Computational Geometry in the Human Brain

Kokichi Sugihara*
Meiji University, 4-21-1 Nakano, Nakano-ku, Tokyo 164-8525, Japan

Abstract

Geometric information processing in the human brain is very different from that in a computer: it is slow, local, and imprecise. However, humans are able to manage a huge amount of visual data, can understand the scenes in front of them, and thus can survive in their daily lives. We use visual illusions to investigate how the human brain treats geometric data, and we point out the similarities between the robustness of human geometric processing and the topology-oriented principle, which we have proposed for use in the design of robust geometric algorithms for computers by presenting a new algorithm for straight skeletons.

Keywords: visual illusion, human vision, robust geometric algorithm, topology-oriented approach, brain computing, Zöllner illusion, Ouchi illusion, impossible motion, straight skeleton

1 Introduction

Computational geometry is the field in which geometric algorithms are designed for computers [5, 7, 17]. Computers are much more precise than human brains, and hence the main concern is to make these algorithms as efficient as possible [1]. Indeed, a huge number of very efficient algorithms have been established, and sometimes these are the most efficient, i.e., optimal, in terms of the order of the computational time with respect to the problem size. In this sense, computational geometry is one of the most successful areas of computer science.

However, we note that computational geometry mainly treats well-defined problems, while in the real world, we encounter many geometric problems that are not well defined and cannot be solved easily by the current techniques of computational geometry. Such problems include, for example, image pattern recognition and scene understanding [3].

The human brain, on the other hand, seems to be able to solve those problems relatively easily. We receive geometric information about the world around us in the form of projected images on the retina, and our brains process those

*This manuscript was published in *Discrete and Computational Geometry and Graphics*, J. Akiyama et al. (eds.), LNCS, 8845 (2013), pp. 145–160 (Springer) images and understand the scenes in front of us without any major difficulties. This ability is surprising when we recall that the computations of the human brain are slow and imprecise, compared to electronic computers. If we can understand the way in which human brains perform geometric processing, we may be able to apply it to the construction of algorithms for ill-defined geometric problems.

Motivated by this observation, we used visual illusions to study the way in which the human brain processes geometric data, in order to find an approach for solving ill-defined problems in computational geometry.

On the other hand, our research group has long been studying an approach to the design of robust geometric algorithms, which we call the topology-oriented approach [22, 24, 26]. In this approach we start with the assumption that numerical errors cannot be avoided and moreover the amount of errors is not bounded a priori, but still we aim at robust geometric algorithms. This task is ill-conditioned because the correctness of the algorithms cannot be guaranteed due to numerical errors. However, we can successfully construct stable algorithms by guaranteeing the consistency of topological structures of geometric objects and thus circumvent failures.

We first applied this idea to an incremental algorithm for ordinary Voronoi diagrams [25], and then extended to various geometric problems including Voronoi diagrams for line segments [26] and line arrangements [10] in the plane, and convex hull [15], Voronoi diagram [26] and polyhedra [20] in the three-dimensional space.

Therefore, we might regard the topology-oriented approach as an example of human-like robust computation. In this paper we compare human brain processing with the topology-oriented algorithms and discuss their similarities.

The structure of this paper is as follows. We first observe and discuss three typical examples of visual illusions, the Zöllner illusion [11], the Ouchi illusion [14], and the impossible motion illusion [21], in Sections 2, 3, and 4, respectively. In Section 5, we construct a new algorithm for robust computation of the straight skeleton as an example of a geometric problem, and discuss the similarities between the computations in the human brain and the topology-oriented algorithms. We present our concluding remarks in Section 6.

2 Zöllner Illusion and Overestimation of Acute Angles

Fig. 1 shows the famous Zöllner illusion; the four long, straight lines are exactly parallel and horizontal, but they look as if they are alternately slanted in opposite directions. This optical illusion is evoked by the shorter lines crossing the longer lines, and it is usually explained by the overestimation of the acute angles.

When two lines cross, they generate two acute angles and two obtuse angles. It is commonly observed that the acute angles are apt to be perceived larger

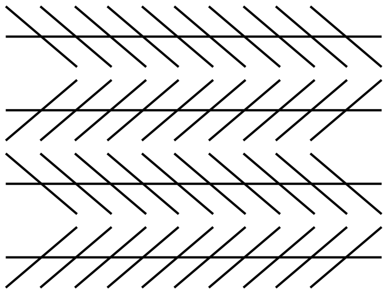
Figure 1: Zöllner illusion.

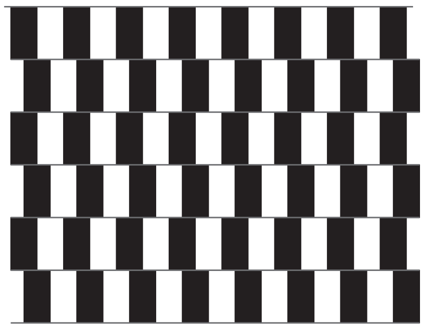
Figure 2: Café Wall illusion.

than they actually are, and the obtuse angles are apt to be perceived smaller. There are many other illusions explained in the same way, including the Hering illusion, the Wundt illusion, and the Luckiesh illusion [11].

