Hierarchical Advancing Front Triangulation Using Symmetry Properties

Victor G. Ganzha and Dmytro Chibisov[1], Evgenii V. Vorozhtsov[2]

[1] Institute of Informatics, Technical University of Munich, Garching 85748, Boltzmannstr. 3, Germany; ganzha@in.tum.de, chibisov@in.tum.de

[2] Institute of Theoretical and Applied Mechanics, Russian Academy of Sciences, Novosibirsk 630090, Russia; vorozh@itam.nsc.ru

Abstract. We show how the implicitly given complex geometric regions can be subdivided into symmetric parts to speed-up the computationally expensive advancing front triangulation and finite element computation on the resulting grid. We calculate for the implicitly given geometric region the symmetry axes by computing the invariant finite matrix group, such as, for example, reflections and rotations. It is proposed to use the so-called R-Functions for the description of complex geometric regions, for finding (with the aid of Maple) the symmetry groups of the given region, and for finding the initial front for the advancing front method by boundary discretization using octal trees. The advancing front triangulation can also be performed for only one of the symmetric parts, and the resulting grid is assembled. It is shown that the use of symmetry properties of a given planar region enables the CPU time savings by a factor from 3 to 8.

1 Introduction

In our work we are interested in the integration of computer aided geometric design (CAGD) and numerical simulation in such a way that would allow us to design robust, efficient, and reliable scientific software. On the one hand the physical or numerical properties of the computational problem make demands on the possible geometric representation of an object under consideration. On the other hand different topological and geometric representation of an object exist, which can not be converted to each other in simple and efficient way.

By far the most common representation for curves and surfaces in CAGD is the parametric representation (Bezier, NURBS or BSPline curves). But the researchers recognized early the power of implicit curves and surfaces for the purpose of modeling and simulation. The present paper shows how the complex geometric regions whose boundaries are given as implicit algebraic curves can be subdivided into symmetric parts to speed-up the computationally expensive advancing front triangulation and finite element computation on the resulting grid. We calculate for the geometric region given as an implicit curve the symmetry axes by computing the invariant finite matrix group of reflections. The advancing front triangulation can also be performed for only one of the symmetric parts, and the resulting grid is assembled. By using the possibility offered by computer algebra to perform FEM calculations with symbolically given boundary conditions, as shown in [1], the partial FEM solutions on subregions can be assembled what leads to CPU time savings.

2 Hierarchical Methods in Computer Aided Geometric Design and Symmetry

In the present paper we consider the constructive hierarchical geometry representations. A constructive representation defines an object by the sequence of operations for constructing an object [4]. The most common constructive representation is called Constructive Solid Geometry (CSG) and uses the boolean (set theoretic) operations. The operation sequence is typically stored as a tree. For example, the object shown on the right hand side in the following figure can be constructed from rectangle, circle, and cone using set union and difference operations.

To convert this set theoretic operations to a real valued functions the R-Functions proposed in [7] (a short introduction and basic applications of R-Functions can be found in [9] too) can be used. R-Functions allow us to write easily an equation for an object of arbitrary shape, in the same way as one forms the

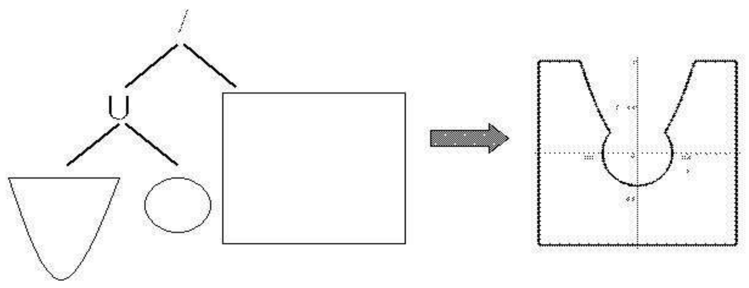


Fig. 1.

solid by the boolean operations. If $\mathbf{x} = (x_1, ..., x_n)$ is a point in R^n, then:

$$\begin{array}{ll} f(\mathbf{x}) > 0 & \text{if } \mathbf{x} \text{ is inside the object} \\ f(\mathbf{x}) = 0 & \text{if } \mathbf{x} \text{ is on the boundary of the object} \\ f(\mathbf{x}) < 0 & \text{if } \mathbf{x} \text{ is outside the object} \end{array} \tag{1}$$

The set-theoretic operations on objects described as R-Functions can be defined as follows

$$\begin{aligned} f_1(\mathbf{x}) \cup f_2(\mathbf{x}) &= f_1(\mathbf{x}) + f_2(\mathbf{x}) + \sqrt{f_1^2(\mathbf{x}) + f_2^2(\mathbf{x})} \\ f_1(\mathbf{x}) \cap f_2(\mathbf{x}) &= f_1(\mathbf{x}) + f_2(\mathbf{x}) - \sqrt{f_1^2(\mathbf{x}) + f_2^2(\mathbf{x})} \\ f_1(\mathbf{x}) \backslash f_2(\mathbf{x}) &= f_1(\mathbf{x}) - f_2(\mathbf{x}) - \sqrt{f_1^2(\mathbf{x}) + f_2^2(\mathbf{x})} \end{aligned} \tag{2}$$

