Progress In Electromagnetics Research B, Vol. 40, 201–220, 2012

CONTOUR- AND GRID-BASED ALGORITHM FOR MIXED TRIANGULAR-RECTANGULAR PLANAR MESH GENERATION

T. A. Linkowski* **and P. M. Słobodzian**

Institute of Telecommunications, Teleinformatics & Acoustics, Wroclaw University of Technology, Wyb. Wyspianskiego 27, 50-370 Wrocław, Poland

Abstract—A mesh generation algorithm for the Method of Moments (MoM) is described. The algorithm, named CGSM, can mesh arbitrary planar shapes described with line segments and circular arcs into mixed triangular and rectangular cells. CGSM creates contours of the meshed shape and uses them to provide edge mesh (denser mesh near edges), creates an adaptive grid and uses it to insert axis-aligned rectangles in the interior, and finally, triangulates the remaining area (the Delaunay condition is imposed on the triangulation). CGSM is compared to two commercial applications (Designer® and IE3D$^{\mathrm{TM}}$) on the example of a 2-GHz hybrid ring coupler. The same simulation results are obtained. However, with CGSM, simulation time is significantly reduced.

1. INTRODUCTION

The meshing algorithm described in this paper, named CGSM, was already briefly described in [1], and a short comparison with Zeland IE3D$^{\mathrm{TM}}$ was given in [2]. Since then, the algorithm has been improved, and here, it is described in detail and compared to commercial software: Ansoft Designer® and IE3D$^{\mathrm{TM}}$.

A meshing algorithm for the MoM/SIE (Method of Moments/Surface Integral Equations [3, 4]) must, on the one hand, minimize the number of unknowns N in the MoM matrix equation [5] (N. B. for most basis functions, N is the number of internal mesh edges [4]). On the other hand, the algorithm has to maintain certain minimum density of cells. Specifically, the mesh should: a) match the meshed shape and preserve its area [4]; b) contain cells whose size is not greater

Received 1 March 2012, Accepted 12 April 2012, Scheduled 19 April 2012

* Corresponding author: Tomasz A. Linkowski (tomasz.linkowski@pwr.wroc.pl).

than 1/30–1/10 of the wavelength [6], and which do not overlap [7]; c) contain rectangles wherever feasible [7]; d) contain triangles without very small or very large angles [8]; e) be denser near edges [9].

According to the best of our knowledge, there are few publications concerning mesh generation for the MoM. There are several such papers but only about triangular/quadrilateral mesh generation [3, 10, 11], and there is truly extensive literature concerning triangular mesh generation for FEM [8, 12–14]. There are also domain decomposition methods other than mesh generation that are recently under investigation, e.g., [15].

The techniques from FEM cannot be directly used in MoM, since in FEM, the entire region under consideration has to be meshed, while in MoM/SIE, we mesh only the boundary of the structure to be analyzed. Moreover, some rules of mesh generation (e.g., mesh density distribution) are different, which precludes direct application of a FEM mesher in the MoM analysis. Yet, some of these techniques can be adapted for MoM, and we try to do it here.

On the other hand, purely triangular meshes require higher numbers of unknowns than mixed ones to represent the same quantities. As to quadrilateral meshes, doublets (i.e., edge basis functions) spanned over arbitrary quadrilaterals cannot approximate constant charge distribution (see List of Desired Properties of Doublets [3]), which both triangular and rectangular doublets can do. This serves as the motivation for investigating mixed triangular-rectangular meshes. Finally, note that doublets spanned over axis-aligned rectangles will have only one non-zero vector component (either X or Y), which can reduce the MoM matrix filling time.

As for the algorithms used in commercial MoM-based software (FEKO, Ansoft, IE3D$^{\text{TM}}$, WIPL-D), they are not open to public, and they also generate only triangular or quadrilateral meshes (with the exception of IE3D$^{\text{TM}}$and, partly, Ansoft Designer®). Therefore, a triangular-rectangular mesh generation algorithm for the MoM/SIE, named CGSM, has been proposed and described.

2. FORMULATION

We begin with several definitions: 1) *edge*: a line segment or circular arc defined in 2D, which has a *source* and *target point/vertex*; 2) *simple figure*: a planar simply connected list of edges; 3) *complex figure*: a figure that may have holes, i.e., a list of simple figures, first of which is the *boundary*, and the rest — *holes*; 4) *structure*: a set of complex figures; 5) *simple/complex polygon*: a simple/complex figure with segments only; 6) *cell*: a triangle or rectangle defined in 2D; 7) *mesh*: a set of non-overlapping cells covering the structure.

Table 1. Parameters of the CGSM mesh generation algorithm.

Symbol	Name	Range	Default
$\text{cells}_{\text{per}\lambda}$	cells per wavelength	5–50	20
$k_{\max}$	number of contour levels	≥ 0	2
$r\,(CW_k)$	width of k-th contour level (ratio)	0.01–1.0	0.1, 0.15
SWM	shape-wise mesh	on/off	off
$r\,(L_{\min})$	minimum length (ratio)	0.01–0.5	0.2
$r\,(CL_{\min})$	min. conversion length (ratio)	0.1–1.0	0.4
$r\,(GL_{\min})$	min. grid eye length (ratio)	0.2–1.0	0.8
$r\,(GL_{\max})$	max. grid eye length (ratio)	1.0–1.5	1.1
CA_{dflt}	default arc conversion angle	10–60 [°]	30°
$CRA_{\min}$	min. corner rounding angle	200–330 [°]	271°

The parameters of the CGSM algorithm are given in Table 1. Many of the parameters p are ratios $r\,(p)$ to the *nominal length* L_{nom}, defined as

$$r(p) = \frac{p}{L_{\text{nom}}}, \tag{1}$$

$$L_{\text{nom}} = \frac{299.8}{f_{\max\,\text{GHz}}\text{cells}_{\text{per}\lambda}\sqrt{\varepsilon_{reff}}}[\text{mm}], \tag{2}$$

where ε_{reff} is the effective dielectric constant calculated from substrate dielectric constant ε_r, substrate height h, and metallization width w [16].

One test structure will be used throughout the paper — it is constructed from segments and arcs, and contains one hole (Fig. 1). In the next section, it will be meshed using the default parameters given in Table 1 (for simplicity, $\varepsilon_r = 1$).