Various mathematical models have been proposed to explain this overestimation of acute angles. A typical such model is the one by Fremüller et al. [9]. In their theory, the retina, acting as a photo sensor, has finite resolution, and hence images are blurred, resulting in greater rounding of acute angles than of obtuse angles. This makes acute angles appear to be greater than the actual angles.

According to this mathematical model, we can strengthen other types of slanted-line illusions, such as the Café Wall illusion [11] shown in Fig. 2. In this figure, in each row, white and black rectangles alternate in the horizontal direction, and each row is offset from the adjacent rows by half the width of the rectangles. Although the lines between the rows are straight and horizontal, they appear to be curved and sloped. This illusion can also be explained by the Fermüller model [9].

Now, since we know that acute angles will be perceived to be larger than they are, we can expect that this illusion will become stronger if we distort the rectangles into parallelopipeds, since this will generate a series of acute angles. The parallelopiped-based Café Wall pattern is shown in Fig. 3. We can observe

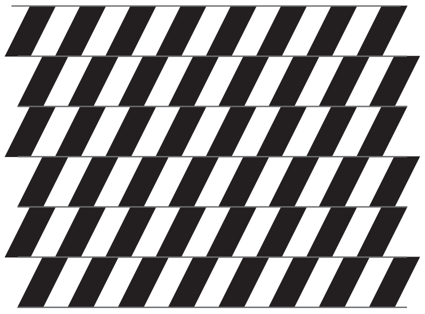
Figure 3: Stronger version of the Café Wall illusion (Sugihara, 2012).

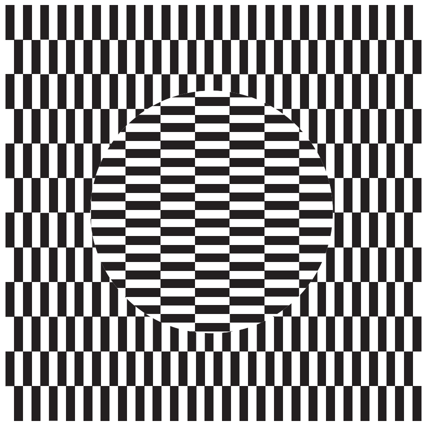
Figure 4: Ouchi illusion (adopted from [16]).

that the illusion is stronger, as was predicted by the mathematical model.

These observations, together with the mathematical model, tell us that human visual perception is affected by blurring, even though we feel that we see the figures accurately. We can summarize this observation in the following way.

Observation 1. The computations in the human brain are imprecise.

3 Ouchi Illusion and Local Motion Detection

The second example we consider is the Ouchi illusion. The picture shown in Fig. 4 is included in Ouchi's book [16]. This is a still picture, but the central circular area seems to drift at random, independently from the surrounding area. This drift illusion can be explained in the following way.

First, our retina, which is an array of photo sensors, usually has some slight random motion. A sensor generally decreases its sensitivity if it detects the same signal for a relatively long period of time. This is also true of the retina.

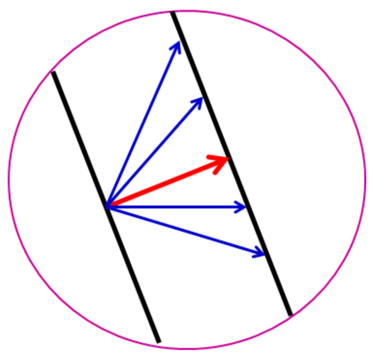

Figure 5: Aperture problem.

In order to avoid this decrease in sensitivity, the retina tries to use different sensors to detect a given signal. This is why the retina moves, and hence, when we look at a picture, the image moves on the retina.

Secondly, neurons in the brain, particularly the neurons used in the earlier stages of processing, cover only a small area of the retina. Hence, they process only the local information. This causes ambiguity in the direction of the detected motion, in the following sense.

Suppose that, as shown in Fig. 5, a neuron receives visual data in a small circular area of the retina, and detects displacement of an edge for which both terminal points are outside the circular area. Then, the neuron can tell that the edge moves, but cannot tell in which direction. There are many possibilities for the direction of the motion, as shown by the arrows in Fig. 5. This ambiguity is called the "aperture problem" [13]. In other words, a neuron can only detect motion perpendicular to the edge, no matter which direction the edge actually moves.

On the basis of these two observations, we can explain why the Ouchi illusion arises. Suppose that the Ouchi pattern moves slightly on the retina. In the central part of the Ouchi pattern, horizontal edges prevail, and, consequently, primarily vertical motion will be detected. In the surrounding part, on the other hand, vertical edges prevail, and, consequently, primarily horizontal motion will be detected. As the result of this, the central and surrounding parts appear to move differently. This is a typical way of explaining the Ouchi illusion [8].