Note that the boundary of the geometric region is represented as roots of the R-Functions $f(\mathbf{x}) = 0$. We can isolate the squared roots in (2) and sqare left and right hand side respectively, in case of the intersection, for example :

$$\begin{gathered} -\sqrt{f_1^2(\mathbf{x}) + f_2^2(\mathbf{x})} = f_1(\mathbf{x}) + f_2(\mathbf{x}) \\ f_1^2(\mathbf{x}) + f_2^2(\mathbf{x}) = f_1^2(\mathbf{x}) + 2f_1(\mathbf{x})f_2(\mathbf{x}) + f_2^2(\mathbf{x}) \\ 2f_1(\mathbf{x})f_2(\mathbf{x}) = 0 \end{gathered}$$

In this way we obtain the the point set containing boundary given by the equation $f_1(\mathbf{x})f_2(\mathbf{x}) = 0$:

$$\begin{aligned} \partial(f_1(\mathbf{x}) \cup f_2(\mathbf{x})) &\subseteq \{\mathbf{x} : f_1(\mathbf{x}) * f_2(\mathbf{x}) = 0\} \\ \partial(f_1(\mathbf{x}) \cap f_2(\mathbf{x})) &\subseteq \{\mathbf{x} : f_1(\mathbf{x}) * f_2(\mathbf{x}) = 0\} \\ \partial(f_1(\mathbf{x}) \backslash f_2(\mathbf{x})) &\subseteq \{\mathbf{x} : f_1(\mathbf{x}) * f_2(\mathbf{x}) = 0\} \end{aligned} \tag{3}$$

In the next section we will show, how the domain boundary obtained according to (3) can be used to compute finite symmetry groups of the domain.

The rectangle in Fig. 1 can be constructed as the intersection of 4 half-spaces according to (2) :

$$\begin{aligned} f_1(x, y) &= 1 - x \\ f_2(x, y) &= 1 + x \\ f_3(x, y) &= 1 - y \\ f_4(x, y) &= 1 + y \end{aligned}$$

Then we obtain:

$$\mathrm{Rect}(x, y) = (f_1 \cap f_2) \cap (f_3 \cap f_4) = f1 + f2 - \sqrt{f1^2 + f2^2} + f3 + f4 -$$
$$\sqrt{f3^2 + f4^2} - \sqrt{\left(f1 + f2 - \sqrt{f1^2 + f2^2}\right)^2 + \left(f3 + f4 - \sqrt{f3^2 + f4^2}\right)^2}$$

Another representation of the rectangle boundary is the one according to (3):

$$\partial \mathrm{Rect}(x,y) \subseteq (x-1)(1+x)(y-1)(1+y)$$

Other primitives used in the above example are:

$$\mathrm{parabola}(x,y) = y - 3\,x^2; \qquad \mathrm{circle}(x,y) = -x^2 - y^2 + \frac{1}{8}.$$

The complete object is given by

$$O(x,y) = Rect \backslash (circle \cup parabola) =$$
$$Rect - parabola - circle - \sqrt{parabola^2 + circle^2} - \sqrt{Rect^2 + \left(parabola + circle + \sqrt{parabola^2 + circle^2}\right)^2}$$

Obviously the following symmetry properties hold:

$$\mathrm{Rect}(x,y) = \mathrm{Rect}(\pm x, \pm y)$$
$$\mathrm{Rect}(x,y) = \mathrm{Rect}(\pm y, \pm x)$$

or in the matrix form

$$\mathrm{Rect}(D_4\mathbf{x}) = \mathrm{Rect}(\mathbf{x})$$

$$D_4 = \left\{ \begin{bmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{bmatrix}, \begin{bmatrix} 0 & \pm 1 \\ \pm 1 & 0 \end{bmatrix} \right\}$$

D_4 is the well known dihedral group whose elements correspond to rotations and reflections in the plane. The circle has a symmetry group SO_2

$$SO_2 = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}$$

and the parabola is a reflection symmetric with respect to the y-axis:

$$Ry = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

In the present paper we present an algorithm for computation of the symmetric decomposition shown in the following figure and show how the costs of advancing front triangulation can be reduced by per-

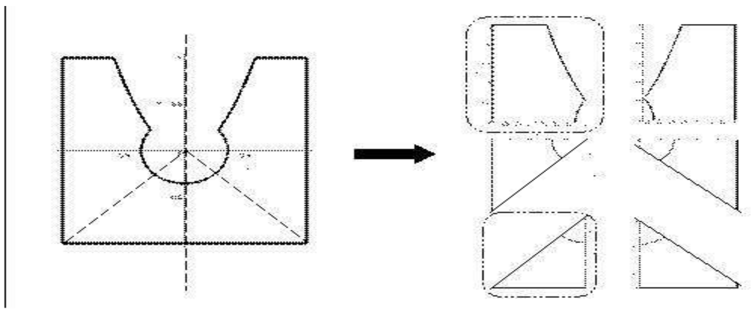


Fig. 2.

forming them on the symmetric parts only marked in the figure.

