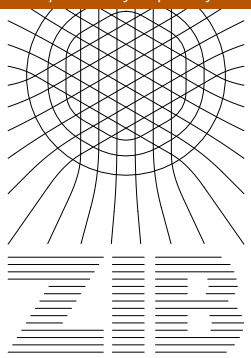


Konrad-Zuse-Zentrum für Informationstechnik Berlin

Rainer Roitzsch Ralf Kornhuber

BOXES

A Program to Generate Triangulations from a Rectangular Domain Description

Technical Report TR 90–9 (December 1990)

BOXES

A Program to Generate Triangulations from a Rectangular Domain Description

Rainer Roitzsch Ralf Kornhuber

Abstract BOXES computes a triangulation from a 2D domain description which consists of an arbitrary set of rectangles. Each rectangle may have attributes to control the triangulating process, define subdomain classes, or specify boundary conditions. The output of the program can be used as a coarse grid for KASKADE or one of its variants.

Chapter 1

Introduction

1.1 Motivation

The finite element program KASKADE [2, 8, 9] needs a coarse, initial triangulation to solve a problem by adaptive refinement techniques. Some applications as for example semiconductor device simulation lead to rather complex geometries with subdomains representing different materials where different sets of constants (doping, dielectric constant) are valid. See Figure 1.1 for a problem from [4].

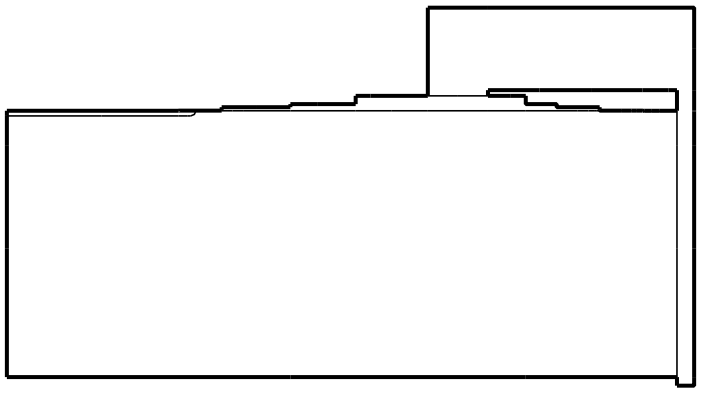


Figure 1.1: Different materials in a domain Ω

This geometry is described quite simply by a set of rectangular or almost rectangular boxes (Figure 1.2) ignoring some special features. For numerical reasons a suitable partition of Ω has to represent the inner boundaries. Note, that the factor between the longest and the shortest edge is well over 10^3.

BOXES generates from such a partition a coarse triangulation which is well suited to the adaptive refinement algorithm used in BDKASK (a program treating reverse biased *pn*–junction [6]) including anisotropic refinement [5]. The triangulation generated by BOXES from the set of rectangles shown in Figure 1.2 is depicted in Figure 1.3.

BOXES includes features to generate necessarily finer triangulations which have no flat triangles. As the resulting triangulation (Figure 1.4) involves much more nodes it is less desirable in the multi–level context, where the initial grid is intended to be very coarse.

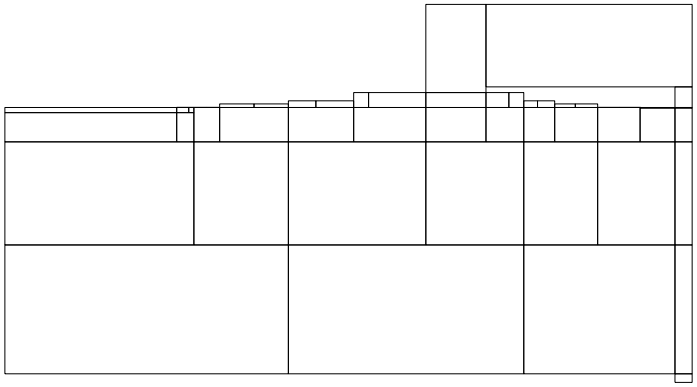


Figure 1.2: Initial partition of Ω by a set of boxes

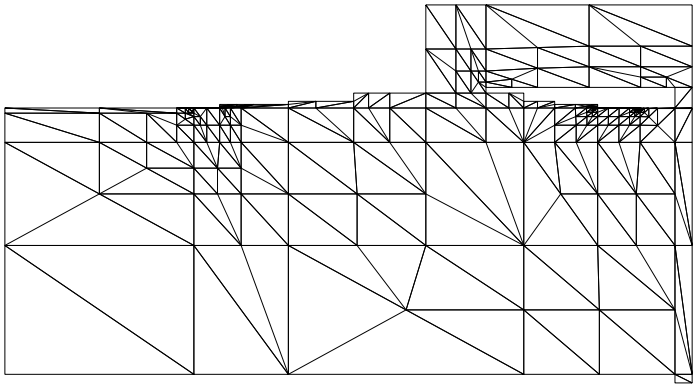


Figure 1.3: Triangulation generated by BOXES (267 nodes)