3. ALGORITHM

In this section, the four stages of CGSM are presented in detail: 1) Contour creation, 2) Grid creation, 3) Subdivision of edges, and 4) Mesh generation.

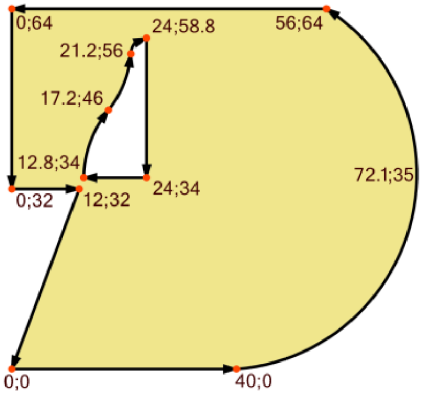


Figure 1. Test structure (units: mm).

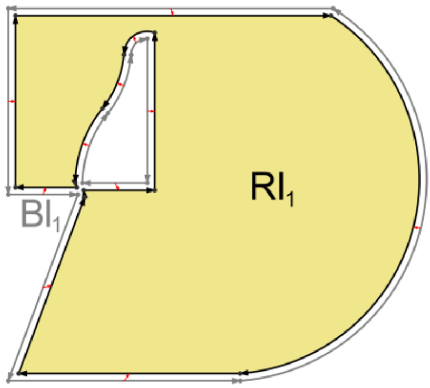


Figure 2. First-level remaining interior RI_1.

3.1. Contour Creation

Parameters used in this section: $k_{\max}$, CW_k, SWM, $L_{\min}$, $A_{\min}$, $CRA_{\min}$.

Contour creation is also known as *offsetting* or *buffering* [17]. Offsetting, but only for polygons, is implemented in Computational Geometry Algorithms Library (CGAL). CGSM uses contours to provide edge mesh (i.e., denser mesh near edges). If no contours are requested ($k_{\max} = 0$), stage 3.1 is omitted.

We define what follows: an *inner* or *outer contour* (C) of a simple figure (F) is another simple figure that is, respectively, inside or outside of F and has an orientation opposite to F. Conversely, figure F will be called a *base figure* for contour C. The distance between F and C will be called a *contour width* (CW_k).

Moreover, having an arbitrary complex figure (called *base interior*, BI), *remaining interior* (RI) is a complex figure based on BI, i.e., RI is composed of: 1) an inner contour of the boundary of BI, and 2) the outer contours of holes of BI (Fig. 2). The area between them is called a *contour area* (CA).

The initial contours of BI may require truncating, merging or removing if they overlap, before they make final RIs. The distance between BI and every point of final RI must be approx. the same. Furthermore, for given BI and CW_k, the number of created RIs is n_{RI}, where $n_{RI} \geq 0$ (Fig. 2). For next contour width (CW_{k+1}), all RIs become BIs ($RI_k = BI_{k+1}$), and the process is repeated (Fig. 3). The boundaries and holes of all RI_k are called k-level contours of the structure.

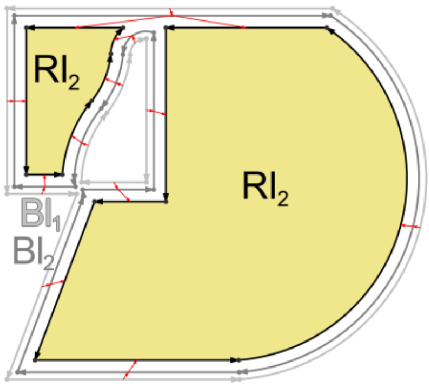


Figure 3. Second-level remaining interiors (RI_2).

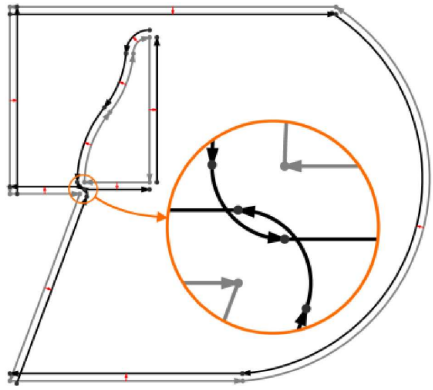


Figure 4. Opposite contour edges.

In conclusion, for every CW_k, one or more contour areas (CAs) and zero or more remaining interiors (RIs) are created. Moreover, if SWM (shape-wise mesh) is on, additional contours with a width ratio $r(CW) = 1$ are created.

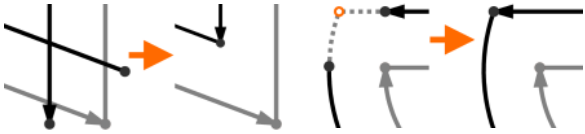
A single iteration of contour creation consists of the steps described in 3.1.1–3.1.7. After all iterations, we get a list of all CAs and a list of final RIs. We denote the contours from the last iteration ($k = k_{\max}$) as *top-level contours*, and these from all previous iterations $[k = 0 \ldots (k_{\max} - 1)]$ as *lower-level contours*.

Finally, we need to define *ancestry* of edges (used in 3.3.4 and 3.4.6). Edge e_A is an *ancestor* of edge e_D (and conversely — e_D is a *descendant* of e_A) if edge e_D is a contour edge based on edge e_A. Note that edges store information about their ancestry (in our figures, marked by tiny red arrows). Note also that an edge may have only one immediate ancestor but many descendants (e.g., Fig. 3).

3.1.1. Creation of Opposite Contour Edges

For each edge e of a complex figure F, a contour edge c is created (Fig. 4). Edge c has a direction opposite to e and is moved: inward if e belongs to the boundary of F, or outward if e belongs to a hole of F, by the distance of CW_k.

Specifically, if e is a segment, c is translated by CW_k along the direction perpendicular to e. If e is a circular arc, c is an arc with an opposite orientation, the same center, and the radius altered by CW_k. The source and target points of c are at angles at which, respectively, the target/source points of e are. Finally, if the angle between edges exceeds $CRA_{\min}$, a linking arc is added (Fig. 4).



(a) (b)

Figure 5. Interconnection in, (a) intersection point, (b) convergence point.

