Contravariant Geometry Description

Ilja Schmelzer

November 25, 1997

Abstract

We introduce a new type of geometry description, dual to the standard way of geometry description. It allows to describe and modify very complex geometries in a simple way.

Contents

1 Introduction

A prerequisite for mesh generation is the availability of a *geometry description*. This description usually defines the boundary of the computational domain. But, this domain often consists of different *regions* with different material properties. In this case, also inner boundaries have to be defined. These boundaries also often have to be subdivided into different *boundary faces* with different boundary conditions. These boundary faces, again, have boundaries, and so on. In principle, the geometry description has to define all these regions and boundary parts.

The most interesting application of geometry descriptions is the three-dimensional space. Often 2D and 1D simplifications will be used. But there are also interesting higher dimensional applications: 4D for space-time, 6D for the phase space, 7D for the phase space in time. Thus, it makes sense to consider the general, n-dimensional problem.

The purpose of this paper is to define a new type of geometry description called *contravariant geometry description* or shortly *cogeometry*. The notion cogeometry we have formed in analogy to the pair cohomology — homology. Indeed, our cogeometry is a variant of the dual construction of the standard geometry description by a cell complex.

It allows to create and modify even very complex geometrical configurations in an easy and natural way, in principle for arbitrary dimension. It was successfully implemented in the 3D geometry description package IBGD which is part of the grid generator IBG.

The current situation in 3D geometry modeling was summarized in [3] in the following words: "The state-of-the-art in commercial geometric modeling technology is solid modeling. Topologically, a solid model is a two-manifold object. [...] There is a growing awareness in the CAD/CAM/CAE community of the importance of providing systems which can model and represent non-manifold objects. Unfortunately, this functionality is not yet a commercial reality." Here, "non-manifold object" refers to a geometry with inner boundaries, boundary lines and so on.

The geometry description proposed in this (and other like [4]) papers is based on a separate "topology description" developed by Weiler [5] which lists all regions and boundaries and the neighbourhood relations between them, and "geometric entity data" which are usually functions from the basic boundary entities into the space. They are usually taken from some special function space, the current favorite seems to be NURBS (non-uniform rational B-splines). See also [8], [10], [9]. From pure mathematical point of view, this may be considered as an implementation of a 3D cell complex.

Contrary, our contravariant geometry description consists of functions F_k which find intersections of k-dimensional simplices with segments of the related codimensions — one function for each dimension. The first function F_0 is simply a function which defines the region containing a given point, the second allows to find intersections of an edge with boundary faces and so on.

The major difference between these two types of geometry description is their functional behaviour. Assume we have a smooth mapping $f : X \rightarrow Y$ and a geometry on Y. In general, this allows to define an *induced* geometry on X, defining the regions on X as the pre-images of the regions on Y. In the other direction, there is no such natural possibility. A geometry on X in general does not allow to define a natural geometry on Y.

Considering the behaviour of the standard geometry description, we see that a cell complex on X defines a natural cell complex on Y. But this is the wrong direction, the cell complex on Y may not be used in general to define a geometry. Our cogemetry has the correct transformation behaviour. A cogeometry on Y allows to define the induced cogeometry on X.

There are a lot of interesting possibilities to create or modify geometries which may be considered as special applications of induced geometries:

- Higher dimensional extension of a lower dimensional geometry.
- Restriction of a higher-dimensional geometry to a surface.
- Intersection of different geometries on a given space.

- Boundary description by an equation $f(x) = 0$.

See [11], [12] about some problems which occur using standard geometry descriptions. In the cogeometry it becomes much easier to define these operations. We consider algorithms which may be used to define cogeometries:

- the "default" for the functions F_k if the related boundary has no non-trivial subdivision into different segments.
- the geometry induced by a mapping.
- the intersection of geometries.
- the geometry described by the standard form — by a cell complex.

Thus, there are algorithms for almost every input possibility. These algorithms are fast enough to be used in complex grid generation algorithms. We also have a "fast prototyping" strategy which allows to implement a complicate geometry step by step.

The cogeometry allows also an easy handling of attributes — application-dependent data which describe the properties of the segments and functions defined on these segments. The functions F_k may be interpreted as methods of the related class "cogeometry", that means, the concept is very natural from object-oriented point of view.

