CORNER-STITCHED TILES WITH CURVED BOUNDARIES

Carlo H. Séquin

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

ABSTRACT

A generalization of the classical corner-stitched data structure for integrated circuit layouts is presented permitting the description of circles and of arbitrary curved shapes. In principle this extended data structure can be built with just the additional space required to store the more complicated curved boundaries. The tradeoffs between different encoding schemes that minimize overall data storage size or the complexity of individual tiles are discussed. The topology of the linkage of tiles by the corner-stitching pointers is equivalent to that of simpler patterns with trapezoidal tiles, but the various tests and operations running on this data structure may become considerably more complicated, and the achievable run-times will be highly implementation dependent.

1. Introduction

The corner stitching data structure introduced by Ousterhout[1] has proven to be a very useful technique for describing 2D planar geometries such as they occur in IC layouts or in floor plans. For many applications the use of rectangular tiles is entirely adequate, and the technique has been used widely in this form.

More recently Marple et al. have extended the application domain to layouts with edges at angles that are multiples of 45 degrees. They have implemented a practical system for IC layout in which these geometries are decomposed into trapezoidal tiles.[2] They also point out that their technique can be extended to trapezoids with arbitrary angles. This would make it possible to describe arbitrary polygonal shapes.

Recently the question has been raised whether this technique can be extended to planar shapes with arbitrary curved geometries which arise in some micro-electromechanical assemblies (Fig. 1) or in opto-electronical devices. This communication builds on the work by Ousterhout and Marple and explores whether such a generalization can be achieved with a reasonable increase in storage requirements and operational complexity.

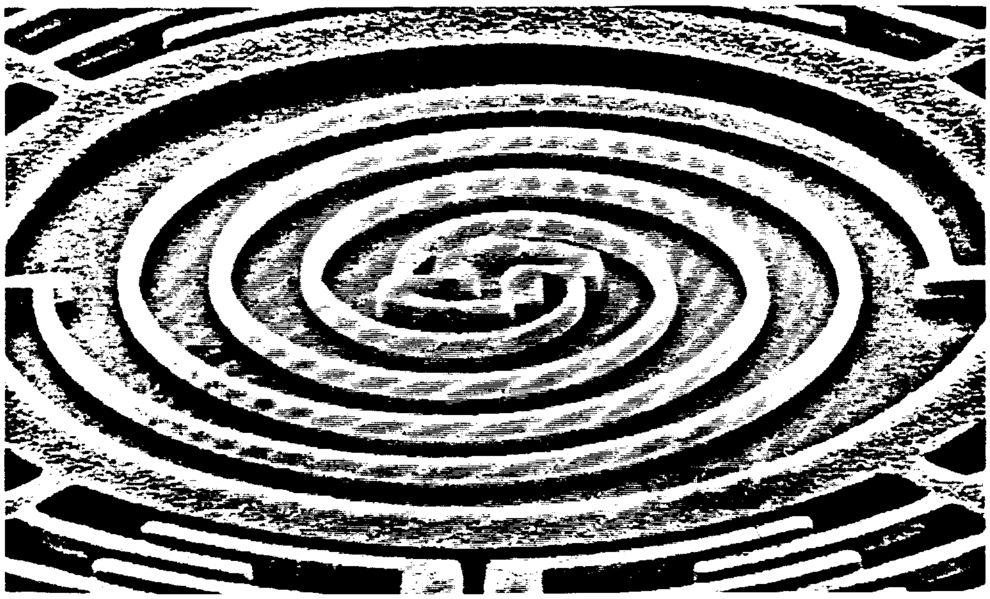


Figure 1: *A device whose layout requires smooth curved boundaries (courtesy, Richard Muller, UCB).*

It turns out that at a conceptual level, an extension to tiles with curved boundaries is fairly straight-forward, but difficulties will likely be encountered in the implementation of an efficient system for a specific application domain. Interesting tradeoffs appear in the compact representation of a given geometrical layout pattern; the number of tiles can be significantly reduced at the price of using more complicated functional compositions to describe their boundary curves.

2. Maximally Horizontal Tiling

The key idea, as in all previous versions of corner stitching, is to break all shapes as well as the intervening spaces into maximally horizontal tiles that all have the same basic properties: First, they have horizontal top and bottom boundaries; these boundaries may degenerate to a single point. Second, they have left-side and right-side boundaries that can be specified as position-continuous (G^0), single-valued functions $x(y)$. These generalized 'trapezoidal' tiles are stitched together in the same basic way as described by Ousterhout and Marple.[1, 2] Such maximally horizontal tiling (Fig. 2) prevents fracturing of the shape description into an uncontrollable number of small tiles, and it represents at the same time a canonical description that depends only on the given geometry and not on the history of its creation.