Note that the appropriate choice of an initial set of rectangles has to be done by hand but is supported by rigid checks for the correctness of the user input. The graphical facilities help the user to find errors and flaws in his design.

Future versions may support features like interactive graphical editing, input and output of standard formats (e.g. PIF [3]). At least the internal data structures are designed to make such extensions feasible.

1.2 Tasks of BOXES

As indicated in the preceding section BOXES is not a general mesh generator but a tool to generate triangulations of a 2D domain described by an initial set of rectangles. To simplify this we require that each side of a rectangle

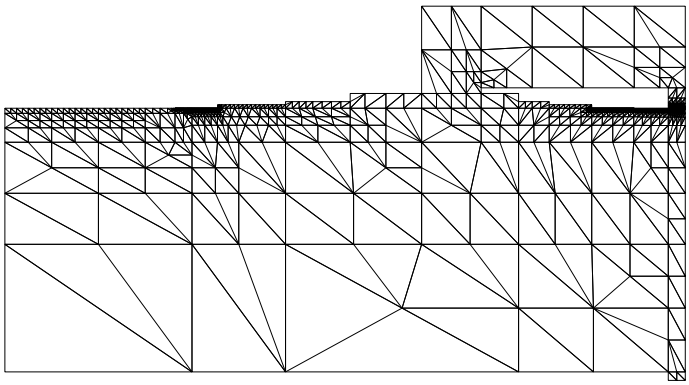


Figure 1.4: Fine triangulation generated by BOXES (2420 nodes)

touches at most two sides of other rectangles completely. A possible situation is illustrated in Figure 1.5 showing the coarsest triangulation for this domain on the right side.

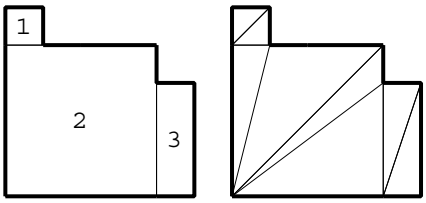


Figure 1.5: Rectangles and coarsest triangulation for a simple example

This triangulation has some nice triangles (box 1), pairs of flat triangles (box 3) and some really nasty triangles from green closure (box 2).

Without being too specific, a nice triangle is one with neither obtuse nor acute angles. A triangulation of nice triangles make a basis for a good discretization of a finite element approximation and acceptable condition numbers of the corresponding linear equation system.

Now the nice triangles inherit their quality by regular (red) refinement due to Bank et al. [1] (Figure 1.6). Pairs of flat triangles resulting from flat rectangles may be improved by directed (blue) refinement [5] as illustrated in Figure 1.7.

Hence the boxes 1 and 2 of Figure 1.5 are candidates for red refinement while the triangles resulting from box 3 are marked for blue refinement.

Note, that the blue refinement is not always possible (see Appendix). These operations are implemented in KASKADE and the need of such a refinement

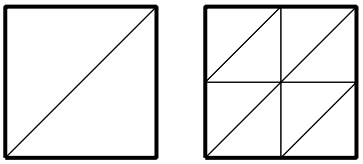


Figure 1.6: Nice rectangles refined red

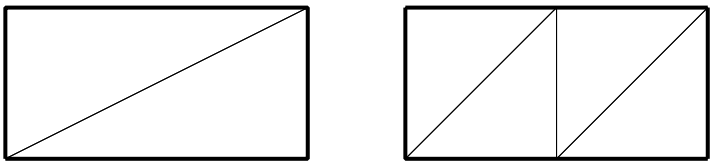


Figure 1.7: Flat rectangles refined blue

can be driven by an edge oriented error estimator or purely geometric criteria.

