have is not like a cooking formula, where whatever you throw in you get something out," says Hwang. "This is a method which can help, but you still have to do a lot with respect to a particular problem. We are looking at several problems, but so far we haven't really seen an immediate application."

Polygonal Pursuit

Bernard Chazelle can chop up polygons faster than anyone—provided they're big enough.

Chazelle, a computer scientist at Princeton University, has found an algorithm for triangulating polygons that achieves, in one sense, the ultimate in computational efficiency: The amount of computation is roughly proportional to the number of vertices. In the jargon of theoretical computer science, Chazelle's new algorithm is $O(n)$.

A polygon, of course, is a region in the plane bounded by a simple closed "curve" consisting of line segments. Triangulating a polygon is to cut it up into triangles, all of whose vertices are vertices of the polygon. Thus, a polygon with n vertices will decompose into $n-2$ triangles after $n-3$ cuts. The question is where to make the cuts.

At first glance, it doesn't look as if there's even a problem here. Anyone—even a child—can look at a polygon like the one shown in Figure 2 and immediately see how to cut it up into triangles (especially since the right-hand side of the figure shows one way to do it!).

But look at it from the computer's point of view. The computer doesn't "see" the polygon. All it has to work with is a listing of the vertices. Somehow or other, the computer has to take a list of n points and find $n-3$ pairs whose connecting line segments stay within the polygon and don't intersect one another.

Still sound easy? Then quick, where do you make the cuts for the nonagon with vertices (in clockwise order) at (0,0), (2,0), (0,3), (4,3), (2,2), (5,2), (3,0), (1,1), and (3,–2)? No fair drawing a picture!

Straightforward thinking about the problem almost invariably produces a $O(n^2)$ algorithm. For many purposes this is sufficient. A programmer just wanting to get some bug-free software to market may sacrifice speed for simplicity, especially if the software is only intended for small polygons (where the overhead costs of an "efficient" algorithm may actually make it slower than a simple but asymptotically inefficient approach). And complexity theorists, accustomed to the dizzying heights of NP-completeness, may be excused for writing off the problem as another member of the trivial class P.

But for other purposes, $O(n^2)$ is less than desirable. "As hardware gets more powerful, the temptation is to use big polygons," Chazelle explains. Indeed, polygons with hundreds, thousands, and even millions of sides are common in computer graphics and numerical analysis. Triangulations are important for such purposes as "painting" a picture on a computer screen or preparing a finite element method solution to a differential equation. It can be aggravating when an otherwise fast program bogs down on a simple but plodding subroutine.

Continued on page 17

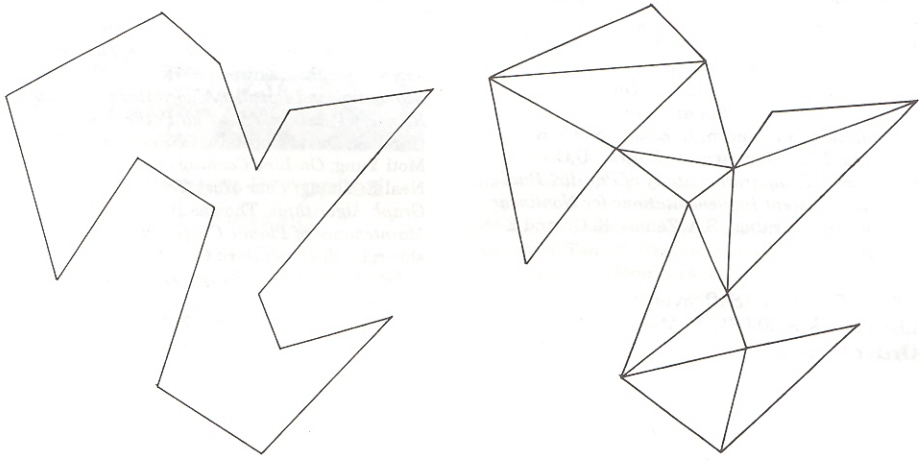

Figure 2. *Left,* a 14-sided polygon. *Right,* a triangulation of the polygon.

Euclidean Geometry,

continued from page 16

There is also the purely theoretical challenge of identifying the exact computational complexity of a given problem. For triangulation, the nature of the output—a list of $n - 3$ pairs of vertices—means that you can't do better than $O(n)$. But how close can you come?