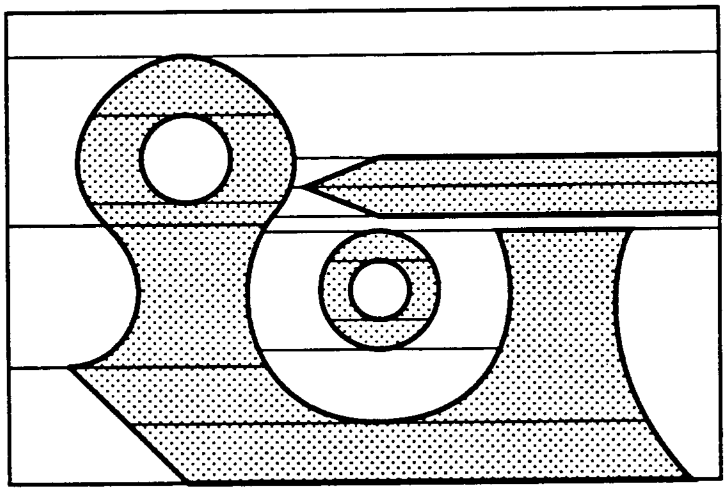


Figure 2: *An arrangement of generalized maximally horizontal trapezoidal tiles.*

Marple's Tailor system[2] makes an explicit distinction between nine different trapezoidal (or triangular) shapes based on the slopes of the left and right edges. These shape types thus implicitly encode the equations of the left- and right-side tile boundaries. For our generalized tiles with curved sides, we need to store such boundary information more explicitly. We also permit that either the top or bottom edge, or both, degenerate to a single point; in these cases, the left- and right-side boundaries would share one or two points. Each tile contains the open segments constituting its left-side boundary and its bottom edge as well as, typically, its lower left corner. With the additional rule that when several tiles have the same lower left corner, the corner point belongs to the right-most tile (which will have a non-zero length bottom segment), every point in the plane belongs to exactly one tile. Figure 3 shows the spectrum of increasingly more complex tiles ranging from the original rectangular corner-stitching data structure[1] (a), through the trapezoidal tiles with arbitrary angles suggested by Marple[2] (b), to our new tiles with curved boundaries. We show two types of tiles; one has simple parameterized cubic Bézier curves as its left-

and right-side boundaries (c), while the other has composite paths composed of linear, circular, and spline segments (d). More will be said below about the tradeoffs between these two types of tiles. Also shown in Figure 3 is the minimal information that needs to be stored in each tile record.