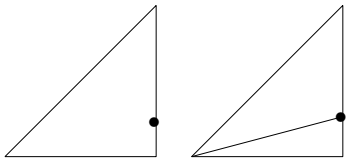


Figure 1.8: Green closures of irregular points

To remedy the nasty green closures (Figure 1.8) we apply successive red or blue refinement as illustrated in Figure 1.9 and 1.10.

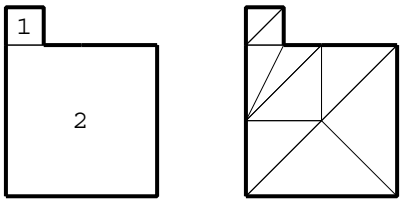


Figure 1.9: Resolving in the red case

In general we use a refinement algorithm which is roughly described as follows.

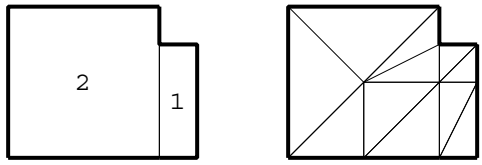


Figure 1.10: Resolving in the blue case

Algorithm 1: Resolving a coarse triangulation.

Step 1: If no coarse triangulation exists – generate one.

Step 2: Refine edges harmoniously.

Step 3: Refine all triangles with more than one edge refined or one edge refined twice. Use the blue refinement if the longest edge is twice as long as the shortest.

Step 4: Repeat Step 3 as long as triangles are refined.

Step 5: Generate the green closure.

The second step of Algorithm 1 needs some further explanation:

Algorithm 2: Refine edges harmoniously.

For all edges in the edge list do

Step 1: if the midpoint of the edge is in $[1/3, 1/3]$, go on.

Step 2: Divide the edge (midpoint at 0.5).

Step 3: Take the old midpoint as midpoint of the corresponding son.

Step 4: Append the new edges at the end of the edge list.

Step 4 introduces a kind of recursiveness: the edges are refined until a midpoint is in $[1/3, 1/3]$.

Remark: The refinement process described in Algorithm 1 allows no hierarchical interpretation as for example the usual father/son relationship of edged or triangles. Hence it does not fit in the data structures used in KASKADE.

Remark: To generalize Algorithm 1 to nonrectangular initial boxes the criterion for blue refinement in step 2 has to be extended to quadrangles. This can be done in a natural way as long as quadrangles are "almost" rectangles.

Remark: Rectangular structure may be lost, if conflicting requirement from opposite edges of the rectangle is imposed (see Figure 1.11). This can be remedied by a better choice of the initial partition as shown in Figure 1.12. The accurate choice of an initial set of boxes will be supported by a future version of BOXES.

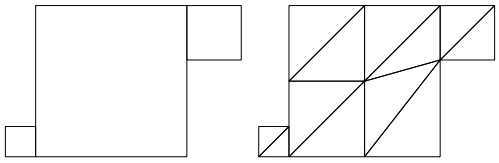


Figure 1.11: Loosing the rectangular structure

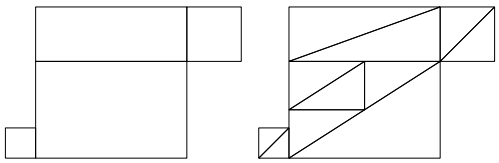


Figure 1.12: Saving the rectangular structure by an additional box

BOXES handles subdomain classes and boundary classes. That means that all triangles inherit a user specified class identifier from its enclosing rectangle. The same holds for boundary edges and points.

Additional features can be used to generate edges following a quarter circle or to discard one triangle from the triangulation.

BOXES generates output in the format for KASKADE or one of its variants. The final green closure is not submitted to KASKADE, which detects the corresponding irregular points by the input routine and generates its own green closure. This allows red refinement of these triangles in the adaptive refinement process.

Chapter 2

The Rectangular Domain Description

The syntax of the domain description is very simple. Items are defined on a single line, the type of the item is given by the first non blank character. Table 2.1 shows all types of items.

item types	description
`P`	point
`B`	box (rectangle)

Table 2.1: Item types

A point item must be further specified by an integer value which is used as an identifier, the x, y–coordinates, and a set of parameters. Each parameter starts with a character followed by optional values, see Table 2.2.

parameter	default	meaning
`T`⟨int⟩	`0`	class of the point
`N|D|C|I`	`I`	boundary condition
`S`⟨real⟩	`0.0`	start value for the iterative solver

Table 2.2: Parameters for points

The class of a point is useful in the KASKADE context when the user has to write routines to define a problem. These procedures may take advantage of the class to decide their behavior. An example is the use of different boundary value functions for different sets of points. The class of a new point is set to the class value of the edge of which it is midpoint. The rules to generate the class of edges are described later.

