Reduced data storage, simplified data, and faster plotting of designs on graphics displays—this algorithm has much to offer for IC mask making and CAD applications.

A Polygon-to-Rectangle Conversion Algorithm

Kevin D. Gourley and Douglas M. Green
Texas A&M University

Computer-aided decomposition of complex patterns into some sort of standardized geometric area is essential for many IC mask-making and CAD applications.[1] In these applications, high-level IC design data must be converted to the formats required by conventional pattern generator machines[2-4] before it can be used in IC fabrication. Typically, pattern generator formats describe IC designs as sets of rectangles. Note that the use of disjoint rectangle sets not only speeds up the process of mask generation,[5] but also ensures better edge definition (by avoiding multiple exposures) and permits faster plotting of designs on computer graphics displays. Some systems use sorted geometries, such as rectangles, keeping in memory only those rectangles that reside above a moving lower bound.[6]

Developed as a component of an IC mask CAD/CAM system called SLIMM, for Scanning Laser IC Mask-Making System, the PTR (Polygon-to-Rectangle) algorithm presented here converts orthogonal polygons into disjoint sets of rectangles that are automatically sorted by their lower left corner points. (Orthogonal polygons are composed exclusively of horizontal and vertical rectilinear boundaries.)

Input to the PTR algorithm is an array of unique coordinates describing the corner points (listed in any order) of either a single orthogonal polygon or any number of nonintersecting orthogonal polygons. The algorithm's output consists of a set of disjoint rectangles ordered by increasing lower Y values.

There are a number of reasons for using the PTR algorithm. First, as previously mentioned, the conversion from complex geometries to simpler geometries is required by some IC mask generation equipment.[1-4] But this simplification process can benefit other IC CAD applications as well—by reducing data storage requirements, for example.

But the PTR algorithm not only decreases the *amount* of data used in this decomposition process, it also *simplifies* it. Rectangle data can be quickly and simply processed and plotted in applications requiring geometries that have to be filled or shaded. Once polygons have been converted into rectangles, shading becomes a trivial problem.

Yet another attribute of the PTR algorithm is the automatic ordering of output rectangles by their lower Y values. This ordering is quite useful whenever a sorted list of rectangles is needed or desired.[6]

PTR definition

PTR is an iterative algorithm. Each pass through the algorithm alters or reduces an array of points describing an increasingly simplified polygon and generates one rectangle, which is added to a list of rectangles describing the polygon. PTR continues to be iterated until the array of corner points is empty.

0272-1716/83/0100-0031$01.00 © 1983 IEEE

An accurate description of the PTR algorithm requires the definition of certain terms. First, we define a point P_j that corresponds to a unique coordinate pair (X_j, Y_j). Every P_j represents an intersection of the end points of two perpendicular line segments bounding an orthogonal polygon. An array A_i is defined as the set of n_i corner points, which in turn defines an orthogonal polygon or set of nonintersecting orthogonal polygons:

$$A_i = P_{1_i}, P_{2_i}, \ldots P_{n_i}$$

(The subscript i is used to show that the array differs for each iteration.)

Any polygon can be nonsimple (meaning that it has boundary segments that intersect somewhere other than at end points), but no defined corner point P_j can simultaneously exist on *and* within the boundaries of a single polygon. To illustrate, the polygons in Figure 1 are considered "illegal" in terms of the PTR algorithm, as the circled corner points violate the exclusivity rule given above.

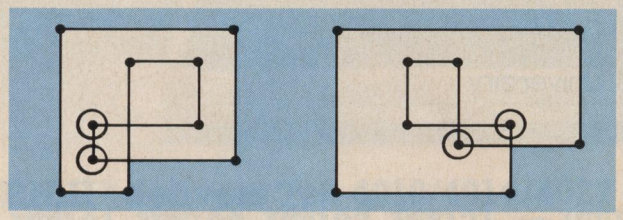


Figure 1. Examples of illegal polygons for the PTR algorithm. The circled points exist both on and within the boundaries of a single polygon.