In 1978, Michael Garey and David Johnson of Bell Labs, along with Franco Preparata of the University of Illinois and Robert Tarjan, then at Stanford, came within a whisker. They found a $O(n\log n)$ triangulation algorithm. This was a huge improvement over the naive $O(n^2)$ estimate, at least theoretically. It's hard to get much closer to linear complexity.

There the matter stood for the better part of a decade. Various researchers, including Chazelle, chipped away at the problem, producing algorithms that ran fast—even in linear time—for special classes of polygons, and other algorithms that were $O(n^2)$ only in the worst cases. But $n \log n$ seemed to be the sticking point.

Then in 1986, Tarjan, with Christopher Van Wyk of Bell Labs, managed to sneak an extra "log" into the estimate. Their $O(n\log\log n)$ algorithm came within a whisker's whisker of linear complexity. "It broke the $n \log n$ barrier, and it raised hope that you could do it in linear time," Chazelle says.

Like many others, Chazelle had thought about triangulating polygons from time to time. The Tarjan-Van Wyk result encouraged him to look at the problem again. In 1989, he started working on it full time. The effort paid off in early 1990. "I worked on it pretty much nonstop for a year," Chazelle recalls. He first tried to tease a linear algorithm out of the Tarjan-Van Wyk result, but the $O(n\log\log n)$ estimate refused to yield. Chazelle finally decided that approach wouldn't work. By then, though, he had other ideas.

Six months passed before he felt anything good would come of the time he'd invested. But by November he had something worth publishing: an algorithm with complexity $O(n\log^* n)$. The asterisk essentially means you can insert as many more log's as you want—in effect, Chazelle had shaved the iterated whiskers off a whisker's whiskers. He wrote up the result and planned to call it quits. "I actually stopped for a couple of weeks," he recalls. "But then I had another idea."

The final $O(n)$ algorithm, which took another four months to work out (Chazelle wrote the final paper in April), is based on a complicated divide-and-conquer scheme and a simple lemma that says it suffices to cut the polygon into trapezoids. The trapezoidal decomposition consists of drawing a horizontal line through each vertex of the polygon to the nearest edge on either side (this decomposes the outside of the polygon as well).

In 1984, Chazelle and Janet Incerpi, then a graduate student at Brown University, and, independently, Alain Fournier and Delfin Montuno, then at the University of Toronto, demonstrated a $O(n)$ algorithm for converting trapezoidal decompositions into triangulations. The heart of the new result is a $O(n)$ algorithm for computing the trapezoids.

Chazelle's new algorithm first chops the boundary of the polygon into short segments and finds a trapezoidal decomposition for each piece. It then halves the number of pieces by "merging" pairs of adjacent segments (the first with the second, the third with the fourth, and so on). This merging step is to be repeated until the entire polygon has been reassembled, at which point the trapezoidal decomposition can be converted into triangles. However, this means that each merger must be accompanied by an adjustment of the trapezoids. The trick is not to get bogged down doing this, as a $O(n\log n)$ algorithm could be the result.

Chazelle avoids that fate by simply erasing some of the trapezoidal lines. This speeds up the merging process, but there's a price: What comes out at the end is not exactly a trapezoidal decomposition, but only an approximation to one. The final step, then, is to make the approximation exact, and it turns out that this too can be done in linear time. While the strategy of the algorithm may be simple, the detailed tactics are not. "This algorithm is so hairy, there's no way to describe it," said one observer. Chazelle himself is less awed. "I've spent so much time on it that I think my method is not very complicated," he says. "It's a matter of where you stand."

Whether or not it's easy to understand, Chazelle freely admits his new algorithm is much too complicated to put into practice "unless the polygon is really huge." For that matter, the recent $O(n \log \log n)$ algorithm and even refinements of the original $O(n \log n)$ algorithm are too complicated for practical application. What they have provided, Chazelle says, is insight into the problem. He is hopeful that his most recent insight will give rise to "a very simple algorithm that would be linear or almost linear, but would [also] be very very simple and very practical. . . . I believe that this so-to-speak 'ultimate' triangulation algorithm will come soon, and I don't believe it would have come without understanding the theory behind it."