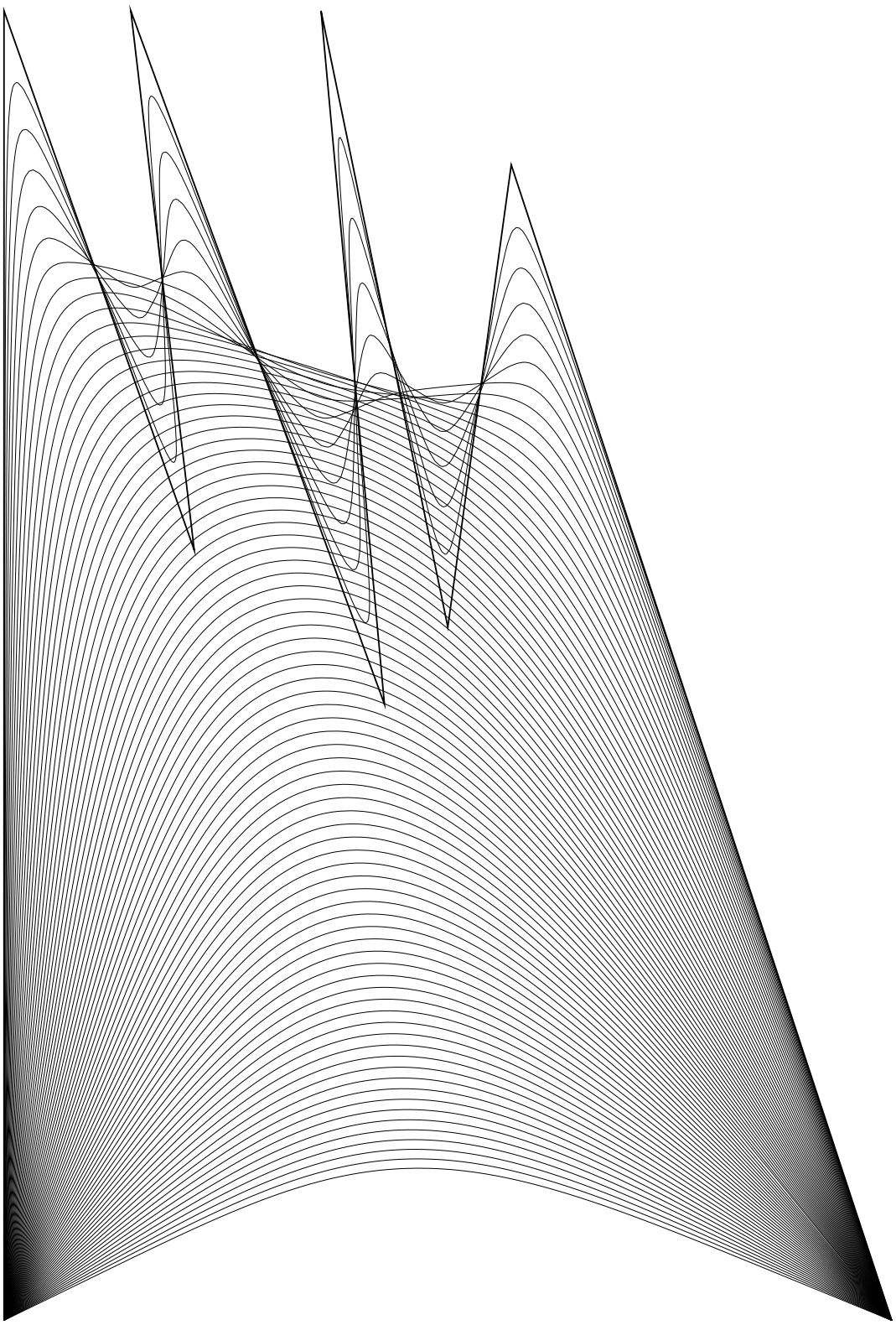

Figure 7.39: Affine scale space a a non-closed curve (modified teeth polygon)