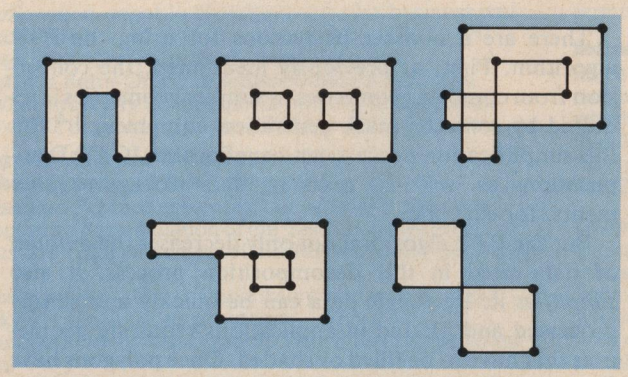


Figure 2. Some legal polygons for the PTR algorithm.

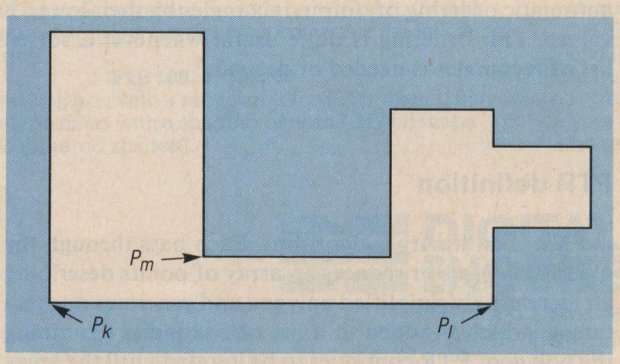


Figure 3. A sample polygon choosing P_{k_i}, P_{l_i}, and P_{m_i}.

The orthogonal polygons shown in Figure 2, however, have no such dual points and are therefore "legal," acceptable geometric shapes for the algorithm.

Certain members of set A_i must also be defined. The points P_k, P_l, and P_m have special significance. (Note that these points change with each iteration.)

P_{k_i} is defined such that

$$X_{k_i} = \min(X^*) \qquad P^* \in A^*$$

where

$$P^* \in A^* \text{ if and only if (iff) } Y^* = \min(Y_j) \quad P_j \in A_i$$

and

$$Y_{k_i} = Y^*$$

P_{k_i} can therefore be described as "the leftmost of the lowest points in array A_i." For example, Figure 3 illustrates P_{k_i} in a sample set of corner points.

P_{l_i} is defined such that

$$X_{l_i} = \min(X^{**}) \quad P^{**} \in \{A^* - [P_{k_i}]\}$$

where P_{k_i} and A^* were previously defined, and

$$Y_{l_i} = Y_{k_i} = Y^*$$

P_{l_i}, then, can be described as "the next leftmost of the lowest points in A_i," as shown in Figure 3.

P_{m_i} is defined such that

$$X_{m_i} = \min(X^\circ) \quad P^\circ \in A$$

where $P^\circ \in A^\circ$ iff the following conditions are true:

$$X_{k_i} \le X^\circ < X_{l_i}$$
$$Y^\circ > Y_{k_i}$$
$$P^\circ \in A_i$$

Looking to Figure 3 again, we see that P_{m_i} can be described as "the leftmost of the lowest points existing in A_i with Y values greater than Y_{k_i} and with X coordinates in the range between X_{k_i} and X_{l_i}."

It is also necessary to define a function F that operates on set A_i as follows:

$$F(A_i, X', Y') = A_i - [P'] \text{ iff } P' \in A_i, \text{ otherwise;}$$
$$F(A_i, X', Y') = A_i + [P']$$

where

$$P' = (X', Y')$$

Finally, rectangles are described as

$$R_i(X_{LL_i}, Y_{LL_i}, W_i, H_i)$$

where X_{LL}, Y_{LL} is the lower left corner point of a rectangle, W is the width, and H is the height.