Some parts of our dual concept have already been used. The idea of the first of our intersection functions F_0 which returns the region containing a given point is very simple, and often this information is the only available. That's why in many applications this function will be used to describe the geometry. If necessary, the boundary position will be approximated by a simple iteration. But in this way it is not possible to describe the geometry of the boundary itself. Thus, usually such a geometry description will be considered only as a poor substitute of a complete geometry description.

To obtain a unique, modular interface for different geometry descriptions which use a lot of different types of elementary cells and mappings of these cells (spline types) it is a natural idea to use intersection functions instead of the explicit mapping functions. This idea was also already used for geometry description, for example in [4]. But, for the description of the topological neighbourhood relations, the classical cell-complex concept was used.

2 Definition of a Cogeometry

There are a lot of different technical realizations of the dual concept. So, a simple dualization leads to a function which returns for a given simplex and a segment of the related codimension their intersection index. Such a realization may be easier to use in theoretical considerations, but not for implementation. We try to find here a variant which allows easy implementation and usage.

2.1 The Continuous Case

At first, we introduce the definition of a cogeometry for exact real arithmetics.

The notion *in general* we use to distinguish between the "general situation" of "transverse intersection", which define an open, dense subset in an appropriate topology, and "degenerated situations". Related results and techniques you can find in [6]. We consider only the smooth case and don't try to establish the number of derivatives which is necessary.

There will be different possibilities to handle the degenerate cases. But this strategy is not relevant for the problems of implementation, because the problem of degenerate cases will be covered by another serious problem — the rounding errors. That's why we simply use the strategy to define the cogeometry only for the general case.

Let's now introduce the basic object. The required properties of these objects we define later.

A k-simplex is a smooth mapping from the standard k-dimensional reference simplex into X.

A side of a k-simplex is a (k-1)-simplex defined by the mapping of the related side of the reference simplex. To avoid exceptions we define the side of a 0-simplex as the empty object.

A k-flag consists of a point p (called position), a sequence of (k+1) segments (S_0,...,S_k) and k orthogonal directions (v_1,...,v_k). To avoid exceptions we define the flag also for k = -1 as the empty object.

An intersection of a k-simplex may be a k-flag with position in the simplex — an *inner intersection* — or a (k-1)-flag on a side of the simplex — a *boundary intersection*.

A cogeometry $G(X)$ is a sequence of functions F_k for k $\geq$ 0 so that:

The function F_k allows to find intersections of k-simplices. Input is a k-simplex and an inititial intersection of the simplex. The result is another intersection of the same simplex which we call the continuation of the first intersection through the simplex.

Before we define the properties required for these objects, let's define them for the case of an n-dimensional geometry described by a smooth cell complex. We consider only smooth simplices and general position.

In this case, the position of a k-flag is inside the segmentS_k and boundary point for all S_i with i $<$ k, S_i is part of the boundary of S_j for i $>$ j, the direction v_i is in p tangential to S_j for j $<$ i, orthogonal to S_j . It points into S_{i-1}.

For an inner intersection, we require not only that the position of the flag is inside the simplex, but also that the projections of the flag directions into the plane of the simplex define a non-degenerate volume. This allows to define an orientation of the intersection.

Now let's define the intersection function F_k. The input flag defines a point on some (k-1)-segment S_{k-1}. Consider the intersection of the k-simplex with this segment, especially the component containing the initial point. In the general situation we obtain a smooth 1-dimensional manifold, and the initial intersection is one of the two ends of this curve. The position of the return value is the second end of this curve. The related flag we obtain using the continuation of the flag along the curve. For degenerate cases we do not define the function.

This description may fail, if it is not possible to continue the flag because of a change of the neighbourhood relations in some intermediate point of the curve. To avoid this effect we have to require that such intermediate points must be part of some boundary of codimension k. For an arbitrary geometry, this may be obtained by further subdivision of the related boundaries into parts with identical neighbourhood relations.

Let's define now the properties of a cogeometry. The strategy we use to fix these properties is to find properties which are fulfilled for our example. Let's list at first the most obvious properties:

- At first, we have a list of "transversality and orthogonality conditions"

— the directions of a flag have to be orthogonal, their projection on the tangential plane of the simplex not degenerated.