3 Computing the Invariant Matrix Group

We start with the decomposition of single polynomials from which our region was built. Consider the polynomial

$$f(x,y) = \sum_{i,j=0}^{N} a_{i,j} x^i y^j$$

Let the transformation matrix be given by

$$G = \begin{bmatrix} g_{1,1} \; g_{1,2} \\ g_{2,1} \; g_{2,2} \end{bmatrix}$$

Then the polynomial remains invariant iff

$$f(G * \mathbf{x}) = f(G * (x,y)^T) = \sum_{i,j=0}^{N} a_{i,j} (g_{1,1}x + g_{1,2}y)^i (g_{2,1}x + g_{2,2}y)^j = f(\mathbf{x})$$

Exponentiating and collecting coefficients of like power leads to

$$f(G * \mathbf{x}) = \sum_{i,j=0}^{N} x^i y^j f_{i,j}(g_{1,1}, g_{1,2}, g_{2,1}, g_{2,2}) = \sum_{i,j=0}^{N} a_{i,j} x^i y^j = f(\mathbf{x}).$$

In this way we obtain the following system of N^2 equations ([11])

$$f_{i,j}(g_{1,1}, g_{1,2}, g_{2,1}, g_{2,2}) = a_{i,j},$$

where $f_{i,j}$ are some functions, which depend on $(g_{1,1}, g_{1,2}, g_{2,1}, g_{2,2})$ and can be computed, for example, with Maple, as follows:

```
# transform the polynomial according to (10) and expand it
 [> fG:=expand(f(op(convert(G.xx,list))),{x,y});

# calculate the coefficients of f(G*x)
[> f:=[coeffs(fG,{x,y})];

# calculate the coefficients of f(x)
[> a:=[coeffs(f,{x,y})];
```

The last step is to solve the system of equations to obtain the invariant matrix group

```
[> solve({(f[i]=a[i])$i=1..N}):
```

For example, for the $Rect(x,y)\backslash circle(x,y) = -(x-2)*(y-2)*(x+2)*(y+2)*(x^2+y^2-1)$ we obtain the following system of equations

$$\begin{aligned}
-20 &= -20\,{g_{1,1}}^2 - 20\,{g_{2,1}}^2, 16 = 16, 4 = 4\,{g_{1,1}}^4 + 4\,{g_{2,1}}^4 + 9\,{g_{1,1}}^2{g_{2,1}}^2,\\
9 &= 9\,{g_{1,2}}^2{g_{2,1}}^2 + 24\,{g_{2,1}}^2{g_{2,2}}^2 + 9\,{g_{1,1}}^2{g_{2,2}}^2 + 36\,g_{1,1}g_{2,1}g_{2,2}g_{1,2} + 24\,{g_{1,1}}^2{g_{1,2}}^2,\\
-1 &= -6\,{g_{1,2}}^2{g_{2,1}}^2{g_{2,2}}^2 - 8\,g_{1,1}{g_{2,2}}^3g_{1,2}g_{2,1} - 6\,{g_{1,1}}^2{g_{2,2}}^2{g_{1,2}}^2 - {g_{1,2}}^4{g_{2,1}}^2 - {g_{1,1}}^2{g_{2,2}}^4 - 8\,g_{1,1}g_{2,1}g_{2,2}{g_{1,2}}^3,\\
-1 &= -8\,{g_{1,1}}^3g_{2,1}g_{2,2}g_{1,2} - 6\,{g_{1,1}}^2{g_{2,2}}^2{g_{2,1}}^2 - 6\,{g_{1,2}}^2{g_{2,1}}^2{g_{1,1}}^2 - {g_{1,2}}^2{g_{2,1}}^4 - {g_{1,1}}^4{g_{2,2}}^2 - 8\,g_{1,2}{g_{2,1}}^3g_{1,1}g_{2,2},\\
20 &= -20\,{g_{1,2}}^2 - 20\,{g_{2,2}}^2, 4 = 4\,{g_{2,2}}^4 + 4\,{g_{1,2}}^4 + 9\,{g_{1,2}}^2{g_{2,2}}^2.
\end{aligned} \tag{4}$$

Note, we are looking for symmetric decomposition and, therefore, are interested in reflections groups only. According to [12] the following condition must be satisfied for any reflection transformation:

$$g_{1,1}g_{2,2} - g_{2,1}g_{1,2} = -1 \tag{5}$$

The system of equations (4), (5) has the solutions

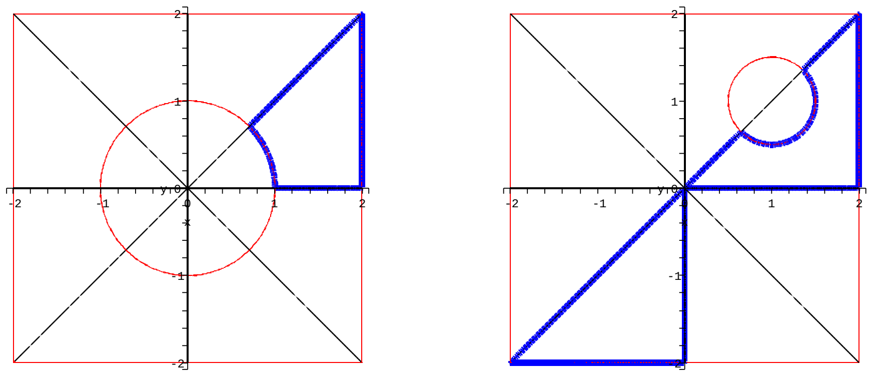


Fig. 3. Decomposition of $Rect(x,y)\backslash circle(x,y)$ and $Rect(x,y)\backslash(circle(x-1/2,y-1/2)+3/4)$