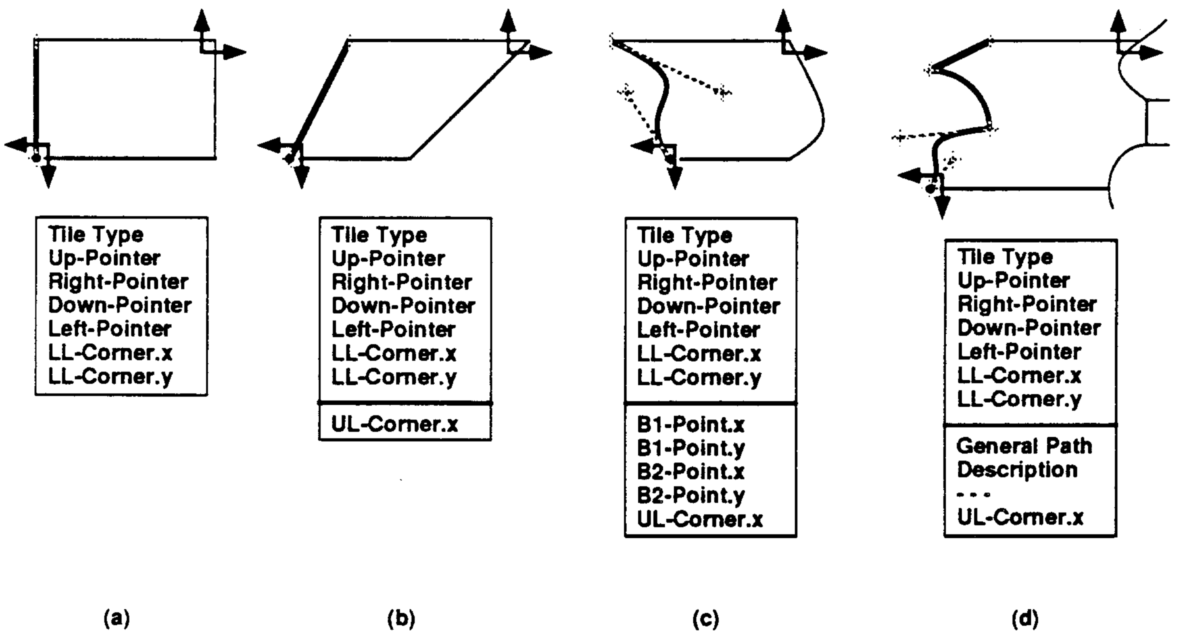


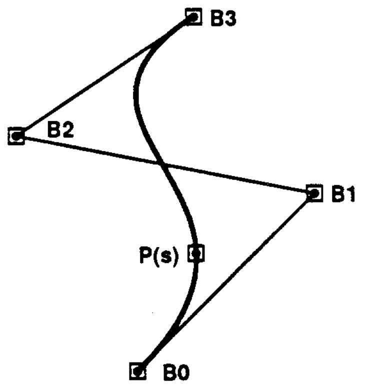
Figure 3: *Generalizations of the original, rectangular, corner-stitched tile (a) to trapezoidal tiles (b) and to tiles with curved boundaries (c) and (d).*

A key issue is to find a storage-efficient representation, so that large designs can be stored compactly. Thus implementors of corner-stitching data structures have always tried to avoid storing redundant information, even at the expense of using more complicated data operations to obtain some information about the geometry of a tile indirectly from data stored in neighboring tiles. We continue this practice and push it to its limits; for curved tiles the possible pay-off in storage density becomes even more significant. In the case of rectangular tiles, the right-side boundary is easily implied from the corner coordinates of the tiles adjacent to the right. The description of a curved boundary is more expensive to store; thus avoiding redundant specifications amounts in more significant savings.

A further advantage of this storage parsimony is a more consistent representation. Rectangular tiles can be made to abut easily. When tiles with curved boundaries are packed against one another, the curves must match exactly, or there will be undefined spaces or overlaps that would lead to ambiguous or contradictory results for elementary operation such as point-finding. We thus make sure that the tiles to the right and to the left of a curved boundary share one and the same boundary description. This boundary is specified in the tile to the right and is considered to be a part of this tile except for its end points. The left edge of each tile is thus described explicitly as part of that tile, while the right boundary is inferred by the left edges of all the adjacent tiles immediately to the right.

3. Minimizing the Number of Tiles

Another important consideration is the number of tiles into which the layout is subdivided. Obviously, the fewer tiles, the better. Fewer tiles not only require less storage, they also speed up the actual operations carried out on the data structure, such as point location. The main trade-off is the complexity of the boundary curve that one is willing to handle in a single tile: fewer tiles with more complex boundaries, versus more tiles with simpler boundary curves.



$$P(s) = \sum_{i=0}^{3} Bi\, Wi(s)$$

$$W0(s) = (1-s)(1-s)(1-s)$$

$$W1(s) = 3(1-s)(1-s)s$$

$$W2(s) = 3(1-s)s^2$$

$$W3(s) = s^3$$

$$s : [\,0, 1\,]$$

Figure 4: *A cubic Bézier curve and its polynomial representation.*

Let's assume for the moment that the boundary curves are represented as Bézier segments. An n-th degree Bézier segment can be described conveniently by its two end points and n-1 intermediate control points.[3] For instance, some 'S'-shapes with a single inflection point can be approximated with a single cubic Bézier segment (Fig. 4). The actual boundary curve is described with a parametric function of four control vertices **B1...B4**, whose coordinates are blended together by the four weighting functions **W0**(s)...**W3**(s), as the parameter s runs from 0 to 1.

If the same boundary curve must be approximated with Bézier curves of only second degree, it needs to be split into at least two segments (at the inflection point), and the two segments together require a total of five or more control points. If we further restrict ourselves to tiles with just a single Bézier segment as their left-side boundary, then the above split will also require at least one additional tile. This tile will then carry with it all its associated storage overhead of tile-type descriptor and corner-stitching pointers. Tracing a vertical search path through this region will require one extra tile traversal.

Thus, in order to minimize the number of tiles, it seems advantageous to permit the boundary curves to be of higher degree. Another way to increase the expressiveness and adaptability of boundary curves is to use *rational*, rather than polynomial, spline formulations. Good results have been obtained with quadratic rational Bézier splines for the task of describing smooth outline fonts.[4] This representation is also recommended if circles and ellipses need to be represented exactly.

But one can go even further in reducing the number of tiles, by permitting the left-side boundary of a tile to be a composite of several segments from a predefined collection of curve types. It is possible to use any G^0-continuous path $x(y)$ that can be composed of, say, straight line segments, circular arcs, or polynomial or rational splines defined by a set of control vertices (Fig. 3d). Of course, a more sophisticated procedure, *'left of boundary'*, is needed now to determine the position of a point with respect to this more complicated boundary. This general function must properly for all possible boundaries that can be composed from the admissible curve segment types.