One might think that this illusion would become stronger if the elongated checkerboard patterns were replaced with stripes, because the edge directions would then be more uniform. However, this is not true. The illusion becomes weaker if we replace the central part of the Ouchi pattern with horizontal stripes and the surrounding part with vertical stripes. This can be understood in the following manner. When a neuron becomes excited, it suppresses the excitation of its neighboring neurons. This is called lateral inhibition. If a moving edge is long, the excitation of a neuron covering part of the edge will suppress

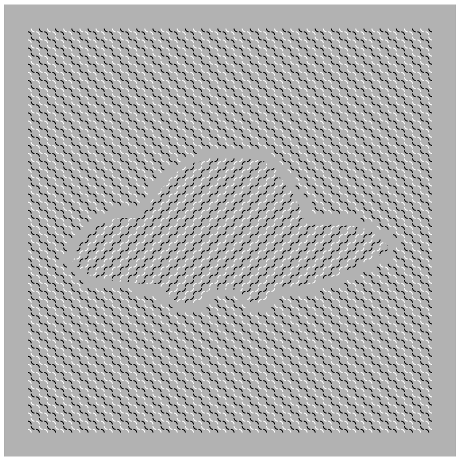

Figure 6: Drift illusion “UFO in the evening glow” (Sugihara, 2013).

the excitation of the neurons covering the neighboring parts. In this way, the excitations of neurons cancel each other, and the illusion becomes weaker.

A straight edge will stimulate only those neurons that detect the direction perpendicular to that edge. If the edge direction deviates slightly, such as like a sine curve, it will stimulate more neurons because the edge contains many directions. We can thus expect that the illusion will become stronger if the straight edges are replaced with slightly curved edges.

Based on these observations, we can create patterns that will give a stronger illusion of drift than does the Ouchi pattern. An example of such a pattern is shown in Fig. 6.

From this illusion, we get the next observation of the nature of the human brain.

Observation 2. The basic computations in the human brain are local.

4 Impossible Motion Illusion

Impossible motion is a new type of illusion evoked by a three-dimensional object. Fig. 7 shows an example of impossible motion called “magnet-like slopes”. Panel (a) shows an object with four slopes. We initially perceive that the four slopes each go down in a different direction from the high center. However, if we place balls on the slopes, they appear to roll uphill toward the high center, defying gravity, as shown in panel (b). Panel (c) shows another view of the same situation; here, we can see that the center is the lowest point, and the balls are just rolling downhill. Thus, the actual motion obeys gravity, but it appears to be an impossible motion that defies physical laws.

This class of visual illusion comes from the fact that a single picture lacks

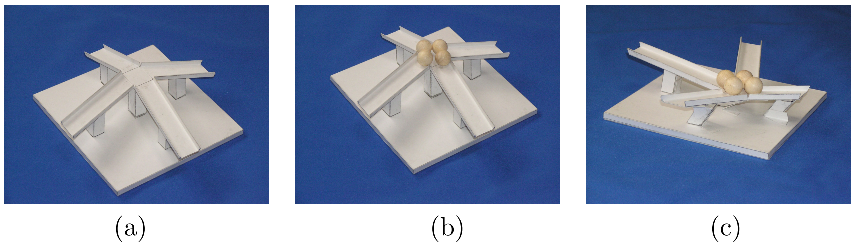

Figure 7: Impossible motion “Magnet-Like Slopes” (Sugihara, 2009): (a) three-dimensional object; (b) result of apparently impossible motion; (c) another view of the object.

depth information, and so the human brain guesses at the most common solid among infinitely many possibilities that are consistent with what is seen. This illusion was found as a byproduct of computer vision research [18, 19]; refer to [19, 21] for the details of designing this class of illusion.

A remarkable aspect of this illusion is that even after we understand the actual shape of the object, as in Fig. 7(c), we incorrectly perceive the shape when we return to the vantage view point shown in Fig. 7(a). This observation may be expressed in the following way.

Observation 3. Computations in the human brain persistently retain the initial interpretation.

5 Robust Geometric Computations Suggested by the Human Brain

As we have observed, computations in the human brain are imprecise, local, and persistent. However, in spite of these disadvantages, the human brain still can robustly and efficiently process visual geometric data in our daily lives. In this section, we consider how these remarkable characteristics of the human brain can be used in the design of algorithms for computers.

Geometric algorithms are usually designed on the assumption that numerical computations will be done precisely, and hence, in particular, that geometric predicates will always be evaluated correctly. However, this is not true in real computers, and theoretically correct algorithms sometimes fail when they are implemented as software. This failure is common when the input is very close to a degenerate situation.