$$\{g_{2,2}=0,g_{1,2}=1,g_{2,1}=1,g_{1,1}=0\},\{g_{2,2}=0,g_{1,1}=0,g_{1,2}=-1,g_{2,1}=-1\},$$
$$\{g_{1,2}=0,g_{2,2}=1,g_{2,1}=0,g_{1,1}=-1\},\{g_{1,2}=0,g_{2,1}=0,g_{2,2}=-1,g_{1,1}=1\},$$

which correspond to the reflection of part R_4 of G_4 given by:

$$R_4=\left\{\begin{bmatrix}0&-1\\-1&0\end{bmatrix},\begin{bmatrix}0&1\\1&0\end{bmatrix},\begin{bmatrix}1&0\\0&-1\end{bmatrix},\begin{bmatrix}-1&0\\0&1\end{bmatrix}\right\}.$$

Note that R_4 does not satisfy the closure property and, therefore, is not a group.

Obviously, the symmetry axes are given by those eigenvectors of these matrices, which correspond to the eigenvalue 1:

$$\left\{\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}1\\0\end{bmatrix},\begin{bmatrix}1\\1\end{bmatrix},\begin{bmatrix}-1\\1\end{bmatrix}\right\}.$$

Solving $\mathbf{R}_i\mathbf{x}^T=\mathbf{x}^T$ we obtain four symmetry lines:

$$\begin{aligned}l_1(x,y)&=x,\\l_2(x,y)&=y,\\l_3(x,y)&=x-y,\\l_4(x,y)&=x+y.\end{aligned}\tag{6}$$

As shown in Fig. 3, four lines (6) decompose the initial domain $O(x,y)$ given by

$$O(x,y)=Rect(x,y)\backslash circle(x,y)=Rect(x,y)-circle(x,y)-\sqrt{Rect(x,y)^2+circle(x,y)^2}$$

in 8 congruent parts $O_i(x,y)$, which can be obtained using R-intersection (2) of O(x,y) and eight halfspaces given by 4 lines (6) as follows:

$$\begin{aligned}O_1(x,y)&=O(x,y)\cap l_2(x,y)\cap l_3(x,y)\\O_2(x,y)&=O(x,y)\cap l_1(x,y)\cap -l_3(x,y)\\O_3(x,y)&=O(x,y)\cap -l_1(x,y)\cap l_4(x,y)\\O_4(x,y)&=O(x,y)\cap l_2(x,y)\cap -l_4(x,y)\\O_5(x,y)&=O(x,y)\cap -l_2(x,y)\cap -l_3(x,y)\\O_6(x,y)&=O(x,y)\cap -l_1(x,y)\cap l_3(x,y)\\O_7(x,y)&=O(x,y)\cap l_1(x,y)\cap -l_4(x,y)\\O_8(x,y)&=O(x,y)\cap -l_2(x,y)\cap l_4(x,y)\end{aligned}\tag{7}$$

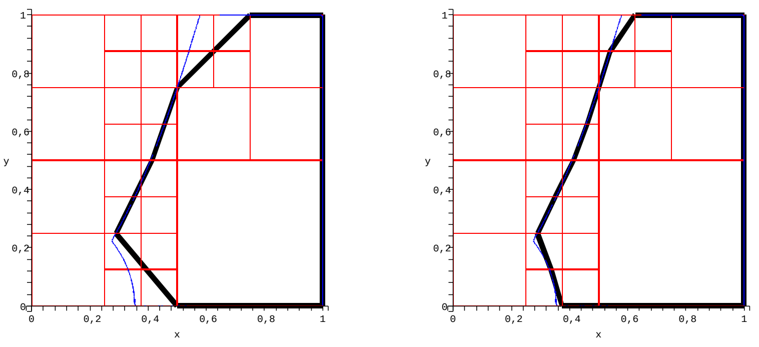


Fig. 4. Curve approximation with quadtrees

In this way the finite symmetry group of simple geometric regions given as roots of polynomial equalities can be calculated. The decomposition of the region shown in Fig. 2 can be derived using symmetry axes of such primitive regions in the same way.

4 Boundary Discretization Using Quadtrees

As shown in section 2, the implicit curves can be used to describe complex geometric regions. In order to perform the advancing front triangulation needed for the FEM calculations on such regions enclosed by $R(x, y) = 0$ we need to partition the curve $R(x, y) = 0$ into linear segments. The term *quadtree* (or *octtree* in 3-dimensional case) is used to describe a well-known class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space [10]. As shown in Fig. 4 we start with the root rectangular element enclosing the geometric region of interest and subdivide it successively into four equal-sized quadrants. Each of these quadrants can be entirely contained in the region ($R(x, y) > 0$), entirely disjoint from it ($R(x, y) < 0$) or crossed by the boundary curve ($R(x, y)$ changes the sign along some quadrant edge). Checking the sign of $R(x, y)$ in the quadrant nodes one can determine the edges wich are crossed by the boundary curve and approximate the curve in the particular quadrant as shown in Fig. 4. The boundary quadrants can be successively subdivided to achieve a better approximation of the region boundary.