From a functional point of view, it also makes sense to keep such a composite boundary together as one entity. Such boundaries typically describe one edge of a complicated part or geometrical shape that may have been produced by some special module generator or by a separate sub-design process. Whether such a shape describes a micro-mechanical gear wheel or an input/output device on an electro-optical chip, it is likely to be moved around as a whole; keeping its boundary together in fewer, larger, but possibly more complicated pieces, makes such manipulation simpler. Furthermore, for reasons of numerical consistency to be discussed in Section 5, it also is preferable to keep a single overall description of a curved boundary rather than braking it up into several separate shorter segments.

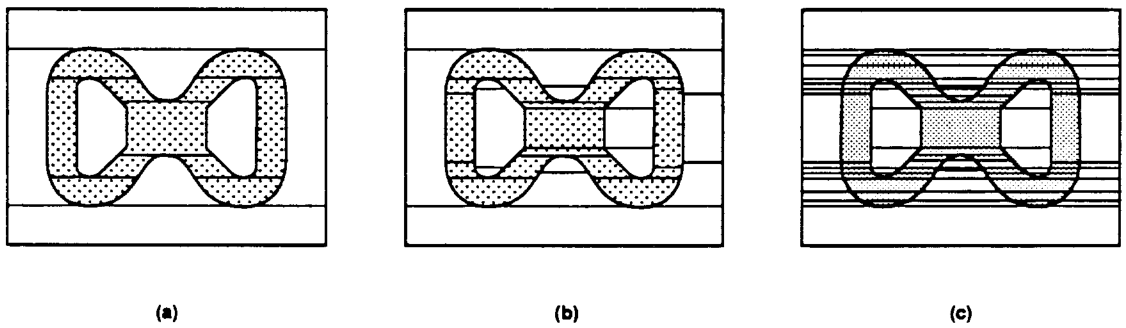


Figure 5: *Tiling of a given shape using composite tile boundaries (a), tiling of the same shape with simple quadratic Bézier boundaries (b), and tiling approximation with simple trapezoidal tiles (c).*

Figure 5 demonstrates the dramatic reduction in the number of tiles that can be achieved for the description of the given example shape if tiles with composite boundaries are introduced. Figure 5 compares three ways of tiling the same shape. The most efficient one is achieved with composite boundaries comprising linear, circular, or spline segments; only 11 material tiles and 8 space tiles are generated. When we restrict ourselves to simpler tiles with single quadratic Bézier segments on their left sides, then a total of 33 tiles are needed. Finally, for comparison, we show a piece-wise linear approximation with 99 simple trapezoidal tiles.

4. Tiling Algorithm

We will now discuss where boundaries need to be broken, given some geometrical pattern, and how the individual tiles are constructed. We will discuss the case of tiles with simple quadratic left-side boundaries as well as the case of more complicated composite boundary curves. In either case we assume that the right side of the tile can be defined by multiple curve segments belonging to different tiles. These are the major steps:

(1) For every horizontal boundary segment, replace it with a tile cut, extending it left and right until it hits two other boundaries.

(2) At every vertical extremum on a shape (max and min y), break the boundary and add a tile cut, extending it left and right until it hits two other boundaries.

(3) The remaining boundary pieces are now all single-valued, continuous, but not necessarily smooth functions $x(y)$. Further partitioning depends on the types of boundary curves that we are willing to permit in each tile.

(3a) If only single quadratic segments are allowed, then the boundary pieces need to be split further at all slope discontinuities and inflection points. Some of the remaining longer and more complicated curve segments may have to split further in order to get a good enough approximation to the desired shape. From all these split points, extend tile cuts only to the right until they hit another boundary.

(3b) If composite boundary curves are permissible, then fewer splits are required. In the limit, any part of a boundary that forms a single-valued, continuous functions $x(y)$ that can be expressed as a single path, can be used as the left-side boundary of a single tile.

5. Curve Sharing

A corner-stitched representation with rectangular tiles or with trapezoidal tiles restricted to 45 degree angles, naturally invites an integer representation. With some care it can be guaranteed, that even when such tiles need to be subdivided, all coordinates that need to be stored remain integer. When more complicated tile boundaries are introduced, this is harder to achieve.