3.1.2. Interconnection of Edges

Consecutive edges are *interconnected* in this step, i.e., they are modified to meet in common points. A pair of consecutive edges f and g may be: 1) stretched to their intersection point, if there is any (Fig. 5(a)); 2) stretched to their *convergence point*, i.e., an intersection point of the supporting lines/circles of f and g, if there is any (Fig. 5(b)); 3) otherwise, they are connected with an arc.

3.1.3. Subdivision in intersections

Thus obtained closed figures will be called the *initial contours*. They may still intersect with themselves or with each other (see the close-up in Fig. 4). If they do not, we proceed directly to 3.1.6. Otherwise, we disassemble the initial contours into stand-alone edges $e_1 \dots e_n$, find all intersections $p_1 \dots p_m$ between them, and subdivide edges $e_1 \dots e_n$ in points $p_1 \dots p_m$.

An efficient algorithm for intersection checking (*sweep-line* algorithm [17]) exists, but for now, we only check if the minimum bounding rectangles of edges e_i and e_j overlap before calculating whether and where they intersect.

3.1.4. Removal of Edges that Are Too Close

Here, we dispose of the edges whose distance d to base interior BI is $d < CW_k$. Edge f is removed if the bounding rectangles of f and BI intersect, and if f is too close to any of the edges of BI. For each pair of edges, we also remove: edges that are exact opposites, and any of the doubled edges (Fig. 6).

3.1.5. Assembling of Remaining Contours

The remaining edges can finally be assembled into correct, non-intersecting contours. We start with an arbitrary edge and find consecutive edges until we come across this edge again. However, if at

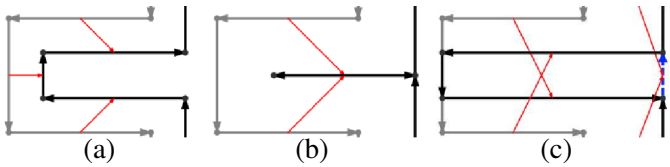


Figure 6. An element and its initial contour, (a) correct — no removal, (b) opposite edges — both removed, (c) doubled edges (dashed blue arrow) — one removed.

some point, having found edge e_i, there is more than one consecutive edge possible, we choose the one that is at the smallest angle from e_i (measured opposite to the orientation of the contour).

3.1.6. Adaptation of Contours

In this step, we merge consecutive edges that lie on the same line (segments) or circle (arcs). The following should also be removed (however, it has not yet been implemented): a) edges shorter than $L_{\min}$, b) gaps between edges smaller than $L_{\min}$, c) angles smaller than certain minimum angle (e.g., $< 2^\circ$).

Note that if — after the above operations — a contour is not a correct simple figure anymore (e.g., it consists of two segments), it has to be removed entirely.

3.1.7. Assembling Remaining Interiors and Contour Areas

From a list of correct contours, those with the same orientation as the base interior *BI* are treated as holes, and the rest — as boundaries. We assign holes to appropriate boundaries (by a point-in-figure check), thus obtaining *RI*s.

Having obtained remaining interiors (*RI*s), we also need to assemble contour areas (*CA*s). Let us assume that for an arbitrary base interior with n holes (of which n_{far} are farther from the boundary than $2CW_k$, and n_{close} are the rest, so that $n = n_{\text{far}} + n_{\text{close}}$), there are created n_{RI} remaining interiors. In such case, we can assemble: a) n_{far} *hole contour areas* (or even fewer than n_{far}, if some of the holes are closer to each other than $2CW_k$); b) one *boundary contour area*: its boundary is the boundary of *BI*, and it has $n_{RI} + n_{\text{close}}$ holes.

3.2. Adaptive Grid Creation

Parameters used in this section: L_{nom}, $L_{\min}$, $GL_{\min}$, $GL_{\max}$ (cf. Table 1).

Adaptive grid is an axis-aligned grid whose vertical and horizontal lines adapt to the vertices of edges. Its primary use is to facilitate distribution of rectangles in the mesh. N.B. only one grid is created for the entire structure.

The 2D grid (g_{XY}) is, in fact, composed of two 1D grids g_X and g_Y spanned along the x and y axis, respectively. A routine for creating such a 1D grid will be denoted *cGrid1D* — it takes a list of vertex coordinates (either x or y) as an input, and returns a sorted list of unique grid coordinates. These coordinates are separated by distance d, where $GL_{\min} \leq d \leq GL_{\max}$ (preferably, $d = L_{\text{nom}}$).

Routine *cGrid1D* stores three lists: *IC*, a list with unique *input coordinates*; *IW*, a list with weights of input coordinates (equal to the number of vertices with such a coordinate); and *GC*, a list with unique *grid coordinates*.

In a loop inside *cGrid1D*, one input coordinate is added in each iteration as a default — its value is $IC_{\text{new}} = GC_{\$} + L_{\text{nom}}$ (*$*: the last element), and its weight is $IW_{\text{new}} = 0$. Then, the algorithm considers only input coordinates IC_j fulfilling

$$GL_{\min} \leq IC_j - GC_{\$} \leq GL_{\max}, \tag{3}$$

and chooses for $GC_{\$+1}$ the IC_j with the greatest desirability D according to

$$D(IC_j) = IC_j - GC_{\$} + 0.05\, L_{\text{nom}}\, (IW_j)^3. \tag{4}$$

The above procedure can be interpreted as follows: next grid coordinate $GC_{\$+1}$ is moved by distance d from the previous coordinate $GC_{\$}$. By default, $d = L_{\text{nom}}$. However, if this were to cause that the distance between $GC_{\$+1}$ and the input coordinate IC_k immediately before or after it be very small, $GC_{\$+1}$ should be moved to IC_k, but only if $\max(d_{\min}, GL_{\min}) \leq d \leq GL_{\max}$, where $d_{\min}$ equals: $0.95L_{\text{nom}}$ if $IW_j = 1$ (i.e., for single vertex having such a coordinate), $0.6L_{\text{nom}}$ if $IW_j = 2$ (for two vertices) and 0, otherwise (three and more vertices).

Finally, *cGrid1D* can be used to create the x-axis grid g_X (input: x coordinates of all top-level contour edges) and the y-axis grid g_Y (input: y coordinates of edges created by subdividing all the top-level contour edges with the vertical lines of g_X).