We have implemented a package `SpaceTrees` for Maple, that provides the following features:

– generating and refinement of quadtrees
– performing the set operations on quadtrees (union, intersection, difference)
– generating the discretization for numerical methods: both initial front for advancing front method described bellow and rectangular elements

This package is implemented in object-oriented way as described in [1]. For example, the following command generates the quadtree with top left corner with coordinates `0 , 1` and widths `1, 1` in x- and y-direction:

```
[> root:=quadtree(0, 1, 1, 1, O(x,y));
```

$O(x, y) = 0$ is the implicit function that bounds the region to be partitioned.

To refine the generated quadtree the method `refine` can be invoked:

```
[> ‘‘||root||refine();
```

After several refinements one obtains the result shown in Fig. 4.

To obtain the approximation of $O(x, y) = 0$ corresponding to a particular depth of the quadtree use:

```
[> ‘‘||root||getBoundary(depth_level);
```

The line segments generated in this way approximate boundary and are now used as initial front for the advancing front triangulation method described bellow.

5 Advancing Front Triangulation

This method [8] starts with the initial front $\mathcal{AF}$ obtained in the previous section. Then, it adds triangles into the domain, with at least one edge on the front. At each step, this will update the front. When the front is empty, the mesh generation is completed. This requires that the domain be bounded, but for unbounded domain the front can be advanced until it is at some large distance from the object. As the algorithm progresses, the front will advance to fill the remainder of the area with triangles.

In Fig. 5 the algorithm that we use is shown in more detail. Let three sets be given:

$\mathcal{AF}$	– current advancing front, consisting of edges
$\mathcal{V}$	– the set of all triangulation vertices
$\mathcal{E}$	– the set of oriented triangulation edges
$E(a,b)$	– the edge connecting vertices a and b

For each edge $E = (a,b) \in \mathcal{AF}$ of the front the algorithm calculates candidate vertex v lying in the vertex of an equilateral triangle with the base E. The triangles can be stretched by the user defined parameter $\delta(x,y)$. Adapting $\delta(x,y)$ the size of triangles can be adapted through the region.

`compute_next_candidate_vertex(`E,δ`)`	- returns the point lying in the vertex of an equilateral triangle with the base E, to the left from E at the distance $dist(x,y)$
$\delta(x,y)$	- determines the stretching factor

Before new candidate edges (a,v), (v,b) are inserted in the current triangulation, we perform the intersection tests with existing edges using the procedure `visible`. Furthermore we calculate the minimal distance and minimal angle between (a,v), (v,b) and existing triangulation edges using `min_distance`, `min_angle`:

`visible(E::edge, v::vertex, s::set)`	– tests, wether the generated edge is crossed by any other edge of the set `s`
`min_distance(s::set)`	– computes the minimal distance between points of the set `s`
`min_angle(s::set)`	– computes the minimal distance between edges of the set `s`

If (a,v), (v,b) does not intersect any other edge and minimal distance and angle condition are not violated, they will be added to the triangulation. If this is not the case, the next candidate vertex will be chosen from the existing triangulation vertices $\mathcal{V}$ using `find_nearst_vertex`:

`find_nearest_vertex(s::set, v::vertex)`	– finds the vertex in `s` nearest to v

```
while $\mathcal{AF} \neq \emptyset$ do
        $v$ := compute_next_candidate_vertex ($E(a,b) \in \mathcal{AF}, \delta(x,y)$)
        while not visible($E(a,b), v$) or min_angle($\mathcal{E} \cup (a,v) \cup (v,b)$ ) $\leq \theta_{min}$
        or min_distance( $\mathcal{V} \cup v$ ) ) $\leq l_{min}$ do
        $v$ := find_nearest_vertex ($\mathcal{V}, v$)
        od:
         $\mathcal{E} := \mathcal{E} \cup \{(a,v),(v,b)\}$
        $\mathcal{V} := \mathcal{V} \cup \{v\}$
        $\mathcal{AF} := \mathcal{AF} \cup \{(a,v),(v,b)\} - \{(a,b)\}$
         od:
```

Fig. 5. A quasi-Maple description of the basic structure of advancing front algorithm

In this way the triangulation result depends on the choice of the following parameters:

l_{min}	– the minimum distance allowed between vertices
θ_{min}	– the minimum angle allowed between edges
$\delta(x,y)$	– stretching parameter

Compare, for example, grids obtained with $\delta(x,y) = 1$ (on the left hand side in Fig. 7) and δ given by

$$\begin{cases} 2\,x + \frac{17}{8} & x \le -\frac{13}{16} \\ 1 & \text{otherwise} \end{cases}$$

(on the right hand side in Fig. 7) In Figs. 6 and 8 the different grids for different choices of these parameters are shown. For two parts of the region of Fig. 1 their triangulation as well as composite grid for the entire region are depicted in Fig. 8. In this case there are six symmetric subregions according to Fig. 2, but the advancing front triangulation is executed only in two of them as shown in Fig. 8. The filling of the remaining symmetric counterpart subregions by a grid is a mere reflection, thus, no costly operations of the advancing front triangulation are performed at this reflection. Therefore, we can neglect the CPU time needed for these reflections. As a result, we obtain for the region of Fig. 1 the speed-up factor of 6/2 =3. For another region shown in Fig. 9 the speed-up factor is obviously equal to 8.