Of course, slanted straight edges can be restricted to having endpoints with integer coordinates, circles can be restricted to having centers on integer positions and integer radii, and splines could be required to have only integer control vertex coordinates. However, even when one cuts a straight line of arbitrary slope at an integer y-coordinate, the resulting x-coordinates may no longer be of integer value. Similarly, when one subdivides a Bézier spline with integer-valued control vertices, the control vertices of the resulting segments cannot be expected to have integer coordinates. In either case, we would be forced to represent some numbers in the tile description record with non-integer values in order to maintain consistency in the representation of the various segments of a subdivided boundary curve. First, this will result in higher computational costs, and possibly also in higher storage costs. Even if we are willing to pay this price, there may be problems of fragmentation. Rounding errors and other numerical inaccuracies, may prevent us from recognizing that two separate parts of a curve can be merged again into the single description from which they originated. This would produce a "composite" boundary description with two separate segments with different descriptions, and might eventually fracture a smooth curve into many small segments that "almost" obey the same equation.

To avoid this problem, it will be better to keep the complete boundary curve in a single description which can be shared by several tiles. The tiles then need only store the bottom y-coordinates and a pointer to the boundary curve $x(y)$; the corresponding x values at the relevant corners are then calculated by evaluating the referenced boundary curve function. With this approach, it is easy to see when two tiles that share the same bottom/top segment can be merged — they must refer to the same boundary curve description.

6. Corner Stitching

Ousterhout's Magic layout system,[1] demonstrating the original rectangular corner stitching algorithm, used a slightly different linkage of its tile-connecting pointers than Tailor[2] for cases where tiles meet corner-to-corner. Marple et al. had to rethink the issues surrounding such special tile constellations in the context of triangular tiles with zero-length top or bottom boundary segments. In Tailor the down-pointers always connect to the left-most tile touching the shared vertex from below. Similarly, the up-pointers always connect to the right-most tile touching the shared vertex from above (Fig. 6a).

With this definition, all the necessary linkages are available to walk around any kind of tile. The patterns generated by curved boundaries do not produce significantly new topologies. The only new feature is a two-edged tile (Fig. 6b), but its two vertices can readily be handled like the corners of triangular tiles in Tailor.

Because curved tiles can be arranged and connected with the same basic topology as the trapezoidal tiles in Tailor,[2] the same algorithms can be used for the various operations discussed by

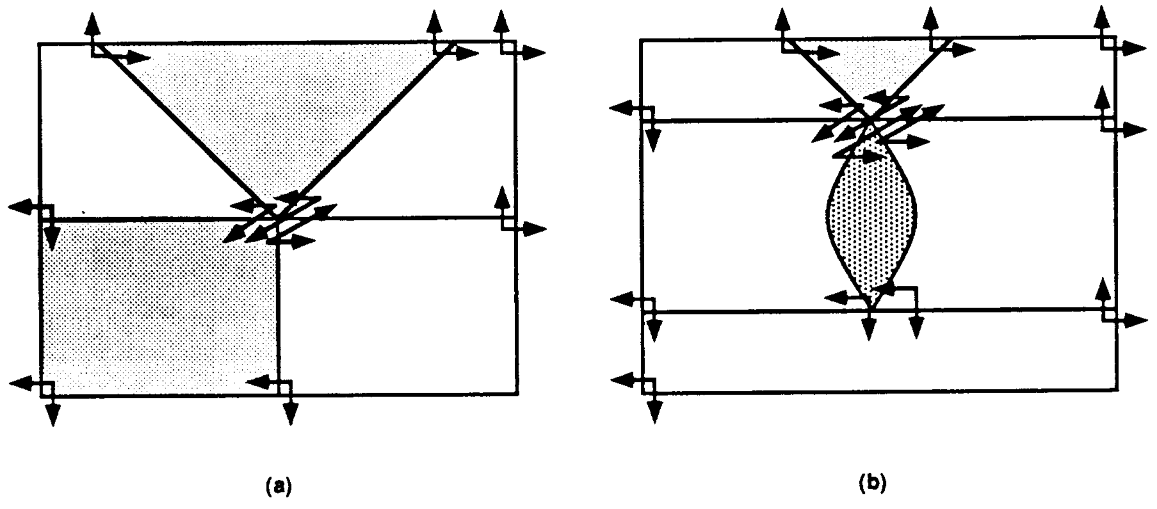


Figure 6: *Corner stitching at vertices touched by more than three tiles (a), including the case of a tile with only two edges (b).*

Ousterhout and Marple: *point search, neighbor search, area search, shadow search, and node search.* Of course, the crucial function that tests whether a point is *'left of boundary'* with respect to a certain tile may be quite complicated now, depending on the type of boundaries allowed. For Bézier curves, a first crude test could be based on the bounding box or on the convex hull given by the control vertices.[3] Circular arcs and other conic segments make the *'left of boundary'* test simple if they are represented in implicit form: The coordinates of the query point are simply inserted into the curve equation.