The boundary condition type denotes `'D'` for Dirichlet, `'N'` for Neumann, `'C'` for Cauchy (mixed), and `'I'` for no conditions. The start value might be used by an iterative solver of KASKADE.

Examples: The following lines define point `123` with (1.0,-3.4) as x, y–coordinates, and Dirichlet boundary conditions, and point `1` with start value 1.0. Note that parenthesis, colon and additional blanks are optional.

```
P 123 1.0 -3.4 D
P 5 (-1, -1) S 1.0
```

The box items use an integer identifier too, followed by four integers defining these edge points. The edge points should be oriented in mathematical positive sense. The parameters are given in Table 2.3.

parameter	default	meaning
`T`⟨int⟩	–	class of the box, default is the last used class valve, at the start 0
`F`⟨int⟩	`0`	coarse triangulation rule,
{`N`|`D`|`C`|`I`}⟨int⟩	–	boundary condition for edge ⟨int⟩
`B`⟨int_1⟩ ⟨int_2⟩	–	class of edge ⟨int_1⟩ is set to ⟨int_2⟩
`A`⟨int⟩	–	alternative class for the second triangle of the coarse triangulated box
`R`⟨int⟩	–	the diagonal should be a quarter circle with point ⟨int⟩ as midpoint

Table 2.3: Parameters for boxes

Figure 2.1 clarifies the numbering of the points, edges, and coarse triangles.

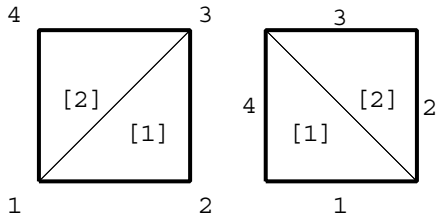


Figure 2.1: Numbering of boxes, F0 and F1

The class of boxes can be used to define subdomains.

A class value `-1` means that the triangle will not belong to the triangulation. The coarse triangulation rules are `F0` to place the diagonal edge of the box from bottom/left to top/right and `F1` for the other case (from top/left to bottom/right). The boundary condition type is defined by the characters `D`, `N`, `C`, and `I` followed by the number of the edge. (The number `5` denotes the diagonal edge.) New edges inherit their boundary condition from their parents or are set to `I` if they are new inner edges. The same is valid for the inheritance of edge classes which are set with the `B` parameter. In this case the value for new inner edges will be taken from the triangle (box) class.

Example:

```
B 1 (1,2,3,4) T1 N4 B1 5
B 23 (2,5,6,3) A2 D1 R1
```

The first box (number 1) is class `1`, with Neumann boundary condition on e4, and class `5` on e1. The second box should be a square, the midpoint of the diagonal is point 1, see Figure 2.2.

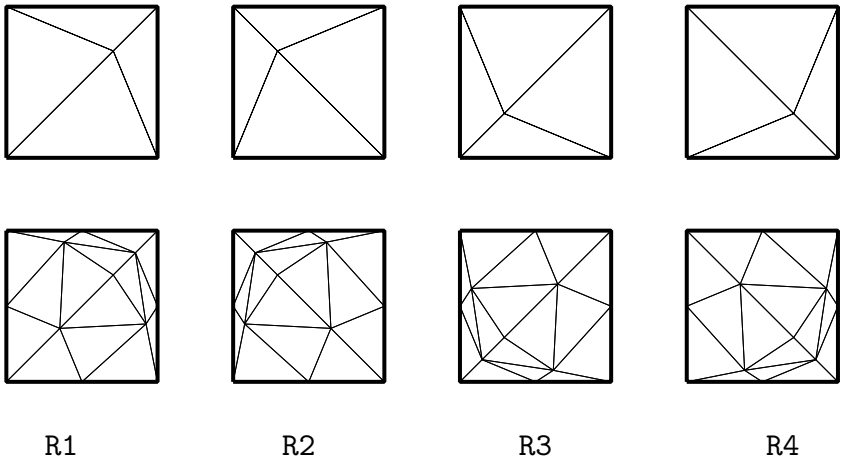


Figure 2.2: Arcs as edges, initial triangulation and first refinement

Remark: It is obvious from the preceding example that successive red refinement including arcs leads to arbitrary acute angles. This problem may be remedied by restricting the accuracy of the resolution of interior arcs by introducing a certain minimal angle. This strategy is supported by the fact that changes in the material are usually not exactly represented by such an arc.