Let us take the straight skeleton as an example. Let P be a polygon in the plane. Suppose that from each edge of P, two copies of the edge, we will call them the sweep lines, start moving in opposite directions away from the edge and at the same speed, and that they continue to maintain contact with the neighboring sweep lines at the terminal vertices. Hence, the sweep lines change

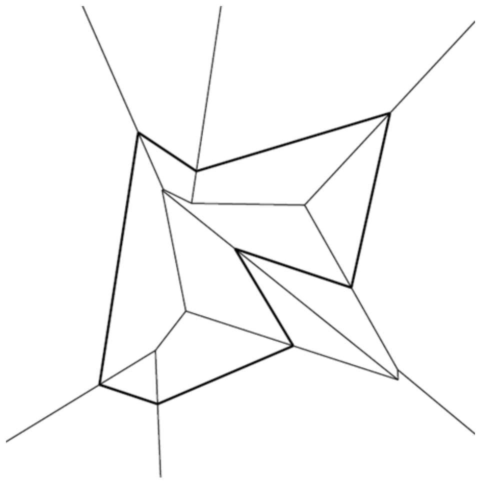

Figure 8: Polygon and its straight skeleton.

their lengths as they move. The motions of the sweep lines terminate at the points of collision with the other sweep lines. The region swept by a sweep line is assigned to the corresponding edge. In this way, the plane is partitioned into the regions swept for each edge and their boundaries. This partitioning, the boundary structure in particular, is called the *straight skeleton* of the polygon P [2]. Fig. 8 shows an example of a polygon (thick lines) and the corresponding straight skeleton (thin lines). The straight skeleton has applications in many fields, such as paper folding [6], solid modeling [27], and pop-up cards [23].

The straight skeleton can be defined for a more general structure, a straight-line graph, and Huber and Held [12] constructed an $\mathrm{O}(n^2 \log n)$ algorithm for the generalized straight skeleton. For the straight skeleton of a simple polygon, an $\mathrm{O}(n^{3/2} \log n)$ algorithm is known [4]. However, these algorithms are theoretical in the sense that they are designed on the assumption that numerical computation is done precisely. Therefore they are not necessarily valid in actual computers because of numerical errors. In this paper, we apply the topology-oriented principle [22,25] which we have developed for designing robust geometric algorithms, and construct a new algorithm which is robust against numerical errors and which runs in $\mathrm{O}(n^2 \log n)$ time, and thus show that the topology-oriented approach is similar to persistent human brain computation.

The straight skeleton can be interpreted as a roof structure in three-dimensional space, in the following manner. Suppose that the polygon P is the shape of the wall seen from above, all parts of the wall have the same height, and we want to construct a roof structure in which all parts have the same angle of declination. The straight skeleton of P tells us how to partition the roof into planar plates so that this is possible. Indeed, to build this roof, the region assigned to an edge of P should be elevated to a roof plate passing through the edge at the top of the wall.

On the basis of this interpretation, we can construct a sweep algorithm for the straight skeleton. Let π be a horizontal plane that is initially placed on top of the wall. We let π sweep upward and, in this way, construct the roof structure inside P step by step from the lower part to the highest point on the roof. Next, we let π sweep downward and thereby construct the remaining part of the roof structure outside of P.

This idea can be summarized in the next algorithm, where we concentrate on the construction of the straight skeleton inside P.

Algorithm 1 (Straight skeleton).
Input: polygon P and angle α between the roof plates and the horizontal plane.
Output: straight skeleton inside P.
Procedure:

```
1. For each edge $e$ of $P$, generate the half-plane containing $e$ that is toward the
   inside of $P$, forming angle $\alpha$ with respect to the horizontal plane, and put it
   into storage $S$. (We will call the elements of $S$ the roof plates.)
2. For each vertex $v$ of $P$, generate the half-line at the intersection of the two
   roof plates that are associated with the two edges incident to $v$, and put it
   into storage $E$. (We will call the elements of $E$ the roof edges.)
3. For each edge $e$ of $P$, trim the corresponding roof plates in $S$ by the two roof
   edges emanating from the two terminal vertices of $e$.
4. Create empty storage locations $\overline{E}$ and $\overline{S}$.
5. Repeat Steps 5.1, 5.2, and 5.3 until $E$ and $S$ are empty.
   5.1 Find a pair $(e, s)$ of a roof edge $e$ and a nonneighboring roof plate $s$ such
       that they intersect and the point of intersection is the lowest among all
       such pairs.
   5.2 Move $e$ from $E$ to $\overline{E}$.
   5.3 Increment the roof structure around the point of intersection (details of
       this procedure will be shown below). If new roof edges are created, add
       them to $E$. If the roof plates in $S$ become completely bounded by roof
       edges, move them from $S$ to $\overline{S}$.
6. Output the roof structure consisting of the roof edges in $\overline{E}$ and the roof
   plates in $\overline{S}$.
```