Insertion and deletion of material tiles also work basically in the same way as in previous corner stitched layout systems but may require considerably more computation. The key operations here are the vertical splitting and merging of tiles in order to maintain a canonical representation. If tiles with composite boundaries are allowed, then two vertically adjacent tiles can always be merged whenever they share the same bottom and top segment, respectively. If the two tiles refer to two separate left-boundary descriptions, then these two boundary sections may be merged into a single, longer composite path description. If the left-side tile boundaries are restricted to individual curves, say, quadratic Bézier splines, then the two tiles should only be merged if they reference the same boundary curve description.

7. Towards a Generic Corner-Stitched System

Because the differences introduced by various tile types all have an impact only on low-level routines, but not on the overall organization of the data structure or on the strategies for the various searching and modification operations, a layout-system based on corner-stitching could be designed in an extensible manner that readily adjusts to different tile types and boundary curves. Such a system would be based on a simple corner-stitched tile that contains minimal information: the tile type (*color, material*), the lower left corner coordinates (x, y), four pointers to neighboring tiles (*up, right, down, left*), and some mechanism to point to a possible extension record that can carry the description of more complicated boundaries. In Figure 3, the information shown below the heavy line in each record is the part that should be relegated to the extension record.

Next, a generic function '*left of boundary*' is needed which must be compatible with all permitted boundary descriptions. The range of boundary curves that should be permissible depends strongly on the application domain. For integrated circuits, it seems plausible that for the near future, paths composed of linear line segments and circular — or possibly, elliptic — arcs are quite sufficient. The '*left of boundary*' function is easily implemented for such paths.

Actual corner stitching by the four corner pointers follows the scheme described by Marple.[2] It also works for rectangular tiles and thus need not be changed as the tiles take on more complicated shapes with curved boundaries or horizontal tile edges of zero length.

One elegant way to organize such a corner-stitched design system is to use an object oriented programming style. The various permissible boundary curve types are object classes that carry within their definitions the methods to test whether a point is '*left of boundary*'. New types of boundary curves can then easily be added without any recoding of previously introduced methods. However, for practical applications, where response time is a crucial performance parameter, this approach may have to wait until more efficient object-oriented run-time environments have been developed.

8. Data Storage Issues

The choice of data representation to yield the most compact representation of the layout depends on the type and frequency of the patterns occurring in the layout. At the present time in the context of IC layouts, the occurrence of curved boundaries is certainly a rarity, and straight edges dominate. Thus we must avoid penalizing all simple rectangular tiles with the extra burden of mostly empty fields that would be used only rarely for describing curved edges.

A practical solution is to make the default tile record the minimum structure required for a simple rectangular tile, and to add just one more pointer field. For the case of a normal rectangular tile, that pointer has the value nil; in all other cases it points to some extension of the tile data record that describes the geometry of slanted, curved, or composite edges.

This extension record would typically start with a type indicator that specifies which of several possible formats for describing the left-side boundary applies and how many fields it contains. For slanted straight edges, it would contain only one more number: the x-coordinate of the upper left corner. For quadratic and cubic Bézier curves, it would, in addition, contain the x,y-coordinates of one or two intermediate control vertices of the corresponding control polygon. For more complicated composite edges, this extension record contains a complete path descriptor, perhaps similar in concept to the manner a path is described in the PostScript language.[5]

In Section 3 we have already pointed out that from a functional point of view, composite boundaries should remain connected entities. In Section 5 we have further noted, that for numerical reasons, all curve segments should be maintained in the form of their original definitions. Both goals can be met if such boundaries are described in a separate location, while all tiles that use part of them as their left-side boundaries, reference them via a pointer. If the boundaries are higher-order Bézier curves, such sharing also results in additional storage economy; the various control vertices have to be stored only once.