Thus obtained grid g_{XY} (Fig. 7) is adapted only to the top-level contours. Therefore, a secondary grid (g'_{XY}) adapted to all contours is created, and if g_{XY} lacks horizontal or verticals lines in some region, they are copied from g'_{XY}.

3.3. Subdivision of Edges

Parameters used in this section: L_{nom}, CW_k, $L_{\min}$, $CL_{\min}$, $GL_{\max}$, CA_{dflt}.

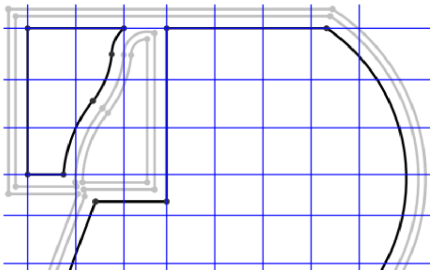


Figure 7. Adaptive grid g_{XY}.

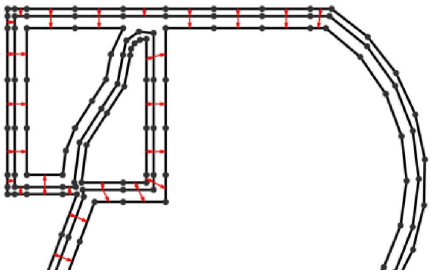


Figure 8. Structure after subdivision.

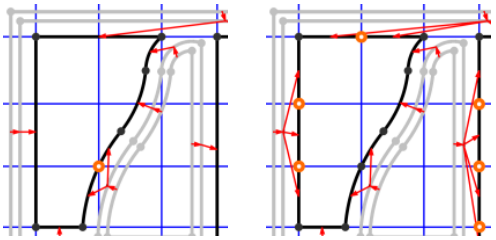


Figure 9. Subdivision, (a) in grid nodes, (b) of horizontal and vertical segments.

This stage consists in approximating each arc with one or more segments, and dividing each segment that is too long into shorter segments (Fig. 8). As a result, after this stage all figures will be referred to as *polygons*.

Note that an edge (e) may be subdivided in point p, if p lies on e and if the distances from p to the source and target points of e are $d \geq d_{\min}$. If not stated otherwise, $d_{\min} = L_{\min}$. The subdivision takes place in the following order: 1) subdivision of top-level contours ($k = k_{\max}$), 2) subdivision of lower-level contours in the descending order of contour levels, i.e., $k = (k_{\max} - 1) \ldots 0$.

3.3.1. Subdivision in Grid Nodes

The edges that contain one or more of the grid nodes (i.e., points where the horizontal and vertical lines of grid g_{XY} intersect) are subdivided in those nodes (Fig. 9(a)). This step assures that if a grid-based rectangle is to be added there, the vertex will be there also. Note that for lower-level contours, $d_{\min} = 0.5L_{\text{nom}}$.

3.3.2. Subdivision of Segments

This step has two parts: subdivision of horizontal/vertical segments (for lower-level contours, $d_{\min} = 0.5L_{\text{nom}}$), and subdivision of oblique segments.

The horizontal and vertical segments of the contour are subdivided in places where, respectively, vertical or horizontal lines of grid g_{XY} intersect with them (Fig. 9(b)). Again, this enables grid-based rectangle insertion (cf. Fig. 14).

The oblique segments longer than $d_{\max} = \max(L_{\text{nom}} + L_{\min}, GL_{\max})$ are uniformly subdivided into segments with length l_{new}. Specifically, each segment with length l_{segm}, if $l_{\text{segm}} > d_{\max}$, is subdivided into n segments with length $l_{\text{new}} = l_{\text{segm}}/n$, where $n = \lceil l_{\text{segm}}/L_{\text{nom}} \rceil$.

3.3.3. Approximation of Arcs with Segments

This step consists in converting each arc into one or more segments of the same length. The number of segments n depends on the conversion angle α_{conv} (initially equal to default angle CA_{dflt}). Having an arc with length l_{arc}, radius r, and sweep angle α_{sweep}, we will usually have $n = \lceil \alpha_{\text{sweep}}/\alpha_{\text{conv}} \rceil$ segments. Yet, if this were to produce segments of length $l_{\text{new}} < \min(CL_{\min}, CW_k)$ or $l_{\text{new}} > L_{\text{nom}}$, n must be decreased or increased, respectively. Moreover, we define that we want to have at least one segment per each 90° of the arc, so another requirement is $n \geq \alpha_{\text{sweep}} \cdot 2/\pi$. Having found n, we obtain $\alpha_{\text{conv}} = \alpha_{\text{sweep}}/n$.

Furthermore, we want to make sure that the approximating segments enclose an area (A_{segm}) equal to that enclosed by the arc (A_{arc}). The preservation of this area improves the accuracy of the MoM solution [4]. Therefore, the subdivision points are inserted not within the original radius r, but within a modified radius R (Fig. 10), obtained from the following formulas

$$A_{\text{arc}} = 0.5\ \alpha_{\text{sweep}}\, r^2, \tag{5}$$

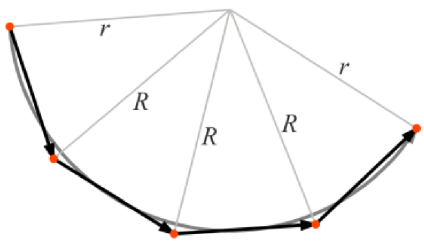


Figure 10. An arc with radius r and its four approx. segments.

$$A_{\text{segm}} = 0.5\ \left(r\,R + (n-2)\,R^2 + R\,r\right)\ \sin\alpha_{\text{conv}}. \tag{6}$$

By equating (5) and (6), we obtain a formula for the modified radius R as

$$R = \begin{cases} \dfrac{1}{2}\gamma r & n = 2 \\ \left(1 - \sqrt{1+\gamma(n-2)}\right)\dfrac{r}{2-n} & n > 2 \end{cases}, \tag{7}$$

$$\gamma = \frac{\alpha_{\text{sweep}}}{\sin(\alpha_{\text{conv}})}. \tag{8}$$

3.3.4. Descendant-based Subdivision of Segments

In this step, the subdivision is based on the descendants of each edge (for definition, see 3.1). Since the edges of top-level contours have no descendants, this step applies only to lower-level contours.