Example:

```
C3 C1 D 1.0 E-5 E 11.9
```

Class `3` will have the color attribute `1` (RED), the doping `1.0 E-5` and the dielectricity constant `11.9` (silicon).

Remark: The lexical analysis ignores white space characters and certain fill characters like color, left or right parenthesis. (That's not really true, any

sequence of these characters is used as a delimiter). Upper and lower case letters are identified. The characters '#' and '%' start a comment up to the end of the current line.

A Complete Example: The following description defines the geometry shown in Figure 2.3. The resulting triangulations are shown in Figure 2.4 and 2.5.

```
p 1   0   5 d t1
p 2  80   5 d t1
p 3  80 140 t2
p 4   0 140 n t2
b1  1  2  3  4 d1 n4 t1

p 5  80 160 d t2
p 6   0 160 d t2
b2  4  3  5  6 d3 n4 t2

p 7 200   5 d t1
p 8 200 140 d t1
b3  2  7  8  3 d1 d2 t1

p 9 100 140 t1
p10 200 160 d t3
p11 100 160 t3
b4  9  8 10 11 d2 t1

b5  3  9 11  5 t1 a2 r4

p12 200 170 d t3
p13  80  170 d t3
b6  5 10 12 13 d2 d3 d4 t3
```

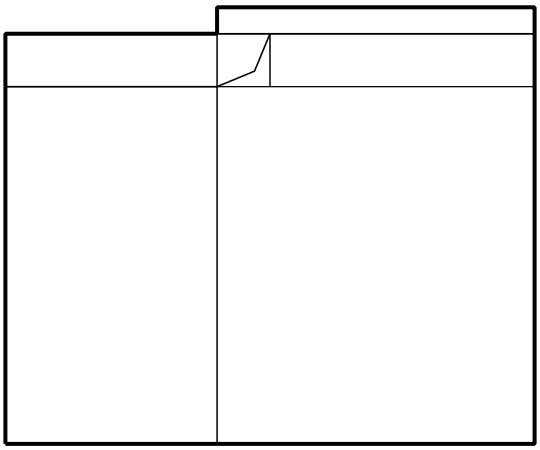


Figure 2.3: Boxes and boundaries of the example

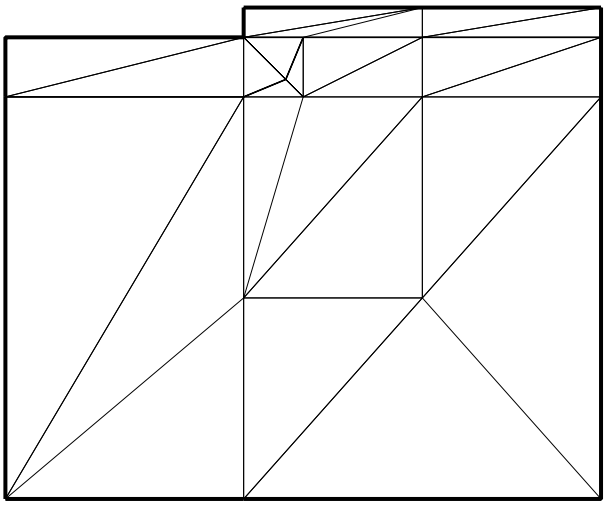


Figure 2.4: Triangulation after resolving

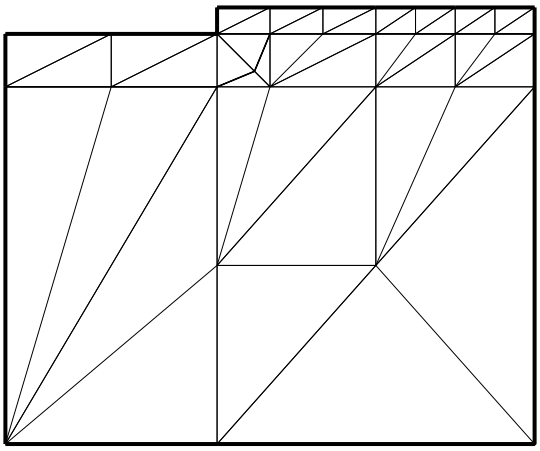


Figure 2.5: Fine triangulation after resolving

Chapter 3

Interactive Command Language

The interactive command language of BOXES has the same basis as the one of KASKADE [8]. The syntax is straightforward, the command name is followed by parameters and values. The commands `do`, `quit`, and `help` are the same as in KASKADE.