□

Intuitively, this algorithm constructs the roof structure below the sweep plane π step by step as π moves upward. However, an inconsistency may arise due to numerical errors in the computations. Let P be the regular polygon shown in Fig. 9(a). In theory, all the roof edges emanating from the vertices meet at a common point. However, in the real world, we have numerical errors, and hence it is difficult to identify this common point of intersection. Instead, the algorithm may find many points of intersection. Suppose that Algorithm 1 first finds the point of intersection of a roof edge and a roof plate as shown in Fig. 9(b), and next finds another pair, as shown in Fig. 9(c). However, this is contradictory because the roof edges cross each other, which should not happen in the roof structure. Therefore, Algorithm 1 is not robust against numerical

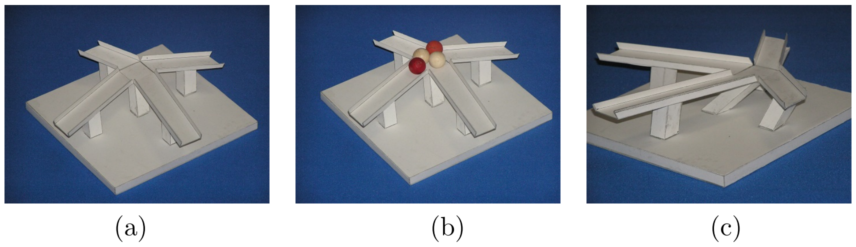
Figure 9: Inconsistency in the construction of a degenerate straight skeleton: (a) regular polygon and growing skeleton edges; (b) detection of the first vertex; (c) detection of the second vertex, which contradicts the first vertex.

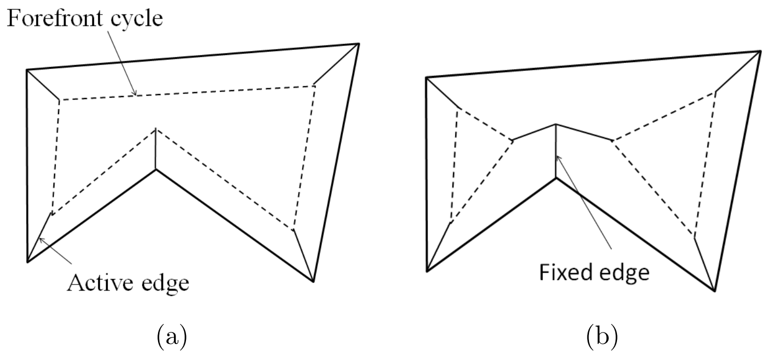

Figure 10: Active edges, forefront cycles, and fixed edges.

errors.

We can modify this algorithm and make it robust by using what we observed in the previous section about how the brain performs such computation. For this purpose, we first classify the roof edges into three types.

Let us concentrate on the structure below the sweep plane π. As shown in Fig. 10(a), the initial roof edges terminate at the points of intersection with π, and the intersection of the roof plates and π form a cycle represented by broken lines in this figure. We call the roof edges that terminate at the points of intersection with π, the *active edges*, meaning that these edges are still growing. We call the cycle formed by the intersection of the roof plates and π a *forefront cycle*, meaning that it is moving toward the inside of the polygon P. Furthermore, we call the roof plates at the forefront cycles the *active roof plates*, meaning that they are still growing.

Initially, all the roof edges and roof plates are active. After the sweep plane π has moved to some extent, some of roof edges below π are completed, as shown by the solid lines in Fig. 10(b). We call these roof edges *fixed edges*. In other words, edges in E are active, while edges in $\overline{E}$ are fixed. Similarly, some of the roof plates become completely bounded by roof edges. We call those roof plates *fixed roof plates*. In other words, roof plates in S are active, while those

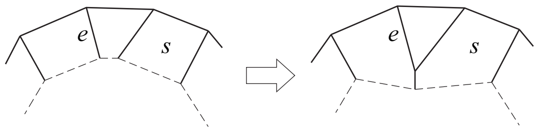

Figure 11: Event type 1: removal of an edge from a forefront cycle.

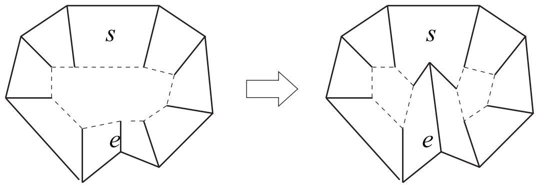

Figure 12: Event type 2: partition of a forefront cycle.

in $\overline{S}$ are fixed.

Once we recognize a forefront cycle, we can identify three types of topological changes in the roof structure that occur during sweeping, as shown in Figs. 11 to 13.

The first type of event is that the sweep plane π hits a point of intersection between a roof edge e and a roof plate s that is on a roof plate adjacent to s, as shown in Fig. 11. In this case, two active edges become fixed, and the forefront cycle becomes shorter by one.