6 FEM Computation

Consider the boundary value problem in the region Ω with the boundary Γ:

$$u(x,y)_{xx} + u(x,y)_{yy} = f(x,y), \quad u(\Gamma) = 0. \tag{8}$$

In order to solve this equation using the finite element method, one has to minimize the following energy functional:

$$E(v) = \int_\Omega\int \left[\frac{1}{2}(v(x,y)_x^2 + v(x,y)_y^2) - v(x,y)f(x,y)\right] d\Omega$$

over a certain functional space X.

The finite approximation $\tilde{u}$ is assumed to be of the following form:

$$\tilde{u}(x,y) = \sum_{i=1}^{N} c_i\phi_i(x,y), \tag{9}$$

where $\phi_i(x,y)$ are the so-called Ansatz functions and c_i the unknown coefficients to be determined.

According to the Ritz–Galerkin approach, the c_i's can be computed by solving the algebraic system $A\overrightarrow{c} = \overrightarrow{b}$, where A is the stiffness matrix and $\overrightarrow{b}$ is the load vector given by

$$A(i,j) = \int_\Omega\int \left[\frac{\partial}{\partial x}\phi_i(x,y)\frac{\partial}{\partial x}\phi_j(x,y) + \frac{\partial}{\partial y}\phi_i(x,y)\frac{\partial}{\partial y}\phi_j(x,y)\right] d\Omega,$$

$$b(i) = \int_\Omega\int \phi_i(x,y)f(x,y)d\Omega.$$

Each Ansatz function ϕ_i is defined in some particular triangular element and is required to take the value 1 in one of the element nodes and vanish in all other nodes.

In [1] the Maple based FEM solver has been presented. This solver has been extended in the present work in order to support unstructured triangular grids. Our FEM package provides two main classes: `FEElement` and `FENode`. For each triangle generated by advancing front algorithm the corresponding grid nodes and finite element can be generated by calling the object constructor `FEElement` with the triangle vertices as parameters:

```
[> fe:=FEElement([0,0],[1,1],[5,0]):
```

The class `FEElement` provides following methods:

- `getLocalStiffnessMatrix` - computes the stiffness matrix of the element
- `getLocalLoadVector` - computes the load vector of the element
- `getGlobalStiffnessMatrix` - perform the assembling of all exisitng element matrices

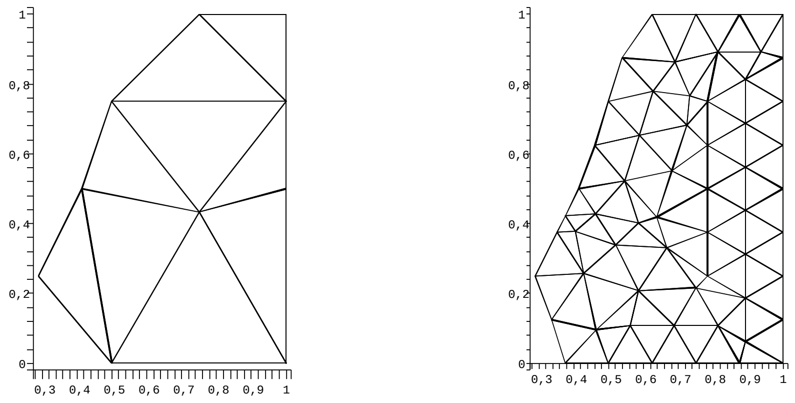


Fig. 6. Advanci ξ (compare with Fig. 4)

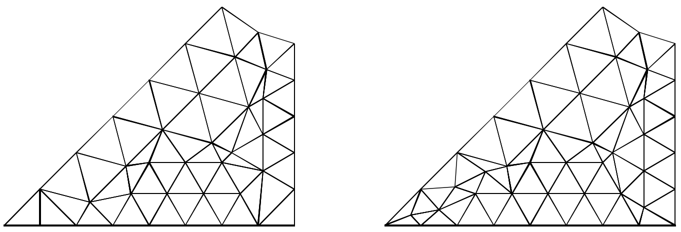
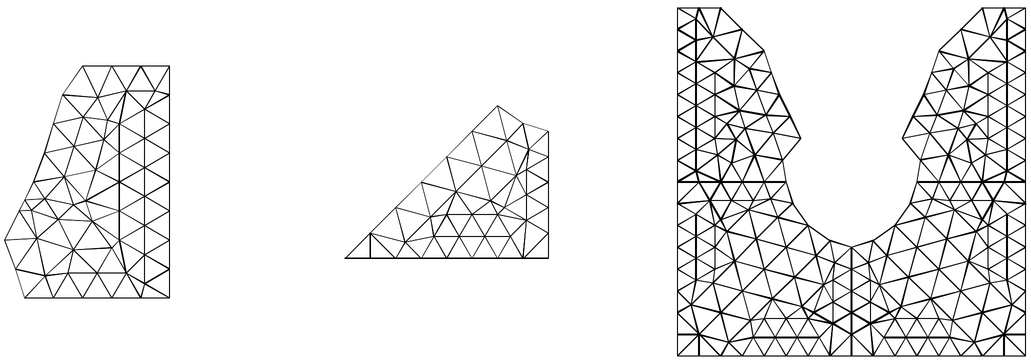


Fig. 7. The triangulation of parts of our region obtained with different $\delta(x, y)$



(a) (b) (c)

Fig. 8. The triangulation of parts of our region obtained with $\delta = 1, l_{min} = 0.3, \theta_{min} = 0.6$ (a), (b) and the derived triangulation of the whole region (c).