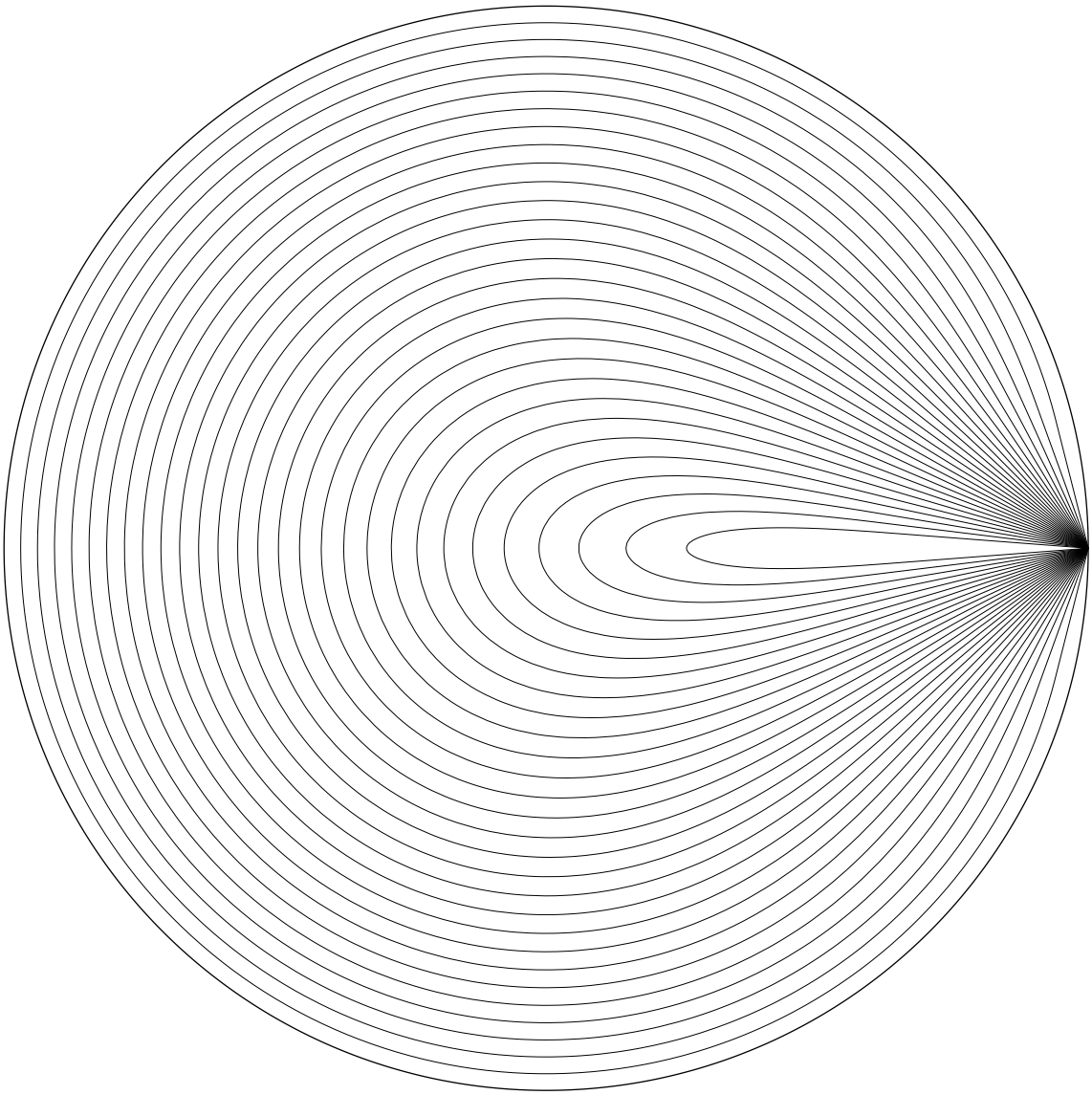

Figure 7.40: Affine scale space of a circle with a fixed point

Chapter 8

Conclusion

In this study, we presented a geometrical algorithm that can compute the affine scale space of a curve. Compared to Sethian's approach based on an image formulation, it is a faster[1] and more accurate method that allows complicated curve evolutions. Unlike classical geometrical schemes that rely on the computation of finite differences to perform point evolutions (see [65]), our scheme satisfies a natural property (the inclusion principle) that guarantees its numerical stability. It is also fully affine invariant, even in its numerical implementation (up to the computer precision). Thanks to these properties, the algorithm we proposed should be an interesting alternative to Sethian's approach, especially for shape recognition tasks (see [26]). We based our method on the iteration of a *non-local* operator which can be exactly computed on polygons. This property allows to separate the two approximation steps required in curve-evolution algorithms : the scale step, directly related to the number of iterations required, and the space step, i.e. the precision used in the discrete representation of curves. In our algorithm, the curve evolution between two iterations can be much larger than the quantization step used to represent the curves, while such a possibility is excluded for classical schemes in order to ensure their stability. The consequence is that our algorithm can accurately compute the evolution of a curve at a large scale in only a few iterations.

8.1 Applications

As we just explained, the main practical application of this study should concern Thierry Cohignac's method for local affine shape recognition (see [26]). Indeed, we can hope that his algorithm would gain computation time, robustness and accuracy by using our geometrical scheme to compute the affine scale space.