Now the PTR algorithm can be completely defined. The following steps are repeated ($i = 1,2, \ldots z$) until $n_i \rightarrow n_z = 0$, that is, A_z = [empty set]:

(1) Determine P_{k_i}, P_{l_i}, and P_{m_i} from set A_i.

(2) $W_i = X_{l_i} - X_{k_i}$
$H_i = Y_{m_i} - Y_{k_i}$
$X_{LL_i} = X_{k_i}$
$Y_{LL_i} = Y_{k_i}$

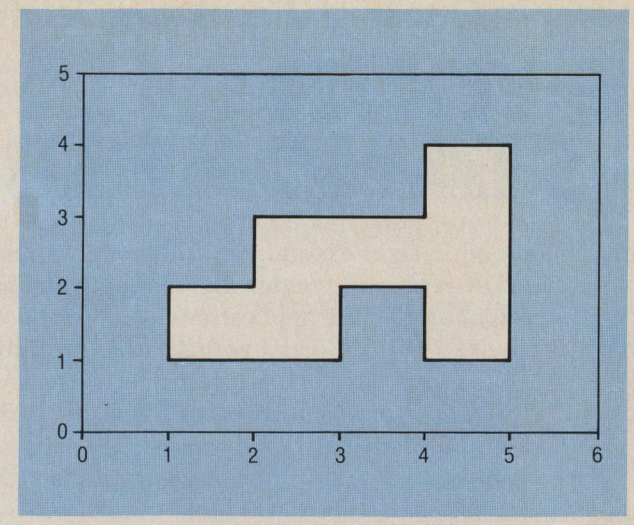


Figure 4. PTR iteration example.

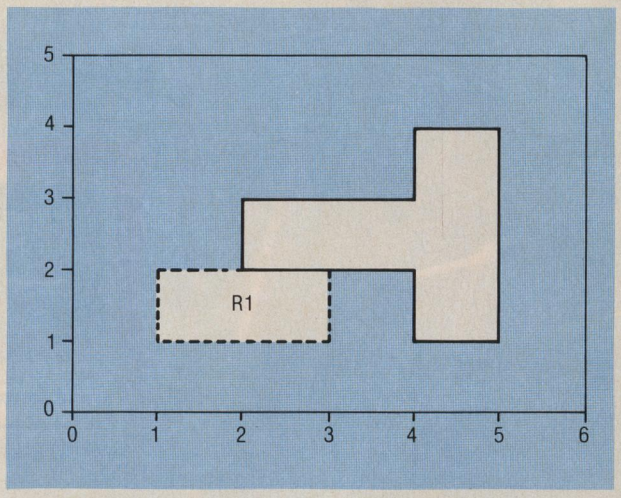


Figure 5. Example polygon after one iteration.

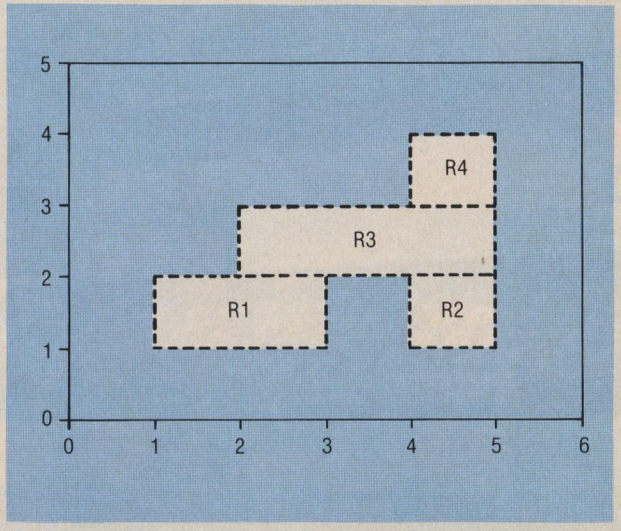


Figure 6. The example polygon reduced to rectangles at the end of the PTR algorithm.

$$
\begin{aligned}
(3)\ A_i' &= F(A_i, X_{k_i}, Y_{k_i}) = A_i - P_{k_i} \\
A_i'' &= F(A_i', X_{l_i}, Y_{l_i}) = A_i' - P_{l_i} \\
A_i''' &= F(A_i'', X_{k_i}, Y_{m_i}) \\
A_i'''' &= F(A_i''', X_{l_i}, Y_{m_i}) \\
A_{i+1} &= A_i''''
\end{aligned}
$$

At the time of the zth iteration, $i=z$ and $n_z=0$. Then a set of rectangles will exist, namely $R_1, R_2, \ldots R_{z-1}$, which equivalently describes the polygon(s) defined by the corner points in A_1.