- We have a symmetry in the definition of F_k. The output of a first call may be used as the input with the same simplex. Then the result has to be the input of the first call.
- The (k-1)-part of the list of segments of the two flags is identical.
- Usually the positions of the two flags are different. Only in the case of inner intersections, the position may be the same. But in this case the directions must be different.
- The result for a simplex may be derived from the results for smaller simplices obtained by subdivision of the initial simplex.

Obviously these properties make sense for every geometry. Especially the last allows to localize the problem: The geometry will be completely defined by the results of F_k for arbitrary small simplices.

To complete the definition, we have to add a local regularity condition which describes the local behaviour of the geometry. This may be a condition of the following type:

- For every point there is a small neighbourhood so that there is a diffeomorphism which allows to transfer the local situation into the linear reference situation.

2.2 The Codimension of a Cogeometry

The *codimension* of a cogeometry is the highest codimension of a segment of the cogeometry. For a cogeometry of codimension k it is not possible to define input values for F_l with l > k+1. So, such a cogeometry will be completely defined by the sequence of the F_l between 0 and k+1. Because of this simplification it is useful to have information about the codimension. We have the following obvious properties of the codimension:

- The codimension of a cogeometry in an n-dimensional space is $\leq$ n.
- The codimension of the induced cogeometry is the same as of the original.

- The codimension of the intersection of two cogeometries is $\leq$ the sum of the codimensions of these cogeometries.

2.3 Connection to Morse Theory

There is a natural connection between the cogeometry and Morse functions (see [7]):

Lemma 1 *A Morse function on a space X defines a cogeometry.*

Proof: Each segment of this cogeometry will be related to a singularity of the Morse function. The segment may be defined as the set of points so that the limit of the gradient flow is the related singularity. The codimension of the segment is obviously the index of the singularity. **qed.**

This connection shows that a cogeometry is a very natural object from mathematical point of view. It also shows that a cogeometry may be defined also for spaces of infinite dimension and can have infinite codimension. A space which allow to define a Morse function on it, allows also to define a cogeometry. This shows that the class of spaces which allow a cogeometry is greater than the class of spaces which allow a standard geometry description.

2.4 The Implementation in Finite Precision Arithmetics

Now let's consider some modifications of the concept for the continuous case which will be necessary or useful for an implementation of the concept.

2.4.1 Affine Simplices

Usually in applications we consider only the n-dimensional Euclidean space, so we have some well-defined global affine structure. To use only affine simplices makes the interface much more simpler to use. Instead of the definition of a mapping we have to define only the coordinates of the corners of the simplex.

This seems to be a restriction, but for a manifold without affine structure we can simply use the affine structure of some local coordinates. The usage of other coordinates does not influence the limit of arbitrary small simplices which defines the geometry.

2.4.2 Finite Distances Instead of Infinitesimal Directions

To define a flag, we have to define a sequence of segments and infinitesimal directions. In finite precision arithmetics, we use instead a sequence of points (p_k, p_{k-1},...,p_0) so that:

- Each point p_i is inside the segment S_i of the flag.
- The direction v_i is defined by the vector from p_i to p_{i-1}.
- The distance between the points is small: $|v_i|<\varepsilon$
- The position of the flag is the position of p_k.

The main reason are the functions which are discontinuous near the boundary. Their value on p_i can be used as boundary limits. Thus, we need no special handling for such discontinuous functions.

2.4.3 Rounding Error Handling

The greatest problem of rounding errors is connected with the degenerate cases which we have not considered in the previous considerations. If we have a degeneration f $=$ 0, it is not the problem what we do - the same as for f $>$ 0 or as for f $<$ 0. A problem occurs if a value which is really (in exact arithmetics) $>$ 0, but in finite precision arithmetics $<$ 0, or, much worse, different parts of the program use slightly different formulas which leads to different results.

Our way to avoid this is to make a small modification of the result if the exact result is in such a dangerous neighbourhood of a degeneration. The modification must be small enough compared with the required accuracy of boundary computation, but it must be big enough to avoid an incorrect classification if it will be used later as input.

Thus, if the required accuracy is big enough compared with the possible rounding errors, this technique allows to avoid fatal errors. It also does not require a special handling for the degenerated situations there the result is not defined in the exact, continuous case, because the "input" is always not degenerated.