The second type of event is that the sweep plane π has a point of intersection between a roof edge e and a roof plate s that is not on a plate that is adjacent to the side plate of e, as shown in Fig. 12. In this case, the forefront cycle is partitioned into two, the active e edge becomes fixed, and two new active edges are generated.

The third type of event is that a forefront cycle of length 3 disappears, as shown in Fig. 13. In this case three active edges become fixed, three roof plates become completely bounded by fixed edges, and no new edges are generated.

On the basis of these observations, we can make Algorithm 1 robust by

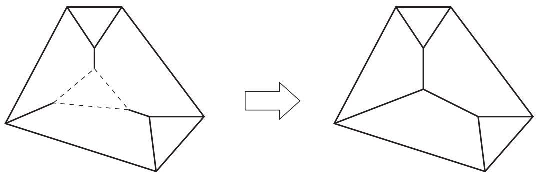

Figure 13: Event type 3: disappearance of a forefront cycle of length 3.

modifying Step 5 in the following way.

Algorithm 2 (Robust construction of straight skeleton).
Step 5 of Algorithm 1 is replaced with the following step, and the remainder is the same as Algorithm 1.

```
5'. Repeat the following until $E$ and $S$ are empty.
    5.1' Find a pair $(e, s)$ of an active roof edge $e$ and a nonneighboring roof
        plate $s$ incident to the same forefront cycle, such that their intersection
        is the lowest.
    5.2' If the event is of type 1, move $e$ and its neighboring edge from $E$ to $\overline{E}$,
        generate a new active edge, and reconnect the forefront cycle, as shown
        in Fig. 11. Move from $S$ to $\overline{S}$ the roof plate that has become fixed.
    5.3' If the event is of type 2, move $e$ from $E$ to $\overline{E}$, generate two new active
        edges, and partition the forefront cycle into two, as shown in Fig. 12.
    5.4' If the event is of type 3, move the three edges (including $e$) from $E$ to $\overline{E}$,
        and change the associated three active edges to fixed edges, as shown in
        Fig. 13. Move from $S$ to $\overline{S}$ the three roof plates that have become fixed.
        □
```

Note that this algorithm does not encounter the topological inconsistency shown in Fig. 9. Indeed, at the initial stage (Fig. 9(a)), a forefront cycle of length 7 is generated, and in the first event shown in Fig. 9(b), which is a type-2 event, the forefront cycle is partitioned into two forefront cycles of length 4. Hence, the situation shown in Fig. 9(c) does not arise because the associated roof edge and the roof plate are incident to different forefront cycles, and hence this pair is not included in the event candidates in Algorithm 2.

In Algorithm 2 , we restrict the event search to each forefront cycle. In this search, we cannot necessarily find the correct event (i.e., the lowest intersection) because of numerical errors. However, whatever pair of a roof edge and a roof plate is chosen as the next event, no topological inconsistency will arise because the chosen pair will be of type 1, 2, or 3, and thus the topological change of the forefront cycle will be well defined.

In other words, the basic structure of Algorithm 2 allows the topological change of the graph structure of the forefront cycles, and numerical computations are used only to choose the most promising pair of a roof edge and a roof plate. Once this pair is chosen, the algorithm persists in the belief that it gives the lowest intersection and changes the forefront cycles by Steps 5.2', 5.3', or 5.4' of Algorithm 2, depending on the type of the event. Thus we obtain the following theorem.

Theorem 1. Algorithm 2 terminates in finite time and produces a planar graph as output, no matter whether the pair (e, s) chosen in Step 5.1' gives the true lowest intersection or not.

Proof. Once a pair (e, s) is chosen in Step 5.1', the associated event is of type 1, type 2, or type 3, and hence the planar graph consisting of the original polygon edges, active edges, fixed edges, forefront cycle edge, and their terminal vertices is changed by Steps 5.2', 5.3', or 5.4', all of which maintain planarity. Thus, it suffices to show the finiteness of the procedure. Assume that the input polygon P has n edges. Initially, the total number of edges on the forefront cycle is n. This number decreases by one in a type 1 event and by three in a type 3 event. In a type 2 event, the total number of edges on the forefront cycles increases by one, but both of the forefront cycles generated by the partition are smaller by at least two than the forefront cycle before the partition. Hence, Step 5.3' is repeated only a finite number of times, and thus the total number of the forefront cycles eventually vanishes. □

Theorem 2. Algorithm 2 runs in $O(n^2 \log n)$ time for an n-gon P.