For a given contour C_k to be subdivided, we use an already subdivided contour of higher level C_{k+1}, containing descendants of edges in C_k. That is why we have to proceed in the descending order of contour levels $k = (k_{\max} - 1) \ldots 0$, and apply this step before the four steps described in 3.3.1–3.3.3. The general idea is similar to the receding front method from FEM [14].

The minimum distance for this step is $d_{\min} = \min(CL_{\min}, CW_k)$. The procedure is: for each edge e from contour C_k such that e is a segment, we project onto e the source and target points of the descendants of e (i.e., edges from C_{k+1}). Then, e is subdivided in each such projection point that lies on e within $d \geq d_{\min}$ from the source and target points of e. This assures that the contour area will contain as many rectangles as possible (cf. Figs. 8 and 18).

3.4. Mesh Generation

In the last stage, only two parameters are used: $L_{\min}$ and $CL_{\min}$.

In the following subsections (3.4.1–3.4.5), a routine *genMesh (P, $d_{\min}$)* is described, which generates a grid-based mesh for an arbitrary complex polygon P, with $d_{\min}$ as the minimal distance between cell nodes. We use it to mesh:

a) the final remaining interiors (*RI*s), consisting of the top-level contours (Fig. 3); meshes for each RI are obtained as $M_{RI} = genMesh(RI, L_{\min})$,
b) all contour areas (*CA*s); meshes for each CA are $M_{CA} = M_{\text{anc}} \cup \bigcup M_{\text{ancRem}}$, where each $M_{\text{ancRem}} = genMesh(P_{\text{ancRem}}, CL_{\min})$, and where the ancestor-based mesh M_{anc} and the remaining polygons P_{ancRem} are created in 3.4.6.

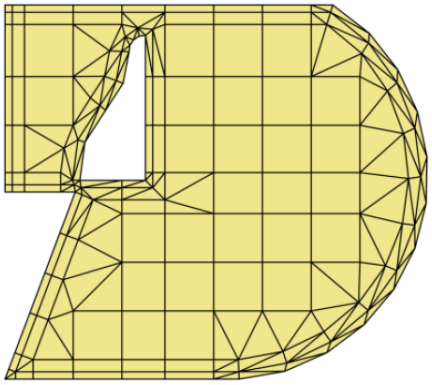


Figure 11. Final mesh (no. of cells: ■ 77, ▲ 152; unknowns: $N = 353$).

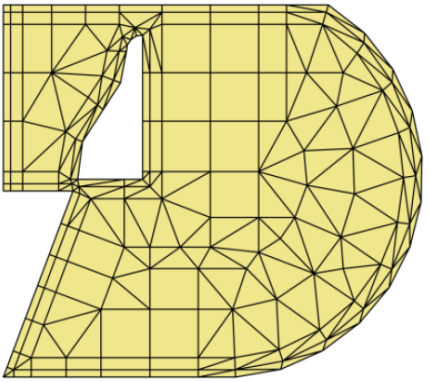


Figure 12. Final mesh when SWM is on (cells: ■ 67, ▲ 192; $N = 392$).

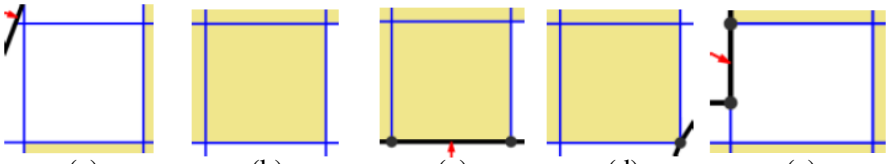


Figure 13. Grid eyes suitable for insertion, (a) no (an edge too close to the corner), (b) yes, (c) yes (edge exactly on the side), (d) yes (vertex in the corner), (e) no.

The final mesh M is thereby a sum $M = \bigcup M_{RI} \cup \bigcup M_{CA}$ (Figs. 11, 12).

3.4.1. Insertion of Rectangles into Grid Eyes

Routine *genMesh (P,* $d_{\min}$) begins with creating a mesh M_{rect} with "full" rectangular cells, i.e., rectangles inserted into eyes of the grid g_{XY} (where *grid eye* is an area limited by two consecutive horizontal and vertical lines of g_{XY}). A suitable grid eye is one that is entirely inside complex polygon P, does not contain any edge from P, and is sufficiently far ($d_{\min}$) from P (Figs. 13(a), (b)). However, a grid eye is also appropriate, if it has an edge lying exactly on its side (Fig. 13(c), or if it has a vertex of an edge in its corner (Figs. 13(d), (e)).

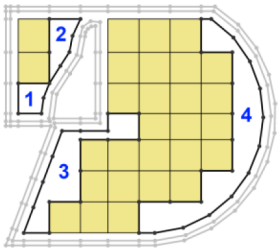


Figure 14. Rectangular mesh and four P_{rem}'s.

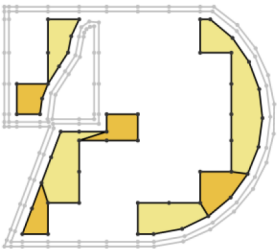


Figure 15. Division into monotone polygons.

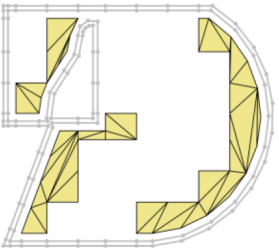


Figure 16. Initial triangulation of all P_{rem}'s.

3.4.2. Subtraction of Rectangles from the Polygon

In this step, we "subtract" mesh M_{rect} from the complex polygon P. To the list of edges of P (all of which are segments), we add all "outer" edges of M_{rect} (i.e., edges present only in one cell, not shared with others), with orientations opposite to that of P. Then, segments that are exact opposites are removed.

Subsequently, we assemble the edges into correct *simple polygons* (i.e., ones that are non-intersecting and without holes), which is analogous to step 3.1.5 in contour creation. The simple polygons, then, are assembled into complex polygons in a procedure analogous to the creation of *RI*s in 3.1.7. This yields zero or more *remaining complex polygons* P_{rem} (Fig. 14).