Remark that the case of a degenerated simplex is not dangerous, if the side containing the input flag itself is not degenerated. There will be simply a smaller set of possible output — there may be no k-flag inside the simplex and no (k-1)-flag on degenerated sides.

2.4.4 Subdivision into Two Different Functions

For a theoretical consideration it looks very nice if we have only one function for every dimension. But in the real implementation it becomes easier two distinguish two functions:

- The first variant of F_k for the case of a (k-1)-flag as input.
- The other variant of F_k for the case of a k-flag as input.

The idea of the simplification is that for the first variant we implement only one special case — the flag on the first side of the simplex. This makes the implementation simpler and faster. For the other variant, we can use a default implementation:

> Subdivide the simplex into smaller simplices so that the k-flag lies on the border between the sub-simplices. Reinterpret this k-flag as a (k-1)-flag on this border. Use the first variant of F_k to find the continuation. While the continuation was found on the inner border between the two sub-simplices, we have to continue the search in the other sub-simplex.

2.4.5 Nonorthogonal Flag Directions

The orthogonality condition for the flag directions require a special consideration.

- The orthogonality may not be exactly fulfilled in finite precision arithmetics caused by rounding errors. Thus, in reality we will not have exactly orthogonal flag directions.
- There are algorithms which do not include the computation of the related orthogonal directions. Usually they allow to create only some set of directions with non-degenerated projection on the simplex plane.
- For a non-smooth boundary, it will not be possible to define the tangential and orthogonal directions required for the definition of a flag.

These problems may be solved using the convention that the directions must not be orthogonal, but only their projections have to be not degenerated. But in this case we obtain a new problem — the projection of the directions on the simplex plane may lead to an incorrect result for the orientation of the intersection. This problem may be solved by the following convention:

- The flag directions of the result must define a set of non-degenerated projection on the related simplex plane so that it's orientation coincides with the projections of the orthogonal flag directions.
- If a flag will be used as input, the plane of the simplex containing this flag must coincide (approximately) with the plane which has contained the flag as output.

2.4.6 C++ Interface and OOP

The contravariant geometry description is very natural from object-oriented point of view:

- The cogeometry is a class.
- The intersection functions F_k are the methods of this class.

This leads to a natural C++ implementation. The cogeometry will be defined as an abstract class, the functions F_k will be virtual methods. A concrete geometry description will be a derived class which contains all necessary data of the geometry.

For simplicity, in the implementation the functions F_0 and F_1 will be implemented in a slightly modified way, not as the specialisation of the general definition given here.

2.5 Attribute Handling

Necessary part of the description of a physical situation is the attribute description. An *attribute* may be an arbitrary application-dependent information. An attribute has a result type (scalar, vector, integer) and can depend on geometrical data (points, segments). The physical sense of the attribute is defined by the application and hidden from the geometry description. That's why it is a good idea to separate the geometry description and the attribute description whenever possible. For many types of attributes this is possible. But often there are attributes which defined on the segment and depending on the point of the segment (f.e. functions which are discontinuous on the boundary, or boundary concentrations).

Because of this deep interaction with the geometry description it is necessary to have a general scheme to manage such attributes. Let's consider now this interaction. At first, remark that we have also a natural functional behaviour for the attributes. Indeed, for the geometry induced by a mapping $f : X \rightarrow Y$ on X and for given attributes of the original geometry on Y related induced attributes may be defined in a natural way: The attribute value of a point in X is simply the attribute value of it's image in Y.

We manage such attributes using the following simple technique:

> We use a data type for the point which contains also the attribute values of the point. The function F_k has to compute also these attribute values for the output points.

In principle, this technique may be considered as a special case of an induced geometry: We have to consider the embedding of a lower-dimensional space into a higher-dimensional space defined by the graphic of the attribute functions. This interpretation shows that it is possible to combine this technique with other operations using the composition of mappings. For example, the attribute values of some geometry may be used to define a mapping which may be used to define an induced geometry.