Polygon-to-rectangle conversion

The following example graphically illustrates the terms defined in the algorithm above.

First, Figure 4 shows an orthogonal polygon defined by these terms:

$$
\begin{aligned}
&i=1 \\
&n_i=12 \\
&P_k=(1,1) \\
&P_l=(3,1) \\
&P_m=(1,2) \\
&W_1=3-1=2 \\
&H_1=2-1=1 \\
&X_{LL_1}=1 \\
&Y_{LL_1}=1 \\
&R_1=(1,1,2,1)
\end{aligned}
$$

The function F will remove points (1,1), (3,1), (1,2), and (3,2) from A_1. The results of this action are shown in Figure 5.

At the beginning of the second iteration, $n_2=8$. After four iterations, the polygon has been reduced to four rectangles. At the start of the fifth iteration, we see that $n_5=0$, thus permitting termination of the algorithm. At the time of termination, the polygon in Figure 4 has been converted into the set of rectangles illustrated in Figure 6.

PTR performance observations

Note that the array of points $P_1, P_2, \ldots P_{n_i}$ need not be listed in any particular order. The only requirements placed on the coordinates are that each point P_j must be unique in A_i and that no point may exist simultaneously within and on the boundaries of a single polygon.

One of the advantages of the PTR algorithm is that it is not limited to operations on single polygons. This algorithm can also be used to reduce a set of nonintersecting orthogonal polygons that satisfy all of the previously mentioned restrictions.

Remembering that the corner points for all polygons being reduced to rectangles within any single iteration can be listed in random order, there is no need to group the points according to the polygon that they belong to. This fact can be observed in the definition of the function F, which includes and deletes points from A_i with no requirements pertaining to the order of the points in the array. The implementation of function F can be quite simple because of this lack of ordering requirements.

The method by which the elements of A_i are managed can affect the speed performance of the PTR algorithm, but not its results. Whether a sorted, linked list of the elements of A_i is kept, or the elements are sorted prior to each iteration, or no sorting at all is performed, the PTR algorithm can be implemented. In other words, the definition of the PTR algorithm, as presented here, is independent of any particular method devised for managing the elements of A_i.

Although the PTR algorithm can be used to reduce a number of polygons into rectangles simultaneously, it functions optimally by reducing the polygons sequentially. By the term "sequentially," we mean that polygon p should be operated on by PTR until $n_i(p)$ goes to zero. Then, polygon $p+1$ should be reduced until $n_i(p+1)$ goes to zero, etc. In the actual implementation of the PTR algorithm, this reduces the required physical size of array A for all applications.

When all requirements are met concerning the list of corner points and their related orthogonal nonintersecting polygons, PTR converges (n_i goes to zero), producing a nonempty set of disjoint rectangles, $R_j (j=1,2, \ldots z-1)$, with nonzero width (W_i) and nonzero height (H_i). This convergence process can be seen more clearly by examining the function $F(A,X',Y')$, and the definitions of P_{k_i}, P_{l_i}, P_{m_i} and their usage in the third step of the PTR algorithm. First, examine the statements

$$A_i' = F(A_i, X_{k_i}, Y_{k_i})$$
$$A_i'' = F(A'_i, X_{l_i}, Y_{l_i})$$

These translate to

$$A_i' = A_i - [P_{k_i}]$$
$$A_i'' = A_i' - [P_{l_i}] = A_i - [P_{k_i}, P_{l_i}]$$