3.4.3. Triangulation

In this step, the remaining complex polygons P_{rem} are triangulated. First, each P_{rem} is divided into *monotone polygons* using an algorithm described in [17]. The division into monotone polygons consists in adding diagonals in both directions between certain points of complex polygon P_{rem} (Fig. 15). Having added such diagonals, we assemble remaining monotone polygons like in 3.1.5. Then, mesh M_{tri} with the triangulation of these polygons is created (Fig. 16) using an algorithm for triangulation of monotone polygons from [17].

3.4.4. Delaunay Flipping

In this step, we will assure that the triangles in M_{tri} are of good quality. For this purpose, the Delaunay condition [12, 17] is applied. It states that inside a circumcircle C of every triangle T there can be no vertices of any other triangle T' (however, a vertex of T' may lie

on the boundary of C). If the condition is violated by some pair of triangles T and T', their common edge must be flipped.

3.4.5. Insertion of Additional Nodes and Merging into Rectangles

Here, mesh M_{tri} is further manipulated. In each grid node (i.e., intersection point of the lines of grid g_{XY}) that is located: a) inside a triangular cell T (within $d \geq d_{\min}$ from the edges of T), T is divided into three smaller triangles; b) on the common edge e of two adjacent triangles T and T' (within $d \geq d_{\min}$ from the vertices of e), T and T' are each divided into two smaller triangles.

Often, there is no need for the above node insertion, yet it is required to assure that oversized cells are not created. Finally, for each pair of adjacent right triangular cells T and T' whose vertices form a rectangle, we merge them into one rectangular cell by removing their common edge. The final mesh M_P for the entire polygon P is $M_P = M_{\text{rect}} \cup M_{\text{tri}}$ (cf. Figs. 16 and 11).

3.4.6. Ancestor-based Meshing

In this step, the meshing is based on the ancestor-descendant edge pairs (for definition, see 3.1). Since such pairs can be found only in complex polygons made of contours with levels k and $k+1$, this step applies only to contour areas, and it is applied before the already described steps 3.4.1–3.4.5.

The step consists in creating a mesh M_{anc} with rectangular cells based on pairs of edges e_A and e_D, where e_A is an ancestor of e_D. For each contour area CA_k to be meshed, we will look for edges e_D in contours with level k (i.e., contours C_k).

For example, in the first contour area of the test structure (CA_1, Fig. 17), only one hole is a first-level contour C_1, while the boundary and the other hole are "zero" contours C_0 (i.e., they are parts of the original structure — cf. Fig. 2).

Now, for each edge e_D from C_k, we find its ancestor e_A in C_{k-1}, and if their vertices form a rectangle, we add such rectangle to mesh M_{anc}. Then, we apply step 3.4.2, i.e., we subtract M_{anc} from CA_k and assemble remaining complex polygons P_{ancRem}, which together with M_{anc} are the output of this step (Fig. 18).

3.5. Implementation

CGSM has been implemented using an object-oriented approach (C#, .NET 2.0). No external libraries, apart from those present in .NET, were used. However, some classes (*Point, Vector*, etc.) are based

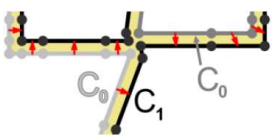


Figure 17. First-level contour area CA_1, (a) boundary: C_0 (light gray), and holes: C_0 (dark gray) and C_1 (black).

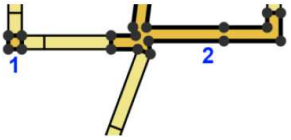


Figure 18. Rectangles of the ancestor-based mesh M_{anc} and two remaining complex polygons P_{ancRem}.

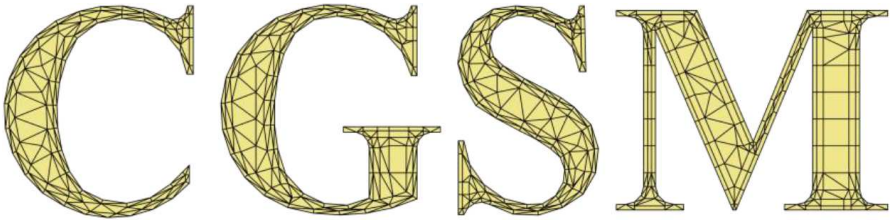


Figure 19. Mesh of a "CGSM" text (■ 120, ▲ 804; $N = 1300$).

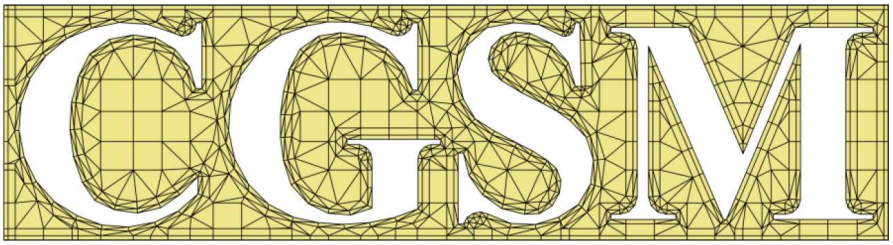


Figure 20. Mesh of the text in a box (■ 267, ▲ 1125; $N = 2034$).

on those described in the CGAL Manual. A separate floating-point number type (named *Real*) was also developed in order to overcome rounding-error problems. In *Real*, the difference between two values $a - b$ is calculated as zero if $|a - b| \leq |a| \times 10^{-10}$.

4. RESULTS

4.1. Example Meshes

Meshes of two example structures are given in Figs. 19 and 20. The data defining the structures and the parameters are not specified due to lack of space. For each, the generation time (on a 2-GHz Intel® CoreDuo) was below 1 s.

4.2. Comparison with Commercial Software

In this section, we consider a 2-GHz hybrid ring coupler that has been designed, fabricated and measured in [18] (Fig. 21). This coupler is simulated using three different meshes: two generated by commercial applications (Ansoft Designer®, v. 6.1, and Zeland IE3D$^{\mathrm{TM}}$, v. 12.21), and one — by CGSM. The simulation results are then compared to the measurements reported in [18].

The coupler is simulated on RT/Duroid 5880 ($h = 1.57\,\mathrm{mm}$, $\varepsilon_r = 2.2$). It has radius $R = 26.6\,\mathrm{mm}$, widths: $w_1 = 2.77\,\mathrm{mm}$ (in the three 90° sections) and $w_2 = 2.25\,\mathrm{mm}$ (in the 270° section), and arms that are 15.1-mm long (Fig. 21(b)).