Our scheme leads to a natural implementation for the interpolation of function values for a given grid. The functions F_k have to find the intersections of a simplex with the related boundary. Using our technique, this function also has to evaluate the function values. But this is a very natural place to interpolate the function values, because the majority of data we need for the interpolation of the function values in the grid (especially the element which contains the point) we need also if we have to find only the intersection point in the grid.

3 Algorithms for Contravariant Geometry Descriptions

In this part we consider different algorithms which allow to define cogeometries.

3.1 Simplex Subdivision

Assume we have an algorithm which works correctly only for small simplices. The subdivision algorithm allows to obtain the correct result also for greater simplices. It works so:

- If the simplex is smaller than ε, the given algorithm will be called.
- Else, the k-simplex will be subdivided into 2^k sub-simplices of the half size.
- It must be detected which simplex contains the initial flag. In this step, degenerate cases have to be handled if the initial flag is in a neighbourhood of a boundary between sub-simplices.
- Then a cycle over the sub-simplices has to be considered. For the given sub-simplex, the same algorithm will be called recursively. If the result is a k-flag inside the sub-simplex or a (k-1)-flag on the outer boundary, this result will be returned. Else, the result is a (k-1)-flag on a boundary between two sub-simplices. In this case, in the next step we consider the related neighbour sub-simplex and use the result flag of the previous step as input.

Is it possible to get an infinite loop in this algorithm? In the general case, there will be only a finite number of intersections of the related boundaries with the inner boundaries between the sub-simplices. So, an infinite loop may be only a cycle between a finite set of flags. If the initial algorithm is really symmetric, this is not possible. Thus, usually there will be no infinite loops, but degenerated situations and errors of the initial algorithm may lead to such infinite loops.

In a typical regular situation, the number of intersections with inner boundaries will be small. Thus, if the number of steps will be great, an erroneous infinite loop may be assumed. Thus, it seems useful to break the loop after a maximal number of step which may be not very large. The last (k-1)-flag after a break lies inside the simplex. The value may be returned as a k-flag on some artificial "error boundary". This allows to continue the computation.

Another idea for the error handling is to restart the computation with a temporary smaller value of ε. This seems useful, because an incorrect value for ε seems to be the most probable error.

3.2 The Default Function

Assume we have defined the first (k-1) functions F_i. Is there a default implementation which may be used for F_k?

There are two useful variants: The first creates a unique k-boundary on places where we have different (k-1)-boundaries. The second variant creates it also where the higher-dimensional parts of the flag change. In principle, only the second variant is consistent. The first violates the condition that the (k-1)-part of the list of segments of the input- and output-flag has to be identical. But the first variant may be used as a "fast prototype". It has the advantage, that their usage for F_k and F_{k+1} will not lead to any (k+1)-boundary, that means F_{k+2} must not be implemented.

The idea of the algorithm is straightforward: A search loop over the boundary of the simplex using the previously defined functionF_{k-1} until a continuation is found. If the continuation is "correct" (this correctness definition is the difference between the two variants), it will be returned. Else, further subdivision up to the required accuracy will be used to find a k-boundary intersection inside the simplex. For a simplex which is small enough, it's centre will be returned as this intersection.

If there are multiple intersections, this algorithm may work incorrect, for example without the required input-output symmetry.

3.3 The Induced Cogeometry

As we have already mentioned, if we have a smooth mapping $f : X \to Y$ and a cogeometry $G(Y)$ on Y, we can define on X an induced cogeometry $G(X)$. In the case of affine mapping, there is a straightforward algorithm. We have to call F_k for the image of the input on Y. For the resulting Y flag we define a related flag in X using the same barycentric coordinates.

For nonlinear mappings we can use the standard subdivision algorithm until the simples is so small that the affine algorithm may be used.

3.4 The Intersection of Geometries

The intersection of geometries is another example of a natural operation which is hard to implement for the standard geometry description but straightforward in the concept of cogeometry.

The general scheme will be analogical to the case of an induced geometry. We can use subdivision until the simplices are small enough to be approximated by the affine situation. To consider the affine situation for some fixed pair (k, i) is straightforward, but the implementation becomes complex for higher values of k and i. But, because for the first functions the implementation is more trivial, we have a useful fast prototyping strategy for implementation.

For some variants, a special implementation may be useful:

- In the case of *partial intersection*, only one "basic" segment of the first cogeometry and it's boundary will be subdivided.