P_{k_i} and P_{l_i} are always deleted from A_i with each iteration. Since $n_i'' = n_i - 2$, this indicates that if an increase of n_i is to occur, it is going to be as a result of at least one of the last two relations established in step three of the PTR algorithm (if n_i is to increase at all):

$$A_i''' = F(A_i'', X_{k_i}, Y_{m_i})$$
$$A_i'''' = F(A_i''', X_{l_i}, Y_{m_i})$$

By the definition of the function F, only two points, at most, can be added to set A_i'' by these last two relations. This condition occurs only if points $(X_{k_i}, Y_{m_i}) \in A_i''$ and $(X_{l_i}, Y_{m_i}) \in A_i'''$. Therefore,

$$n_i - 4 \leq n_i'''' \leq n_i$$

Thus, n_i is never increased after an iteration. The only matter of concern that now remains is the case where $n_i'''' = n_i$. If this condition were to occur repeatedly, n_i would never converge to zero.

In the case $n_i'''' = n_i$, two more coordinates, $(X_{k_i}, Y_{m_i}), (X_{l_i}, Y_{m_i})$, are introduced and P_{k_i} and P_{l_i} are deleted. By definition, $Y_{m_i} > Y_{k_i}$ and $Y_{m_i} > Y_{l_i}$. The points (X_{k_i}, Y_{m_i}) and (X_{l_i}, Y_{m_i}) are somewhere above P_{k_i} and P_{l_i} on the line $Y = Y_{m_i}$. Since PTR operates on polygons of finite size, Y_{m_i} is necessarily finite. After a certain number of iterations, PTR will have deleted all points below $Y = Y_{m_i}$, and function F will have deleted two points on the lowest Y=constant line containing points in A_i until all points on that line have been removed. The highest Y=constant line containing points in A_i will eventually become the only line left containing points in A_i, in which case F will remove these points. Therefore, it takes a finite number of PTR iterations to reduce a finite-size orthogonal polygon into a set of rectangles.

At this point it should be noted that each PTR iteration removes at least one of the original A_1 horizontal line segments, and that the final iteration will actually remove two horizontal line segments. Since $n_1/2$ horizontal line segments exist in A_1, the maximum number of iterations needed to resolve the polygon into rectangles is $(n_1/2) - 1$.

PTR test performance

The PTR algorithm has been implemented in Fortran-77 and has been tested on a large variety of cases. It was found to be both simple to implement and efficient. When input data requirements were met, PTR was successful in every test case. Figure 7 shows a plot comparing the number of corner points of a polygon to the number of rectangles generated by PTR.

The dashed line in the figure indicates the values where the space required for polygons stored by corner points equals the storage required to describe the same polygons with a set of rectangles; the solid line is the lower bound for PTR (no more than four coordinates can be deleted with each iteration); and average PTR performance is indicated by the dotted line. The results shown in Figure 7 are based on data from an actual integrated circuit designed at Texas A&M University. The data included 17,684 coordinates from 1880 polygons. These polygons were converted into 6466 rectangles. Required data storage was reduced by approximately 25 percent over straight polygon format data. In no test case did the use of PTR result in an increase in data storage requirements.