The following has been simulated/measured: insertion loss ($S34/S21$, $S24$, $S31$), isolation ($S41$), and reflection ($S11$, $S44$). The frequency range is 0.5–3.5 GHz (step: 10 MHz), and max. relative interpolation error for S is 0.1%.

Mesh generation parameters are: **a)** common: $f_{\max} = 3.5\,\mathrm{GHz}$, cells per wavelength = 20, edge mesh = 0.1; **b)** Designer®: lambda target = 1/20, arc step size = 10°, max. number of arc points = 16; **c)** IE3D$^{\mathrm{TM}}$: segments per circle = 36, meshing scheme = contemporary, AEC layers/ratio = 1/0.10; **d)** CGSM: $CA_{dflt} = 10°$ (see Table 1). All remaining parameters are default.

The summary of the simulations is given in Table 2. CGSM provided 39% fewer unknowns than Designer® (which reduced the simulation time by 50%), and 32% fewer unknowns than IE3D$^{\mathrm{TM}}$(simulation time reduced by 42%). The created meshes are presented in Fig 22, and the simulation results — in Fig. 23.

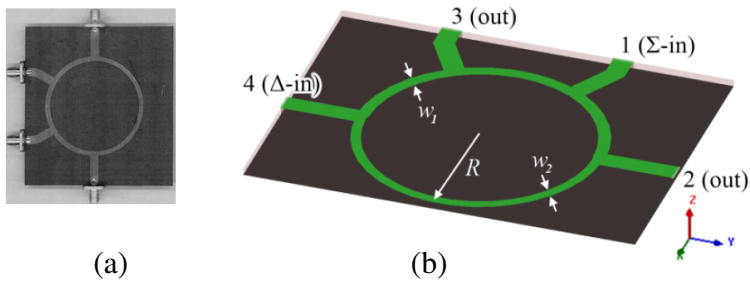


Figure 21. Hybrid ring coupler, (a) photograph of the circuit fabricated in [18], (b) simulation model in Ansoft Designer®.

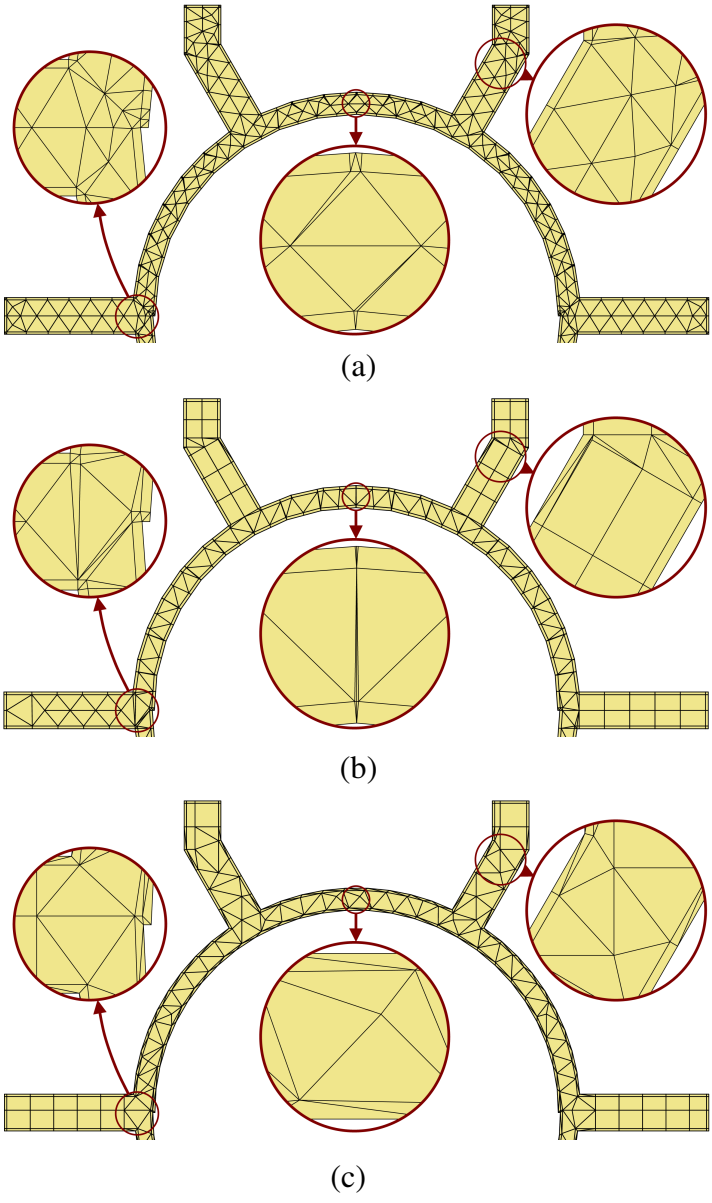


Figure 22. Fragments of meshes generated by, (a) Ansoft Designer®, (b) Zeland IE3D™, (c) proposed algorithm (CGSM).

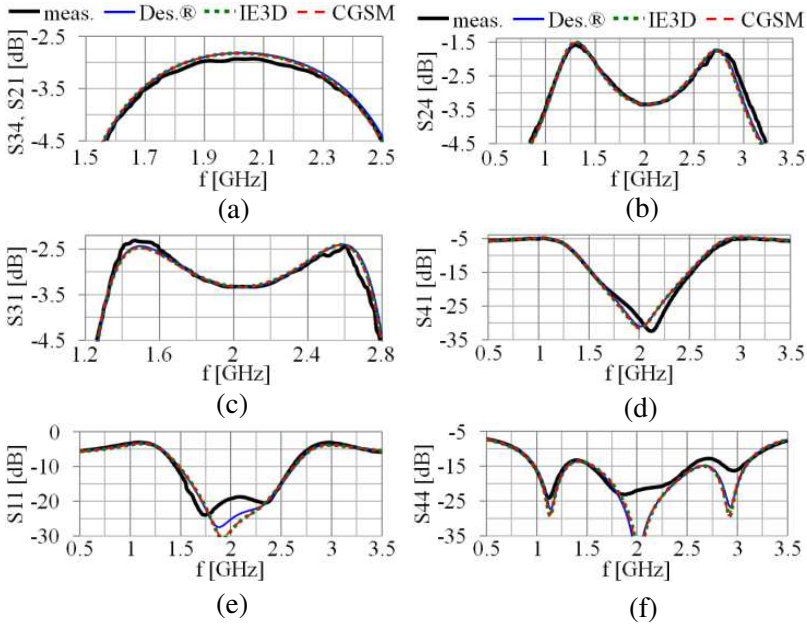


Figure 23. Comparison of measured and simulated results, (a)–(c) insertion loss, (d) isolation, (e)–(f) reflection. (a) S_{34}/S_{21}, (b) S_{24}, (c) S_{31}, (d) S_{41}, (e) S_{11}, (f) S_{44}.