  Remark that the basic segment is not necessarily a region. It may be also a boundary segment of arbitrary codimension.

- Using as the second geometry the geometry induced by a smooth "characteristic function" from X to the real line, we obtain a powerful variant of the intersection which is much simpler to implement because we can use the fact that the second geometry has codimension 1.

3.5 Using a Simplicial Grid

Consider now the algorithms which may be used if the geometry will be described by a simplicial grid. That means, we assume that for every codimension the segments are defined as the union of simplices of the related dimension.

The algorithm is straightforward. A lot of code to handle degenerate cases will be necessary in the implementation, but our general strategy to handle these cases is sufficient. The only problem is to find the related simplices in the simplex grid data structure. To make the related search fast enough, an appropriate data structure is necessary. Possible variants are a search octree or a grid with neighbourhood relations between the elements.

In the last case, a small modification of the interface of F_0 leads allows higher speed. If we allow to transfer the "nearest previously searched point", this point may be used as the start point for the search.

3.6 Using a Boundary Grid

The most usual way to describe a geometry — the boundary grid — is similar to the previous case, but does not contain simplices of codimension 0. Instead, we have the information about the left and the right region for segments of codimension 1. This requires the modification of the algorithm for F_0 and F_1. We can implement F_0 as a variant of F_1 with a fixed start point far away, and for F_1 make a search over all codimension 1 simplices which may have an intersection with the edge to find the first of these intersections. A search tree seems necessary to make the related algorithms fast enough. The main problem of such an intersection algorithm is how to handle rounding errors and degenerate intersections.

Another possibility is to add a grid in the regions, f.e. using Delaunay techniques, and to use the previously described algorithm.

Thus, in principle it is possible to implement a fast algorithm for a boundary grid, but this type of input may be considered as the "worst case" for the cogeometry.

3.7 Other Possibilities

There will be also other natural possibilities to create and modify cogeometries. They usually may be easily implemented, at least for the region function F_0. For example:

- The union of different segments of a cogeometry.
- Different manipulations of attribute values do not immediately change the cogeometry, but it is useful to implement them as operations on the cogeometry.

- If there is an order relation between the segments of each codimension, the minimum or maximum of different cogeometries may be defined.
- The usage of graphical input. There will be different possibilities:
  - defining different regions by the principle one color — one region.
  - consider the picture as a function into the "color space". Define the geometry as induced by some geometry in this color space.

The real power of the contravariant geometry description is that it allows to combine all these separate methods. A 3D cogeometry defined by a grid may be intersected with a 3D cogeometry induced by some mapping. Attributes may be used to define mappings. To switch between different dimensions is trivial. The elevation profile of a region may be used to define the 3D surface geometry of this region. This may be combined with other maps of this region to subdivide the surface into parts.

3.8 Time Requirements

Let's consider the question of time dependence of the previous algorithms. We consider a fixed geometry and, for simplicity, the following simple grid generation algorithm: We create a regular (rectangular) grid, call F_0 for every node, F_1 for every edge, F_2 for every side there we have found an intersection on the boundary, and so on.

The main result is that for all algorithms we have considered before the time requirement for geometry calls is linear in the number of nodes in the grid.

All what we need is some regularity assumption for the geometry which allows to consider it for simplices which are small enough as nearly affine. For the affine geometry and a sufficiently small simplex we can find for all these algorithms some straightforward constant estimate. For greater simplices, we can always use the standard subdivision algorithm. For the finer grid, the number of the calls of F_k for $k>1$ increases, but not so fast as the number of calls of F_0, and the dimension of the simplices also becomes smaller. Thus, the number of calls of the higher F_k becomes irrelevant.

To find a constant estimate for F_0 and for F_1 with small distances is usually not a problem.

It is interesting, that the same result we obtain also in a much worse situation: Assume, the geometry is also defined by a grid, and this grid has approximately the same node density as the grid we want to create. A typical example is the grid for the next time step in a time-dependent process, which is based on a slightly modified geometry of the previous time step. A neighbourhood search algorithm allows even in this situation a linear dependence of the number of nodes. A search tree technique leads to an additional logarithmic factor.