From a theoretical point of view, it would also be interesting to know what happens to the characteristic points of a curve when the evolution step t tends to zero. Our study states that

[1]Since the geometrical algorithm is much more precise than the scalar one, it is difficult to compare precisely their computation costs, but a proportion of 1 for 1000 gives a rough idea of it.

the characteristic area is asymptotically equal to $c \cdot t^{\alpha}$, c and α being universal constants, but it is likely that the second term of this expansion depends on the affine curvature[2], which would prove that the characteristic points of a curve tend to the extrema of a function of the affine curvature when t tends towards 0. Be that as it may, we now have an efficient way to compute the affine curvature on a curve, by considering the affine curvature of the pieces of hyperbolae which compose its affine erosion (for a small value of the area parameter of course). Hence, the shape recognition process can be realized by identifying new "characteristic" points defined as points where the affine curvature reaches an extremum.

Due to the duality of the image and curve formulation for the affine scale space, the iterated geometrical affine erosion also allows to compute the affine scale space of an image accurately. The computational cost is rather heavy since the geometrical scheme must be applied to every level curve of the initial image. However, we think that this way of representing an image without an inherent grid could be useful for some image processing tasks (zooming for example). Notice incidentally that this defines the first purely morphological numerical implementation of the AMSS which does not get "stuck" (see Chapter 2).

Last, the properties of the affine erosion we investigated in Chapter 3 might be useful in order to prove the existence of solutions for the geometrical affine scale space (which has not been done yet, as we explained in Chapter 2).

8.2 Further work

In Chapter 6, we defined two algorithms that compute the affine scale space of a curve : an exact algorithm, based on the iteration of the affine erosion, and a simplified algorithm, where the convex components of the evolving curve are processed separately at each iteration. We noticed that this simplified algorithm performs similar evolutions for a much lower computational cost. In fact, the computation cost of the simplified algorithm is proportional to the size of the input curve (that is, its complexity is linear), whereas in general this cost is approximately multiplied by the number of the convex components for the exact algorithm. For non-convex polygonal curves with more than 100 vertices (which correspond to a rather low precision for a complicated curve), the difference can become important. As it computes almost no intersection, the simplified algorithm is also more robust and easier to implement ("only" 900 lines of C source code). Hence, we think that it would be interesting to study more precisely the corresponding operator (the pseudo affine erosion) that we briefly introduced in Chapter 6. In particular, it should be possible to adjust the area step for each iteration automatically in order to obtain the best compromise between precision and computation time.

[2] we are sure that this is true for *one* term of the expansion at least, because the affine erosion would be the affine scale space otherwise.

We also think that it would be worthwhile investigating the case of non semi-closed curves further in relation with the work of V.Caselles, B.Coll and J.-M.Morel (see [21]). According to this paper, T-junctions should be kept fixed in order to perform an image evolution that preserves occlusions : from a geometrical point of view, this involves the evolution of non semi-closed curves.

Extending the affine erosion to higher dimensions seems difficult to achieve, above all in its numerical implementation. However, we think that the general idea we developed could be applied to several other planar curve evolutions. In particular, it is likely that several other geometrical curvature-driven evolution equations of the kind

$$\frac{\partial \boldsymbol{C}}{\partial t} = F(\gamma)\boldsymbol{N},$$

could be numerically simulated using the same method. The main point is to find a non-local operator satisfying three fundamental properties :

1. it is tangent to the evolution semigroup (i.e. consistent with the evolution equation),
2. it satisfies the inclusion principle,
3. it can be explicitly computed on a dense set of curves (polygons for example).

Condition 1 is obviously required. Condition 2 guarantees the numerical stability of the algorithm —which is fundamental for a curve evolution scheme— and allows large scale steps (and consequently a fast algorithm). Condition 3, which may be weakened, enables to process each iteration without depending on the quantization of the curve.

To give an example, let us investigate the case of Mean Curvature Motion (case $F(\gamma) = \gamma$), which is the Euclidean analog of the affine scale space. The affine erosion is based on an area criterion, since it removes from a set any of its chord set having area less than σ. Coming to Euclidean geometry, we can define a chord length-based erosion operator which removes from a set any of its chord set whose chord segment has length lower than a given δ (see Figure 8.1). Such an operator is consistent with the Mean Curvature Motion, and we think that it can be used to compute efficiently the Euclidean shortening of a curve.

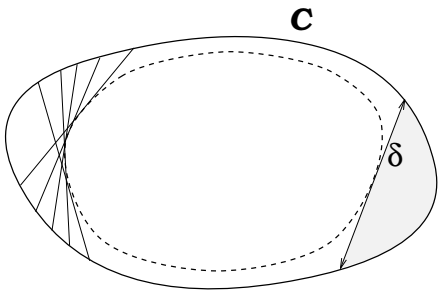

Figure 8.1: Euclidean analog of the affine erosion of a convex curve

The dashed curve is obtained by removing from the inside part of $\mathcal{C}$ any chord set whose chord has length δ. We conjecture that iterating such an operator leads to a good approximation of the Euclidean shortening flow associated to the Mean Curvature Motion.