A Speedier Way to Decompose Polygons

A new theoretical result could soon make this key computational operation a whole lot faster

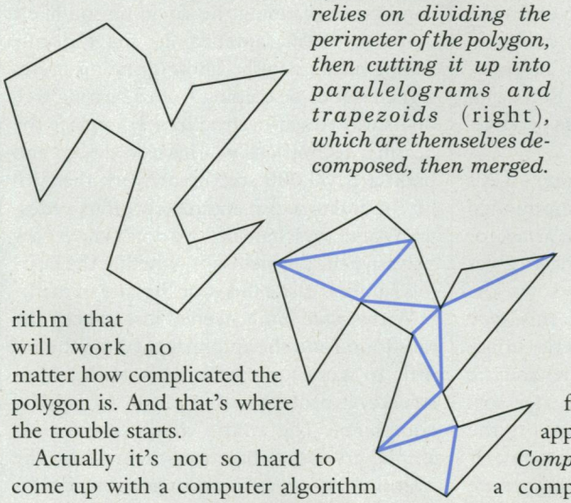
DECOMPOSING POLYGONS. IT SOUNDS LIKE the malodorous aftermath of a biology experiment gone awry. Actually, it's a key procedure in computational geometry. A computer scientist runs a nasty problem in, say, crystal growth—and up pops the world's hairiest snowflake on the computer screen. Unless he can break the snowflake into simpler forms (decompose the polygon), he can't do much more than admire its shape. Unfortunately, decomposing polygons has long been one of computer science's messier jobs.

Enter Bernard Chazelle of Princeton University. Chazelle has broken through a "logarithmic barrier" to show that any polygon can be decomposed in a time that is proportional to its size. If that sounds merely theoretical, to computer scientists it hints at practical methods that could be just around the corner for speeding up the process of decomposing their polygons.

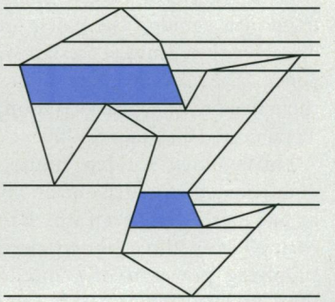
Take one of the commonest applications: "painting" polygons. For example, the computer scientist might want to color his snowflake green. A typical way for a computer graphics program to do this is to cut the polygonal snowflake into triangles and then paint each triangular piece in turn. Another application stems from the many numerical computations that require polygons to be cut into small pieces. One of the most common is the finite element method, a numerical technique for solving differential equations. Here again, decomposing polygons is a logical first step—but it has proved to be a headache.

What turns polygons into computational quagmires is the fact that they have so many vertices. To get a complex polygon under control, computer scientists have long resorted to cutting it into triangles. But not just any old way. You can only cut from one vertex of the polygon to another, and the cuts can't cross each other or go outside the polygon. If you follow the rule, the number of triangles you end up with is two less than the number of vertices, and the number of cuts is three less than the number of vertices. (For example, one cut suffices to turn a square into two triangles.)

Considering that even a child can look at a polygon and show you how to cut it into triangles, you might wonder what's so hard about doing it on a computer. It's a matter of viewpoint: A computer does not have a child's eyes. To the computer, a polygon is just a list of numbers: the x and y coordinates of the polygon's vertices listed in order. So if the polygon has N vertices, the computer must identify N-3 pairs of them, corresponding to the cuts. The job of the computational geometer is to design an algorithm that will work no matter how complicated the polygon is. And that's where the trouble starts.



Triangulation. *A polygon* (left) *is "decomposed" by breaking it down into triangles* (below). *A method devised by Bernard Chazelle relies on dividing the perimeter of the polygon, then cutting it up into parallelograms and trapezoids* (right), *which are themselves decomposed, then merged.*



Actually it's not so hard to come up with a computer algorithm that can do the job; what's difficult is finding an *efficient* algorithm. The "obvious" algorithms tend to require an amount of computation that is proportional to the square of the number of vertices. Roughly speaking, a polygon with 10 vertices requires on the order of 100 computational steps, while a polygon with 1000 vertices requires something like 1,000,000 steps. Computer scientists say that the "complexity" of such algorithms is $O(N^2)$, which stands for "on the order of N^2 computational steps."

By analyzing this challenge carefully, in 1978 Michael Garey and David Johnson at Bell Labs, Franco Preparata at the University of Illinois, and Robert Tarjan at Stanford University were able to find an algorithm whose complexity was $O(N \log N)$—a big improvement over the $O(N^2)$ algorithms. For example, instead of a million steps, a 1000-vertex polygon would, with their algorithm, need on the order of only 3000 steps.

That left computer scientists wondering if they couldn't do still better. In particular, could they find an algorithm with complexity $O(N)$—that is, an algorithm whose computational workload is simply proportional to the number of vertices? Twelve years later, Chazelle has showed conclusively that the answer is yes.

The first glimmer of hope that an $O(N)$ algorithm might be possible came in 1988, when Tarjan and Christopher Van Wyk at Bell Labs found a new algorithm whose complexity was $O(N \log \log N)$. "Their algorithm was quite a big breakthrough," Chazelle says. The extra "log" brought their result to within a hair's breadth of the $O(N)$ goal—but wasn't good enough for those in search of true proportionality.

Chazelle took up the challenge in 1989, and recalls working on it "pretty much nonstop for a year." He first tried modifying Tarjan and Van Wyk's algorithm, but got nowhere. "So I sort of changed my approach completely," he explains. Chazelle's final $O(N)$ algorithm, which is to appear in the journal *Discrete and Computational Geometry*, is based on a complicated "divide-and-conquer" scheme in which the perimeter of the polygon is cut into short pieces. Each piece is analyzed separately, and then merged with one of its neighbors. The merging step is repeated until the entire polygon has been reconstructed, at which point the analysis is refined to yield a full-fledged decomposition.

While Chazelle's algorithm is theoretically better than other methods, it may never see the light of day in actual programming practice. That's mainly because the algorithm is so complicated that it only improves on existing algorithms for polygons that are extremely large. However, the theoretical insights provided by the result are likely to be the basis for future algorithms that will be fast in both theory and practice. Chazelle, for one, believes that's not far off.

"I myself am much more of a theoretician, so I tend to say, 'Oh well, essentially my work is done because I've proven this big theorem,' " Chazelle says. "But actually I also take great joy when something [I've done] turns into something practical." ■ BARRY CIPRA