3.9 Topological Errors and Convexity

Another very interesting question for the cogemetry is the following:

> If we create a grid using the contravariant geometry description, is it possible to guarantee that there will be no topological errors?

The answer to this question seems to be the greatest problem of the contravariant geometry description:

> In general, it is not possible to avoid topological errors without having any additional information.

Indeed, there may be very thin subregions inside a region. If we find such a subregion depends on the grid density and accident. If the subregion is so thin that no point of the finest grid will be inside, we have no chance to detect that there is such a subregion.

Remark that this is a consequence of the possibility to define cogeometries with an infinite number of regions and other cogeometries with such infinite properties like Julia set's or the Mandelbrot set. Such geometries obviously have to be simplified by any finite grid.

On the other hand, this description shows that dangerous situations have some special structure. We can classify such dangerous subregions by a "codimension" which is simply the number of the "very thin directions". For codimension 1, we have a crack. For codimension 2 a channel, for the maximal codimension an enclave. There may be also segments of higher codimension which may be dangerous. They all have some common properties:

- They are small.

- Their environment is highly non-convex.

This leads to two strategies to avoid errors:

- Further refinement.
- To remove highly non-convex situations using artificial subdivision of the related non-convex segments.

The second strategy is universal:

Theorem 1 *There is a grid generation algorithm which allows to define the cogeometry in a given region without topological errors (in exact real arithmetics) for an arbitrary finite cogeometry consisting only of convex segments.*

The proof may be found in [2]. In reality, it is not necessary to subdivide all segments into approximately convex parts. Only small, very non-convex parts have to be modified. Usually, if all details of the geometry are coarse enough, it is not necessary to make such subdivisions.

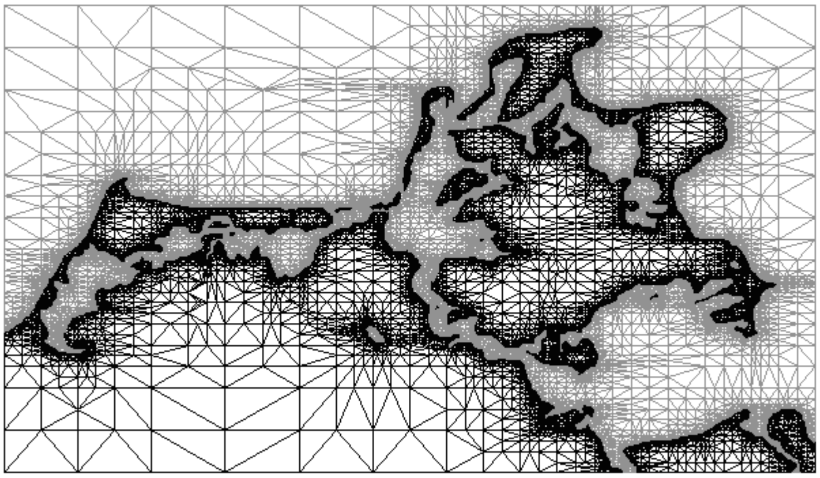
Subdivision of non-convex segments is also useful to help to avoid the "rounding" of sharp edges and corners.

4 Results

The idea of contravariant geometry description was used in the implementation of the "intersection-based geometry description" package IBGD which is part of the grid generation package IBG.

There are some differences between 3D package IBGD and the general, n-dimensional concept described here. Especially, in IBGD there are no flags and orthogonal directions, but only intersection points. But, even in this form, IBGD shows the advantages of the concept of contravariant geometry description and the possibility of implementation of the algorithms described here.