3.1 Basic BOXES Commands

The generation of a triangulation from the domain description is controlled by the following commands.

`read` ⟨filename⟩

⟨filename⟩ or ⟨filename⟩`.box` should contain a description of the domain as outlined in Chapter 2. It is not possible to read a second file.

`write` [⟨filename⟩]

If no ⟨filename⟩ is given the name of input file is taken to generate the output filename by the extension `.geo`. An old extension `box` is stripped.

`coarse`

The `coarse` command generates an initial triangulation. It is automatically triggered by the `resolve` command.

`resolve` [`onestep`]

The coarse triangulation is generated if it did not exist yet. Then the problem areas are resolved as described in Section 1.1 (Figure 1.9 and 1.10). The process can be done step by step with the `onestep` parameter.

`refine` { `red` | `blue` }

All triangles are refined in the "red" fashion if the `red` parameter is used. The `blue` parameter triggers "blue" refinement if the triangles with the appropriate geometrical attributes are found, i.e. the geometrical properties (acute and obtuse angles) are improved.

`inform`

The `inform` command just counts the points, edges, triangles, and boxes of the current triangulation and displays these values.

3.2 Graphic Commands

The graphical environment of BOXES uses one output device with the possibility of an attached second output stream. That means the graphical operations (line or text drawing, filling) are sent to two devices. The default device is a window on the screen, the attached device can be a postscript file.

`window`

The window command opens a window. It is useful if the user starts operating on the postscript device and decides later to use the screen.

`ps` { `tex` | `width` ⟨`cm`⟩ | `height` ⟨`cm`⟩ }

The `ps` command opens the postscript device or changes some attributes. The `width` and `height` parameters define the size of the output given in centimeters. The origin on the page is set to (`0.0,0.0`) through the `tex` parameter. These files can be used as input files for TEX postscript drivers.

`show` { [`index`] { `boxes`| `points`| `edges` } | `boundary` | `areas`
| `triangles` | `green` | `clear` }

The `show` command displays objects of BOXES. Objects are boxes or triangles, which are shown by their surrounding lines. The index of boxes, edges, and points may be additionally requested by the `index` parameters. The boundary is drawn by a thicker line and the different subdomains (`areas`) may be displayed through different colors. The "green" closure is added by the `green` parameter.

The `clear` parameter erases the window or/and inserts a show page command in the postscript file.

Acknowledgements Thanks to everybody who helped.

Appendix: Incompatible Triangulations

It is well–known (see [2, 7]) that red refinement of neighboring triangles causes forced red refinement of these triangles as shown in Figures A.1 and A.2.

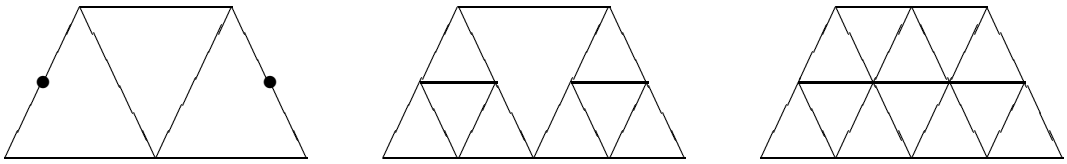


Figure A.1: Forced red refinement

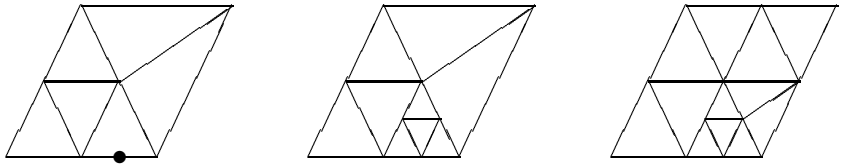


Figure A.2: Forced red refinement

Obviously this structural refinement is a recursive process which is hoped to terminate. It is easily seen that the refinement of a certain triangle cannot cause the forced refinement of one of its sons as long as only red refinement is used. Hence the whole process stops at least after the refinement of all triangles.

The situation becomes more complicated as soon as blue refinement is used. Figure A.3 shows how blue refinement leads to the structural red refinement of one of the resulting sons causing an infinite avalanche of forced red refinements. The reason for this phenomenon is that triangles with different refinement depth may have a common edge (due to blue refinement).