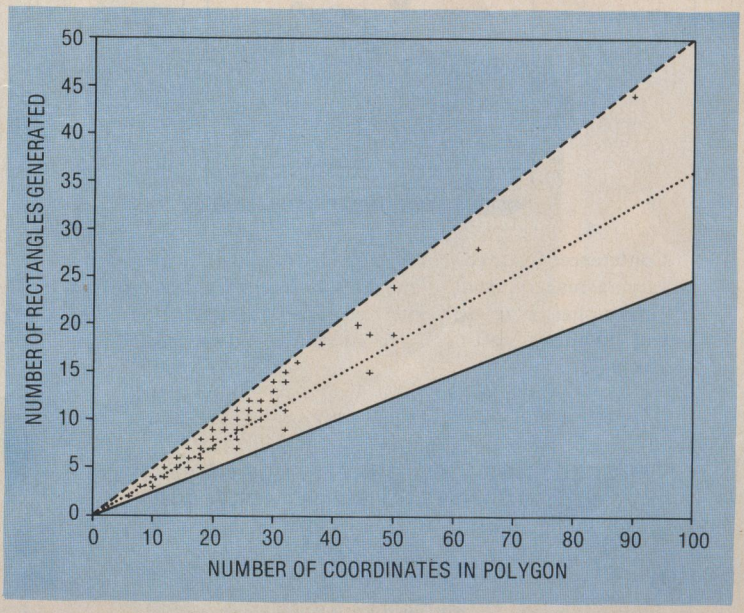


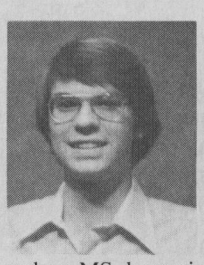
Figure 7. PTR test performance results.

Applications of PTR have been found in IC pattern generation and computer-aided design, where quickly generated sorted lists of disjoint rectangles are desirable,[7-9] if not required. Optical IC pattern generators produce polygons by exposing a series of rectangles. Further, some computer graphics displays support hardware shading of rectangles. Thus PTR provides a simple and convenient method for converting polygonal definitions into the required rectangular format. ■

References

1. K. Patel, "Computer-Aided Decomposition of Geometric Contours into Standardized Areas," *Computer-Aided Design,* Vol. 9, No. 3, July 1977, p. 199.
2. Stephen Trimberger, J. A. Rowson, C. R. Lang, and J. P. Gray, "A Structured Design Methodology and Associated Software Tools," *IEEE Trans. Circuits and Systems,* Vol. CAS-28, No. 7, July 1981, p. 618.
3. Robert W. Hon and Carlo H. Séquin, *A Guide to LSI Implementation,* second edition, Xerox Palo Alto Research Center, Palo Alto, Calif., Jan. 1980.
4. C. Mead and L. Conway, *Introduction to VLSI Systems,* Addison-Wesley, Reading, Mass., 1980.
5. V. Jayakumar and A. Chatterjee, "Description of Rectilinear Areas in Terms of a Minimum Number of Disjoint Rectangles," *Proc. IEEE Int'l Conf. Circuits and Computers,* Part II, Oct. 1980, pp. 777-779.
6. H. S. Baird, "Fast Algorithms For LSI Artwork Analysis," *Proc. 14th Design Automation Conf.,* Vol. 2, No. 2, May 1978, p. 303.
7. G. H. Henriksen, "Reticles by Automatic Pattern Generation," *SPIE Semiconductor Microlithography II,* Vol. 100, 1977, pp. 86-95.
8. W. D. Grobman, "An Overview of Pattern Data Preparation for Vector Scan Electron Beam Lithography," *Proc. IEEE Int'l Conf. Circuits and Computers,* Oct. 1980, pp. 558-561.
9. J. A. Wilmore, "The Design of an Efficient Data Base to Support an Interactive LSI Layout System," *Proc. 16th Design Automation Conf.,* June 1979, p. 445.

Use order form on p. 91.



Kevin D. Gourley is a systems engineer at Electro Scientific Industries, Portland, Oregon. He is responsible for CAD/CAM systems development and support utilizing ESI laser trimming systems. Since 1980, he has done research and development work with Texas A&M University in the area of computer-controlled systems and CAD systems.

Gourley received a BS degree in 1980 and an MS degree in 1982 in electrical engineering from Texas A&M University. During his graduate research work, he developed the SLIMM (Scanning Laser IC Mask-Making) system discussed in this article. His specific interests include scientific computing applications, CADAM systems development, and computer graphics. He is a member of the IEEE and Eta Kappa Nu.

Douglas M. Green is an associate professor of electrical engineering at Texas A&M University. Previously, he was an assistant professor at the University of Texas at Austin. His research interests include computer graphics, VLSI CAD, and video densitometry. Green received his BSEE and MSEE degrees from Texas Tech University and his PhD from the University of Texas at Austin. He is a member of ACM Siggraph and the IEEE Computer Society.