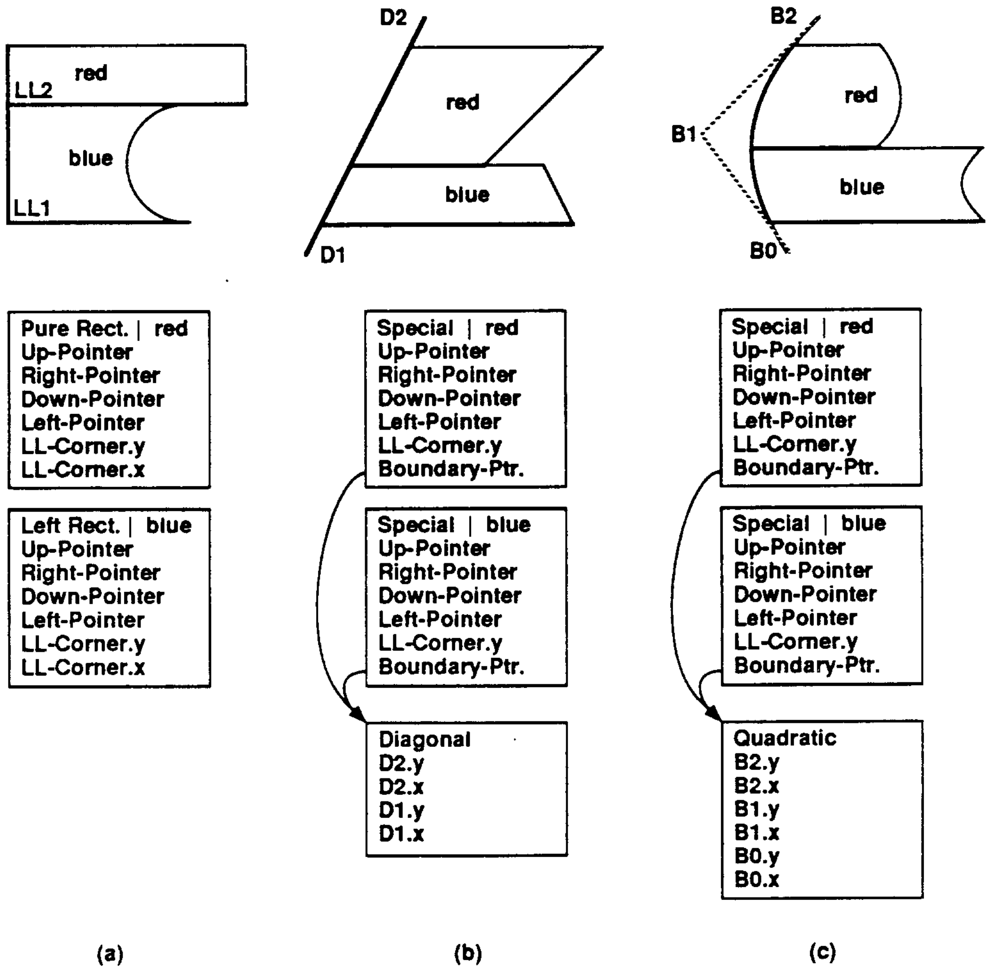


Figure 7: *Data storage for the original, rectangular, corner-stitched tile (a) arbitrary trapezoidal tiles (b) and tiles with shared curved boundaries (c).*

With such shared boundary descriptions, the two-tiered data structure shown in Figure 7 emerges. The tile type field, which normally describes the color or material of a rectangular tile, is enhanced with some extra bits that distinguish pure rectangular tiles from tiles with simple vertical left sides, and from all other tiles that need to reference a special boundary description. The storage overhead on rectangular tiles caused by the introduction of occasional tiles with curved edges could even be avoided completely. The description of the basic rectangular tile is kept at the minimum necessary: seven words per tile. When the type tile field indicates that the tile has a more complicated left-side boundary, the LL-Corner.x field becomes redundant, and this field can be used to accommodate the pointer to the boundary curve description.

For operational efficiency, it might be worthwhile to add some conventions that make handling of the simple — and probably most frequent — cases particularly efficient. For instance, it might be advisable to distinguish in the type field pure rectangles from tiles with a simple vertical left boundaries but with more complicated right boundaries (Fig. 7a). If a pure rectangle is encountered, simpler and faster routines can be used to determine the right-most extension of a tile for *'point in tile'* checks; for some operations, the neighbors on the right-hand side would never have to be accessed.

Similarly, if the only possible curve segments that will appear in the design are circular arcs, then a special encoding of such arcs should be considered. For instance, in Figure 8, in addition to the lower left corner of the tile (B), it would be sufficient to store the center (C) of the circular arc forming the left-hand side. The radius (r) is then implicitly given as the distance between these two points, and the upper termination of the arc (E) is given by the y-coordinate of the lower left corner of the tile directly above. To minimize numerical instabilities, a convention could be set that if this y-coordinate is equal to (within some epsilon) the y-coordinate of the center (C) of the circle plus one radius (r), then the upper left corner of the tile (E) should have the same x-value as the center of the circle (Fig. 8).