Table 2. Summary of the simulations (20 cells per λ, edge mesh: 0.1).

Mesh generator	Rectangles	Triangles				t_{mesh}	N	t_s
		all	$< 2°$	$< 1°$	$> 176°$			
Designer®	192	638	26	0	0	1 s	1202	220 s
IE3D™	231	511	102	58	0	1.5 s	1076	190 s
CGSM	74	453	0	0	0	0.5 s	735	110 s

Key: $< \alpha$ — cell with an angle smaller than, t_{mesh} — mesh generation time (rounded to 0.5 s), N — number of unknowns (internal mesh edges), t_s — simulation time (two significant digits).

5. CONCLUSIONS

In the paper, a planar mesh generation algorithm for the MoM/SIE has been described. The algorithm (CGSM) first creates contours (offsets) of the meshed shape and a rectangular grid adapted to its edges, then subdivides the edges and the contours, and finally generates a triangular-rectangular mesh that is contour-based near the edges and grid-based in the remaining interior.

CGSM has been compared to two commercial applications: Designer® and IE3D™. The comparison shows that all the generated meshes (Fig. 22) provide almost identical simulation results (Fig. 23). However, CGSM outperforms the other generators by yielding smaller number of unknowns (Table 2), which reduces the simulation time (for Designer®, even by 50%). Finally, CGSM does not introduce any triangles that would have an angle smaller than 2°.

Notwithstanding these results, CGSM requires further work. First of all, the authors plan to address the special cases concerning contour creation (3.1.6). It would assure that no edges are shorter than $L_{\min}$ (unless such edges occur in the meshed shape) and that no angles are smaller than certain minimum angle. We also plan to improve mesh generation for contour areas (3.4) so that no cells larger than the contour width are created (as is *not* the case in Fig. 20).

In conclusion, the authors believe CGSM to be a valuable tool for the MoM/SIE. This seemingly simple algorithm is capable of generating meshes that — in the given example at the very least — have fewer unknowns than the meshes generated by Designer® or IE3D™, thus reducing the simulation time.

REFERENCES

1. Linkowski, T. A. and P. M. Slobodzian, "Automatic mesh generation for planar structures based on contours, adaptive grid and the delaunay condition," *EuCAP: 5th Eur. Conf. Ant. Propag.*, 1562–1566, Rome, 2011.
2. Linkowski, T. A. and P. M. Slobodzian, "Comparison of automatic planar mesh generation schemes facilitating edge meshing," *CEM: 8th Int. Conf. Computation in Electromagnetics*, 170–171, Wroclaw, Poland, 2011.
3. Kolundzija, B. M. and A. Djordjevic, *Electromagnetic Modeling of Composite Metallic and Dielectric Structures*, Artech House, Norwood, 2002.
4. Peterson, A. F., S. L. Ray, and R. Mittra, *Computational Methods for Electromagnetics*, IEEE Press, New York, 1998.
5. Harrington, R. F., "Matrix methods for field problems," *IEEE Proceedings*, Vol. 55, No. 2, 136–149, 1967.
6. Rao, S., D. Wilton, and A. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Trans. on Ant. and Propag.*, Vol. 30, No. 3, 409–418, 1982.

7. Newman, H. and P. Tulyathan, "A surface patch model for polygonal plates," *IEEE Trans. on Ant. and Propag.*, Vol. 30, No. 7, 588–593, 1982.
8. Lindholm, D., "Automatic triangular mesh generation on surface of polyhedra," *IEEE Trans. on Magn.*, Vol. 19, 2539–2542, 1983.
9. Sercu, J., N. Fache, F. Libbrecht, and D. de Zutter, "Full-wave space-domain analysis of open microstrip discontinuities including the singular current-edge behavior," *IEEE Trans. on Microwaves Theory and Tech.*, Vol. 41, No. 9, 1581–88, 1993.
10. Tsuboi, H., T. Asahara, F. Kobayashi, and T. Misaki, "Adaptive triangular mesh generation for boundary element method in 3D electrostatic problems," *IEEE Trans. on Magn.*, Vol. 34, No. 5, 3379–3382, 1998.
11. Moreno, J., M. J. Algar, I. Gonzalez Diego, and F. Catedra, "A new mesh generator optimized for electromagnetic analysis," *EuCAP: 5th Eur. Conf. Ant. Propag.*, 1734–1738, Rome, 2011.
12. George, P.-L. and H. Borouchaki, *Delaunay Triangulation and Meshing: Application to Finite Elements*, Hermes, 1998.
13. Lo, S. H., "Finite element mesh generation and adaptive meshing," *Progress Struct. Eng. Mater.*, Vol. 4, No. 4, 381–399, 2002.
14. Ruiz-Gironez, E., X. Roca, and J. Serrate, "The receding front method applied to hexahedral mesh generation of exterior domains," *Engineering with Computers*, published online, 1–18, 2011.
15. Martini, E., G. Carli, and S. Maci, "A domain decomposition method based on a generalized scattering matrix formalism and a complex source expansion," *Progress In Electromagnetics Research B*, Vol. 19, 445–473, 2010.
16. Balanis, C. A., *Antenna Theory: Analysis and Design*, 3rd edition, John Wiley & Sons, New Jersey, 2005.
17. Berg, M., O. Cheong, M. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd Edition, Springer, Berlin, 2008.
18. Okabe, H., C. Caloz, and T. Itoh, "A compact enhanced-bandwidth hybrid ring using an artificial lumped-element left-handed transmission-line section," *IEEE Trans. on Microwaves Theory and Tech.*, Vol. 52, No. 3, 798–804, 2004.