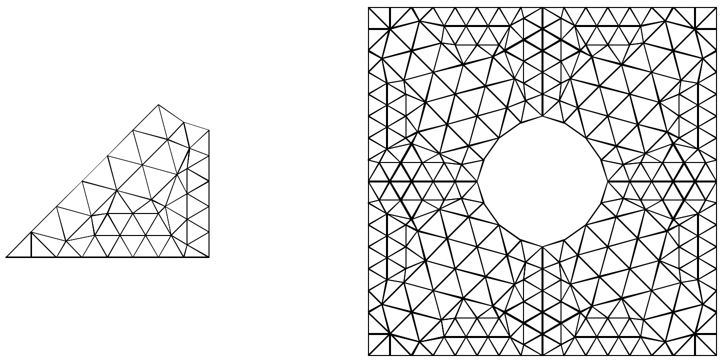


Fig. 9. Decomposing of a channel with a cylinder leads to speed-up factor 8.

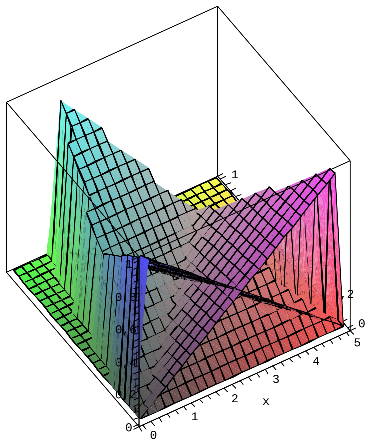


Fig. 10. An example: Ansatz functions of the element

– `getGlobalLoadVector` - perform the assembling of all exisitng load vectors
– `getElementAnsatzFunctions` - returns the Ansatz functions defined in the element

Using, for example, the method `getElementAnsatzFunctions` the Ansatz functions, which belong to the particular element can be plotted (see Fig. 10).
The invocation of both methods `getLocalStiffnessMatrix` and `getLocalLoadVector`

```
[> ``||fe||getLocalStiffnessMatrix();
[> ``||fe||getLocalLoadVector();
```

yields:

$$\begin{bmatrix} \frac{17}{10} & -2 & 3/10 \\ -2 & 5/2 & -1/2 \\ 3/10 & -1/2 & 1/5 \end{bmatrix}$$

$$\begin{bmatrix} \frac{120125}{32} \\ \frac{438895}{192} \\ \frac{1160875}{192} \end{bmatrix}$$

Each object of the class `FEElement` generates three objects of the class `FENode`, which consist of the following data fields:

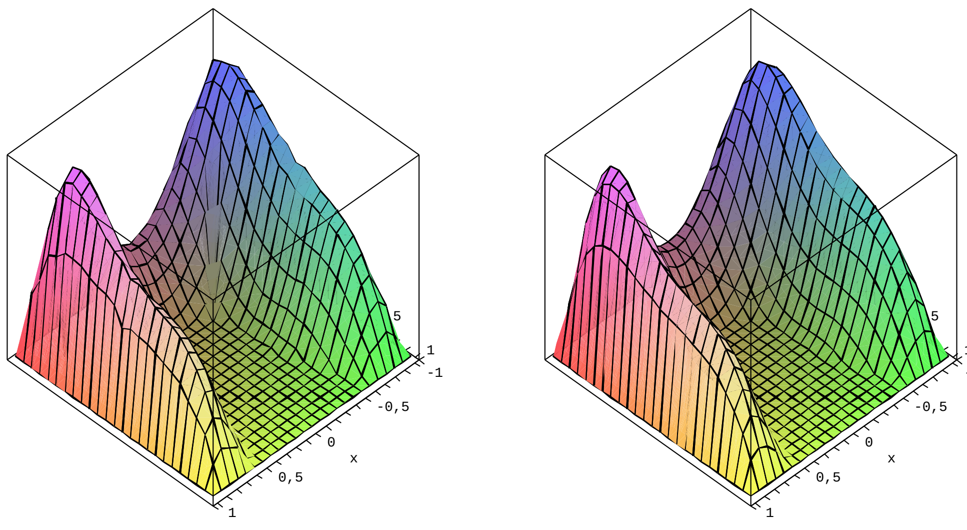


Fig. 11. An example: computed and exact solutions of (8)

- **phis** – Ansatz functions, which take the value 1 in the node
- **ci** – the unknown coefficient in the linear combination (9) corresponding to the node

The **ci** may be, for example, addressed by the user in order to provide boundary conditions. As described in [1] the spatial coordinates can be used to address data objects, for example, with the help of the functions **getAttribut**, **setAttribut** provided by our package:

```
[> getAttribut(FENode, 0, 0, ci);
```

or

```
[> setAttributName(FENode, 0, 0, ci, 0);
```

Assembling of local element matrices is performed using the methods **getGlobalStiffnessMatrix**, **getGlobalStiffnessMatrix**.