The solution for this problem in KASKADE is the substitution of blue refinement by red refinement if this situation occurs. Therefore after each request for blue refinement, the forced red refinement is immediately computed to check its effects.

The same problem has to be handled in BOXES. The sequence of pictures in Figure A.4 should illustrate this situation. Similar problem arise if we use our technique of resolving degenerately parted edges without using blue

Figure A.3: Blue refinement causing endless forced red refinement

refinement, see Figure A.5. Note the small and the thin box on top of the big box. They are revealed only at higher magnifications. The reason for infinite forced refinement is the same as in the case above: forced red refinement around a point.

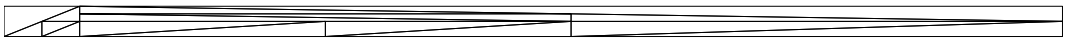
Obviously the problems occurring in Figures A.4 and A.5 can be settled using forced blue refinement where it is possible and geometrically desirable. Supported by computational experiments we conjecture that infinite loops are prevented in this way as long as rectangular initial boxes are used. In the case of arbitrary quadrangles this strategy fails as follows from a simple extension of the above example (Figure A.3).

Initial triangulation:

after the first blue refinement:

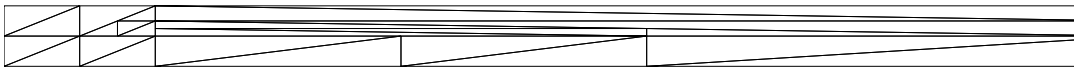
after the second blue refinement:

and the forced red refinement:

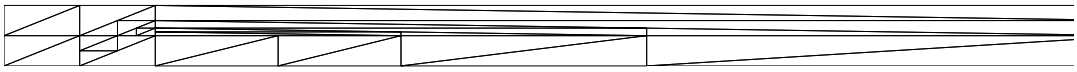


after the third blue refinements:

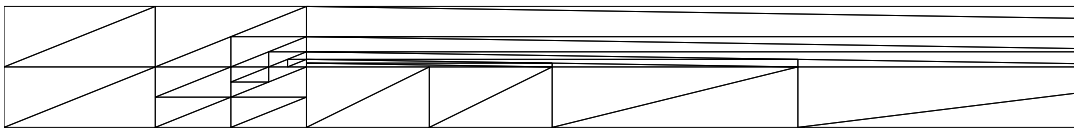
and the forced red refinements (magnified twice):



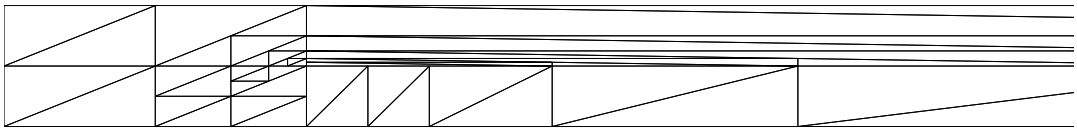
after the fourth blue + forced red refinement (magnified twice):



after the fifth blue + forced red refinement (magnified fourfold):



after the sixth blue refinement (magnified fourfold):