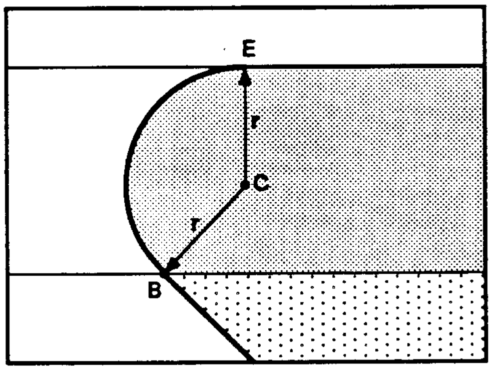


Figure 8: *Description of circular arc in a tile boundary segment.*

9. Discussion

We have shown that conceptually the corner-stitching technique can readily be extended to tiles with curved boundaries. Thus when occasionally a circular shape needs to be included in the layout, tiles with curved boundaries could be added to a corner-stitched environment. By suitable coding techniques, the storage overhead for the rectangular tiles in the layout can be kept small or even reduced to zero.

While there are few conceptual problems with curved tiles, there may well be implementation difficulties. In John Ousterhout's experience: ''*Corner-stitching is pretty straightforward at a high level, but it can become much more complicated when you actually sit down to implement things, particularly if you want the implementation to run fast. We found this to be distressingly true for a 45-degree implementation: it all seemed pretty simple until people started implementing it, and then it became incredibly complex.*''[6]

Thus, it is worthwhile to analyze carefully what amount of representational power is needed by a particular application domain: Does the application really need arbitrary slants or just 45-degree lines, — arbitrary spline curves or just circular arcs ? The minimal subset that gets the job done should be chosen, and the implementation should fully exploit the imposed restrictions, in order to minimize storage requirements, to guarantee numerical consistency, and to maximize efficiency of frequent operations. Nobody likes a system that runs sluggishly ! In particular, the object-oriented approach towards a generic layout system, outlined above, may be appropriate for academic studies and exploration, but it will not be able to compete with a more restricted and highly optimized implementation that uses special coding tricks tailored to the specific requirements and constraints.

Finally one should evaluate carefully what details of a layout should be captured by individual corner-stitched tiles. With the current state of the art of integrated circuits or systems, the special features that need an exact representation requiring tiles with curved boundaries are rather rare and appear in locations in the overall design that might never be subjected to general layout compaction. Micro-mechanical assemblies or special patterns on electro-optical chips typically have fixed geometry not just predefined topological ordering of the individual features. They may thus be encapsulated more sensibly into a special composite tile with which the whole assembly can be manipulated more efficiently. Such a tile may well be of rectangular or trapezoidal shape, and thus there would be no need to enhance the layout system to tiles with curved boundaries.

10. Conclusion

It has been shown that the corner-stitching data structure can be extended beyond just rectangles and trapezoids to general polygons and tiles with curved boundaries. The key insight is that at the topological level there is little difference between the tiling structure using trapezoids and triangles described by Marple[2] and more general 'trapezoidal' tiles with curved left- and right-side boundaries. This generalization can be achieved by simply adding the extra geometrical information needed to describe these boundary curves in a special extension to the the tile data-record. This type of extension is reminiscent of the enhancements that were made to the polyhedral UniGrafix system to arrive at the more recent UniCubix system[7] which can handle curved edges and surfaces expressible by triangular and quadrilateral cubic Gregory patches. For both these cases of systems extensions from linear to curved geometry, the topological and conceptual issues remain virtually unchanged, but the low-level geometrical operations are replaced with more sophisticated — and more difficult to implement — procedures.

Acknowledgments

John Ousterhout and Ping-San Tzeng made some critical and insightful comments on a first draft of this paper. Their input has resulted in the discussion in Section 9.

This work was supported in part by the Semiconductor Research Corporation under contract number SRC 90-DC-008.

References

1. J. K. Ousterhout, ''Corner Stitching: A data structuring technique for VLSI layoput tools,'' *IEEE Trans. on CAD*, vol. 3, no. 1, pp. 87-100, 1984.
2. D. Marple, M. Smulders, and H. Hegen, ''Tailor: A Layout System Based on Trapezoidal Corner Stitching,'' *IEEE Trans. on CAD*, vol. 9, no. 1, pp. 66-90, 1990.
3. R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to the Use of Splines in Computer Graphics*, Morgan-Kaufmann Publishers, Inc., Los Altos, California, 1987.
4. V. Pratt, ''Techniques for Conic Splines,'' *SIGGRAPH'85 Conf. Proceedings*, pp. 151-159, July 1985.
5. C. Geschke, Adobe Systems Incorporated, *PostScript Language, Turorial and Cookbook*, Addison Wesley, Reading, Massachusetts, 1985.
6. John. K. Ousterhout, *Private Communication*, June 1990.
7. C.H. Séquin, ''Procedural Spline Interpolation in UNICUBIX,'' *Proc. of the 3nd USENIX Computer Graphics Workshop*, pp. 63-83, Monterey CA, Nov. 1986.