In order to provide the more complicated example we solve (8) using grid shown in Fig. 8. Let f be given by:

$$f(x,y) = 3 - 189*x^2 - 99*y*x^2 + 450*x^2*y^2 - 27*y^2 + 411*x^4 + 24*y^4 - 97*y^3 + 8*y^5 - 24*x^6 + 128*x^2*y^3 - 504*x^4*y^2 - 144*y^4*x^2 + 24*x^4*y + 36*y$$

Then the exact solution is

$$u(x,y) = 4\,(x-1)\,(x+1)\,(y+1)\,(y-1)\,\left(-x^2 - y^2 + 1/8\right)\,\left(y - 3\,x^2\right).$$

to obtain the result shown in Fig. 11.

We show in Fig. 12 the error $|u_{FEM} - u_{exact}|$ of the FEM solution for $N = 17$ (Fig. 12, (a)) and for $N = 59$ (Fig. 12, (b)). The error can be seen to drop with increasing N.

7 Conclusion and Future Work

In the present paper the advancing front triangulation on geometric regions given as implicit functions has been considered. In order to perform the triangulation we, at first, discretize the boundary of the region with the aid of octal trees. We have proposed to use symmetry properties in order to perform triangulation in much more efficient way. For this purpose we have presented an algebraic technique, which allows one to perform the decomposition of the geometric domain by computing finite reflection groups of the domain described as implicit R-Functions. Such a decomposition of the domain in symmetric

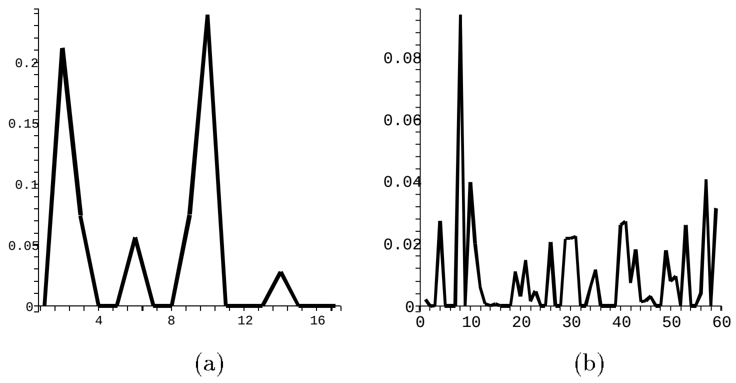


Fig. 12. $|u_{FEM} - u_{exact}|$ of the FEM solution for $N = 21$ (a) and for $N = 91$ (b)

parts leads to CPU time savings by factor from 3 to 8 in order to perform triangulation of the given region.

As shown in this paper computer algebra systems, such as, for example, Maple are powerful tools in order to handle algebraic functions applied, calculate the domain decomposition into symmetric parts by solving algebraic equations, and perform Finite Element analysis, but unfortunately too "slow" in order to deal with real-life applications.

Therefore, the future work in this field may concern the automatic generation of numerical code in the real programming languages, such as C or Fortran (as we have proposed in [2], [3]).

References

1. Chibisov, D., Ganzha, V., Zenger, Chr.: Object oriented FEM calculations using Maple, Selçuk Journal of Applied Mathematics **4** (2003) 58–86
2. Ganzha, V., Chibisov, D., Vorozhtsov, E.V.: GROOME – tool supported graphical object oriented modeling for computer algebra and scientific computing. In: Computer Algebra in Scientific Computing (CASC 2001), V.G. Ganzha, E.W. Mayr, E.V. Vorozhtsov (eds.), Springer-Verlag, Berlin (2001) 213–232
3. Ganzha, V., Chibisov, D., Vorozhtsov, E.: Problem Solving for Scientific Computing: Data Modelling Instead of Algorithms ?, Selçuk Journal of Applied Mathematics **2** (2001) 53–72
4. Requicha, A.: Representations for Rigid Solids: Theory, Methods, and Systems, ACM Computing Surveys (CSUR)archive, Volume 12 , Issue 4 (December 1980)
5. Hoffmann, Chr.: Implicit curves and surfaces in CAGD, IEEE Computer Graphics and Applications, 1993
6. Sederberg, T.W.: Implicit and Parametric Curves and Surfaces for Computera-Aided Geometric Desgin, PhD Thesis, Purdue University,1983
7. Rvachov, V.L.: Methods of Logic Algebra in Mathematical Physics, Naukova Dumka, Kiev (1974)
8. George, P.L.: Automatic Mesh Generation. Application to Finite Element Method, John Wiley & Sons, New York (1991).
9. Shapiro, V.: Theory and Applications of R-Functions: A primer, Technical Report, Cornel University, 1991
10. Samet, H.: The Design and Analysis of Spatial Data Structures, Addison-Wesley, 1990
11. Cox, D.A., Little, J.B., O'Shea D.: Ideals, Varieties, and Algorithms, Springer-Verlag, 1996
12. Groove, L.C., Benson, C.T.: Finite Reflection Groups, Springer-Verlag, 1985