forced red refinement will lead to the catastrophe

Figure A.4: The dangers of blue refinement

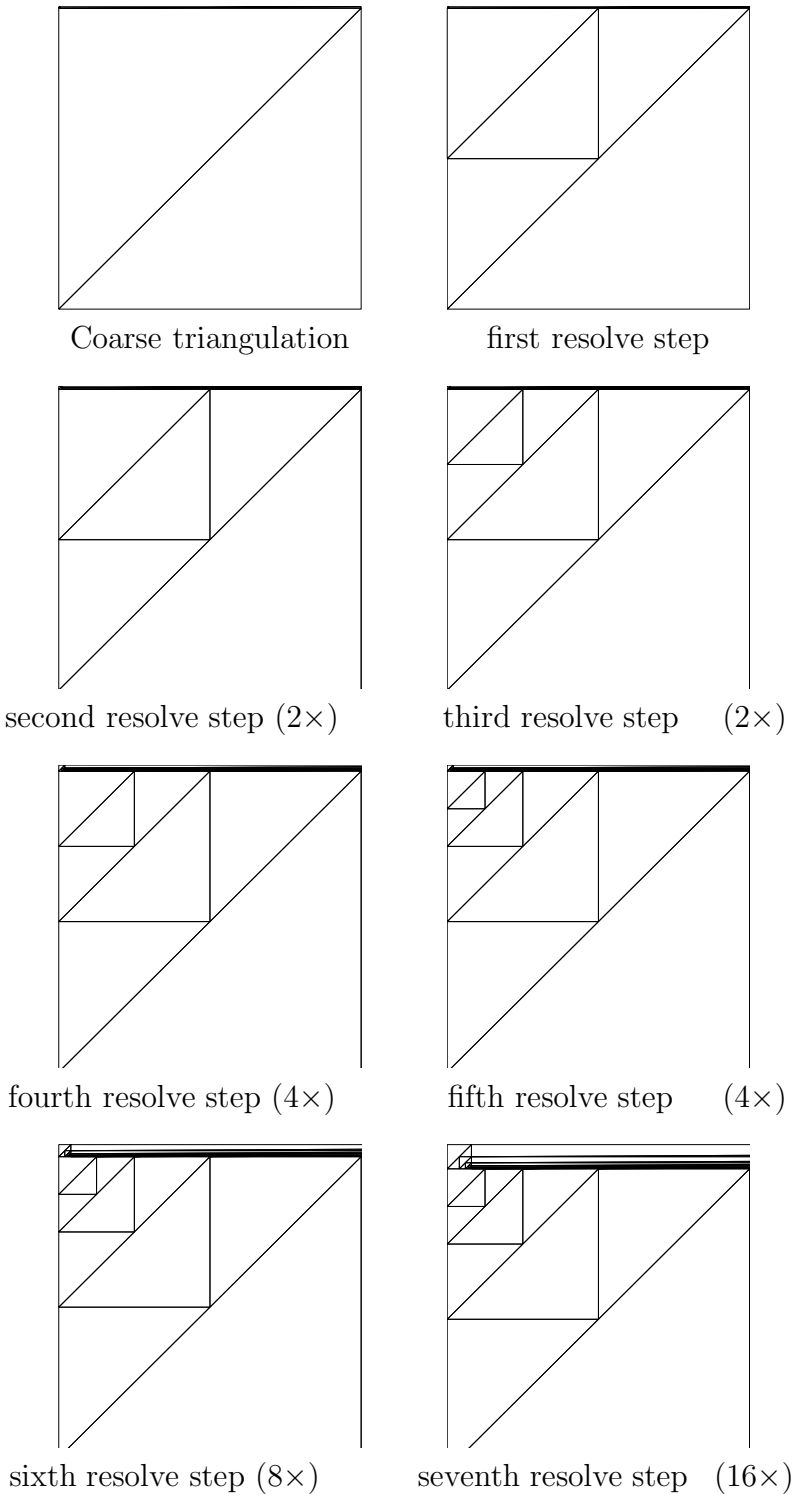


Figure A.5: The dangers of BOXES resolving techniques

Bibliography

[1] Bank, R.E., Sherman, A.H., Weiser, A: *Refinement Algorithms and Data Structures for Regular Local Mesh Refinement.* In R. Stepleman et al.(Eds.), Scientific Computing, p 3-17. IMACS/North–Holland, Amsterdam (1983)

[2] Deuflhard, P., Leinen, P., Yserentant, H.: *Concepts of an Adaptive Hierarchical Finite Element Code.* IMPACT 1 , p. 3–35 (1989)

[3] Duvall, S.G.: *An Interchange Format for Process and Device Simulation.* IEEE Trans. on Computer-Aided Design **7** p 741–753 (1988)

[4] Falck, E., Gerlach, W.: Private communications (1990)

[5] Kornhuber, R., Roitzsch, R.: *On Adaptive Grid Refinement in the Presence of Internal or Boundary Layers .* IMPACT **2** p 40–72 (1990)

[6] Kornhuber, R., Roitzsch, R.: *Self Adaptive Finite Element Simulation of Reverse Biased pn–Junctions.* SC 90–10 Konrad–Zuse–Zentrum Berlin (ZIB) (1990)

[7] P. Leinen: *Ein schnell adaptiver Löser für elliptische Randwertprobleme auf Seriell– und Parallelrechnern.* Dissertation, Universität Dortmund (1990)

[8] Roitzsch, R.: *KASKADE User's Manual.* TR89–4 Konrad–Zuse–Zentrum Berlin (ZIB) (1989)

[9] Roitzsch, R.: *KASKADE Programmer's Manual.* TR89–5 Konrad–Zuse–Zentrum Berlin (ZIB) (1989)