Proof. Steps 1, 2, and 3 are carried out in time $O(n)$, and Step 4 is carried out in time $O(1)$. Step 5' can be performed in the following manner. Initially there are n active edges and n active roof plates. Hence there are $n(n-2)$ pairs of roof edges and nonneighboring roof plates. We store them in a heap with the y-coordinates of the points of intersection as the keys [1]. We can construct the heap in $O(n^2 \log n)$ time. Deletion of the lowest pair (e, s) from the heap in Step 5.1' requires $O(\log n)$ time. In Steps 5.2' and 5.3', new active edges are generated. As soon as a new active edge e is generated, we compute the point of intersection with each of the nonneighboring roof plates on the same forefront cycle, and add the pair (e, s) to the heap. Adding a pair to the heap requires $O(\log n)$ time. Because there are $O(n)$ roof plates on the same forefront cycle, we can add all the pairs with e to the heap in $O(n \log n)$ time. Hence, Steps 5.2' and 5.3' can be completed in $O(n \log n)$ time. Step 5.4' can be completed in $O(1)$ time. Note that the straight skeleton is a planar graph with n connected regions (corresponding to the n edges of the input polygon) embedded in the plane, and the degree of any vertex is at least three. Hence, the total number of vertices, edges, and connected regions is of $O(n)$. This means that Steps 5.1 to 5.4 are repeated $O(n)$ times. Therefore, Algorithm 2 runs in $O(n^2 \log n)$ time. □

Fig. 14 shows an example of a straight skeleton. The input polygon in this figure has 300 vertices. This polygon was generated by inserting a number of vertices into the edges of a 16-gon and then perturbing their locations with small random numbers. This polygon is not degenerate, and hence the construction of the straight skeleton is not difficult.

Fig. 15 shows the output of our algorithm for a regular 30-gon. This is highly degenerate because, if there are no numerical errors, all 30 of the roof edges will meet at the center. In this experiment, the coordinates of the vertices were represented by single-precision floating-point numbers, and the numerical computations were performed in single-precision floating-point arithmetic. Our algorithm was able to compute the straight skeleton, as shown in Fig. 15(a). However, if we expand the central part by 10^5, we get the diagram in Fig. 15(b),

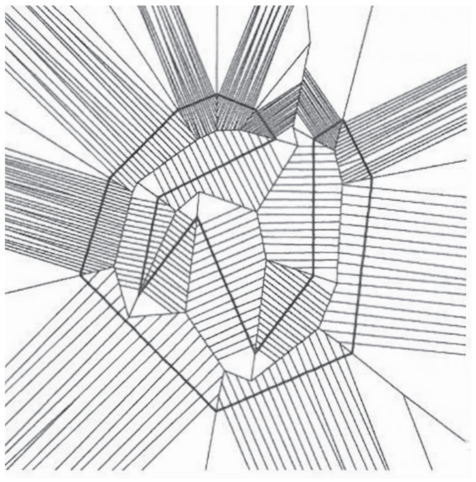
Figure 14: Straight skeleton constructed by Algorithm 2.

where we can see many vertices instead of a single vertex. This kind of disturbance is not surprising; we note that the algorithm gave a topologically consistent output even though the polygon was highly degenerate.

This algorithm is similar to the computations in the human brain in the sense that both are persistent once a decision has been made, regardless of its accuracy. In this way, both are able to achieve robustness against imprecise numerical computations.

The strategy we followed for designing Algorithm 2 can be considered to be a topology-oriented approach, which we have proposed as a basic principle for designing robust geometric algorithms [22, 25]. Indeed, in this approach, the basic part of the algorithm is described in only topological terms, and the numerical data are used only for selecting the most promising branch of the processing. Thus we can say that the idea behind simulating the persistency of the human brain is very similar to the topological approach for robust geometric algorithms.

6 Concluding Remarks

We observed how the human brain processed computations by looking at three visual illusions: the Zöllner illusion, the Ouchi illusion, and the impossible motion illusion, and we then composed a new algorithm for computing straight skeletons. Based on our observations, we pointed out that designing algorithms based on how the human brain computes is very similar to the topology-oriented approach, which we have developed for a long time. Thus, the topology-oriented

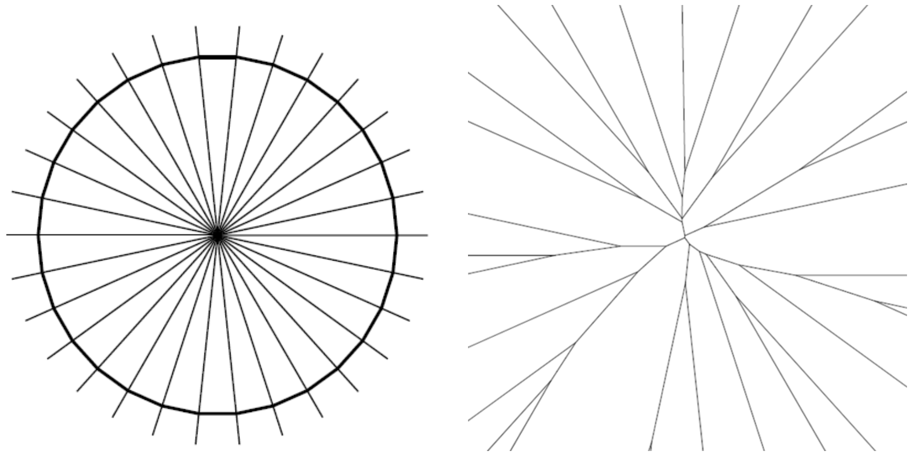

Figure 15: Degenerate straight skeleton: (a) straight skeleton; (b) close-up diagram of the central part.

approach can be used if we want to mimic the processing of the human brain.

Acknowledgment This research is supported by the Grant-in-Aid for Challenging Exploratory Research No. 24650015 and Scientific Research (B) No. 24360039 of MEXT.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullmann: The Design and Analysis of Computer Algorithms. Addison Wesley, Reading, 1974.

[2] O. Aichholzer and F. Aurenhammer: Straight skeletons for general polygonal figures in the plane. In J.-Y. Cai and C. K. Wong (Eds.) COCOON 1996, LNCS 1090, Springer, 1996, pp. 117–126.

[3] D. H. Ballard and C. M. Brown: Computer Vision. Prentice Hall, 1982.

[4] S.-W. Cheng and A. Vigneron: Motorcycle graphs and straight skeletons. Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 156–165.

[5] M. de Berg, O. Cheong, M. van Kreveld and M. Overmars: Computational Geometry: Algorithms and Applications. Springer Verlag, 3rd edition, 2008.

[6] E. D. Demaine, M. L. Demaine and A. Lubiw: Folding and cutting paper. LNCS 1763, Springer-Verlag, 1998, pp. 104–117.

[7] H. Edelsbrunner: Algorithm in Combinatorial Geometry. Springer-Verlag, Berlin, 1987.

[8] C. Fermüller, R. Pless and Y. Aloimonos: The Ouchi illusion as an artifact of biased flow estimation. Vision Research, vol. 40 (2000), pp. 77–96.

[9] C. Fermüller and H. Malm: Uncertainty in visual processes predicts geometric optical illusions. Vision Research, vol. 44 (2004), pp. 727–749.

[10] D. Fogaras and K. Sugihara: Topology-oriented construction of line arrangements. IEICE Transactions of Fundamentals of Electronics, Communications and Computer Sciences, vol. E85-A (2002), pp. 930–937.

[11] T. Goto and H. Tanaka: Handbook of the Science of Illusion. University of Tokyo Press, Tokyo, 2005.

[12] S. Huber and M. Held: A fast straight-skeleton algorithm based on generalized motorcycle graphs. International Journal of Computational Geometry and Applications, vol. 22 (2012), pp. 471–498.

[13] D. Marr: Vision. W. H. Freeman, New York, 1982.

[14] J. Ninio: The Science of Illusions. Cornell University Press, Ithaca, 2001.

[15] T. Minakawa and K. Sugihara: Topology-oriented construction of three-dimensional convex hulls. Optimization Methods & Software, vol. 10 (1998), pp. 357–371.

[16] H. Ouchi: Japanese Optical and Geometrical Art. Dover, Mineola, New York, 1977.

[17] F. Preparata and M. I. Shamos: Computational Geometry—An Introduction. Springer-Verlag, New-York, 1985.

[18] K. Sugihara: Classification of impossible objects. Perception, vol. 11 (1982), pp. 65–74.

[19] K. Sugihara: Machine Interpretation of Line Drawings. The MIT Press, Cambridge, 1986.

[20] K. Sugihara: A robust and consistent algorithm for intersecting convex polyhedra. Computer Graphics Forum, vol. 13, Conference Issue (1994), pp. C-45 – C-54. (EUROGRAPHICS'94, September 12-16, 1994, Oslo, Norway).

[21] K. Sugihara: A characterization of a class of anomalous solids. Interdisciplinary Information Sciences, vol. 11 (2005), pp. 149–156.

[22] K. Sugihara: Robust geometric computation based on the principle of independence. Nonlinear Theory and Its Applications, IEICE, vol. 2 (2011), pp. 32–42.

[23] K. Sugihara: Design of pop-up cards based on weighted straight skeleton. 10th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2013), 2013, pp. 23–28.

[24] K. Sugihara and M. Iri: Two design principles of geometric algorithms in finite-precision arithmetic. Applied Mathematics Letters, vol. 2 (1989), pp. 203–206.

[25] K. Sugihara and M. Iri: A robust topology-oriented incremental algorithm for Voronoi diagrams. International Journal of Computational Geometry and Applications, vol. 4 (1994), pp. 179–228.

[26] K. Sugihara, M. Iri, H. Inagaki and T. Imai: Topology-oriented implementation—An approach to robust geometric algorithms. Algorithmica, vol. 27 (2000), pp. 5–20.

[27] A. Tomoeda and K. Sugihara: Computational creation of a new illusionary solid sign. 9th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2012), 2012, pp. 144–147.