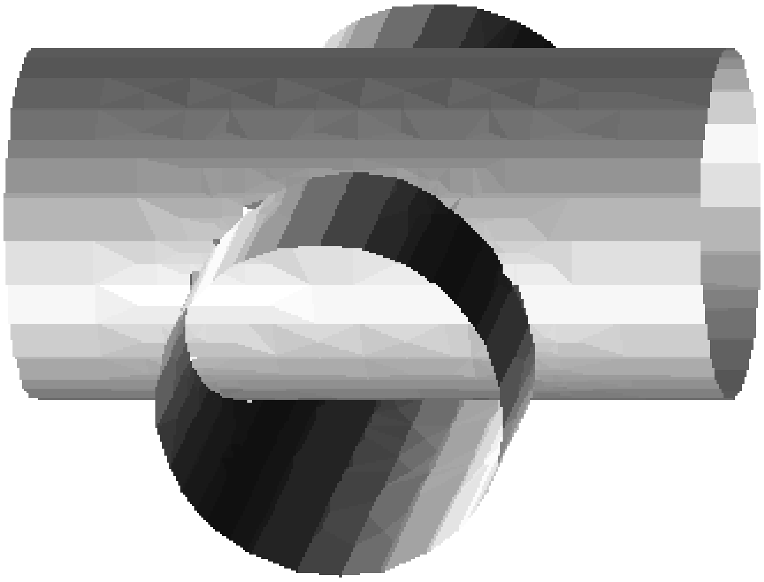
Let's consider some examples of grids created by IBG. The first example shows a two-dimensional mesh of the region around the island Rügen in Germany.



The island Rügen (Baltic coast of Germany)

The grid was created using a simple picture of the region with blue colors for the water and brown colors for land. To avoid topological errors, some artificial subdivision of water and land has been used.

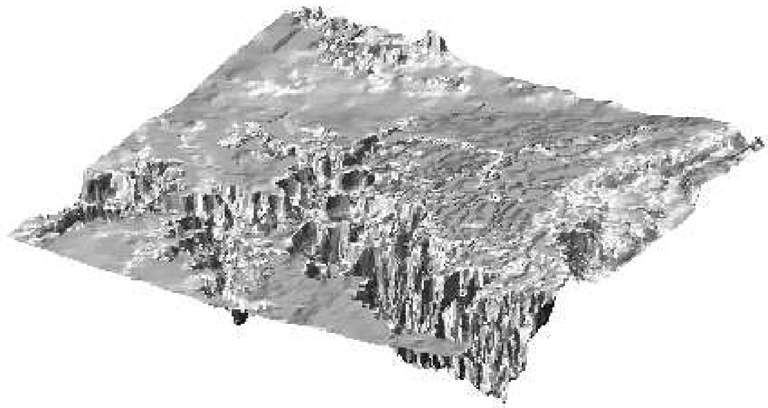
This example shows the intersection of two simple geometries:



Partial intersection of two simple geometries

The two parts have been described simply by their characteristic functions $f_1 = x^2 + (z - z_1)^2 - r_1^2$ and $f_2 = y^2 + (z - z_2)^2 - r_2^2$. To define the cogeometry it was not necessary to compute explicitly the intersection line.

It is also easy to include external data. Using elevation data obtained from the 1-degree USGS Digital Elevation Models we have created the following elevation profile:



Surface of the Grand Canyon (USA)

The related 3D cogeometry has been simply defined by the characteristic function $f(x, y, z) = z - e(x, y)$ where e is the elevation of the surface in the point (x, y).

References

[1] **Schmelzer I.** (1993), 3D anisotropic grid generation with intersection-based geometry interface *IMA Preprint Series Nr.1180, Univ. of Minnesota*

[2] **Schmelzer I.** (1995), Grid Generation with Contravariant Geometry Description.

PhD Thesis — to be published.

[3] **Saxena M., Finnigan P.M., Graichen C.M., Hathaway A.F. and Parthasarathy V.N.** (1995), Octree-based Automatic Mesh

Generation for Non-Manifold Domains *Engeneering with Computers nr.11, pp.1-14*

[4] **Shepard Shepard M.S., Finnigan P.M.** (1987), toward automatic model generation *SCOREC Report nr.9, Rensselaer Polytechnic Institute, Troy, New York, 1987*

[5] **Weiler K.J.** (1986), Topological structures for geometric modeling *PhD Dissertation, Department of Computer and Systems Engeneering, Rensselaer Polytechnic Institute, Troy, New York*

[6] **Hirsch M.W.** (1976), Differential Topology *Springer-Verlag NY, Heidelberg Berlin*

[7] **Milnor J.** (1963), Morse Theory *Princeton University Press, Princeton*

[8] **Barnhill R.E., ed.** (1992), Geometry Processing for Design and Manufactoring *edited by R.E. Barnhill. SIAM, Philadelphia*

[9] **Hagen H., ed.** (1992), Curve and Surface Design *SIAM, Philadelphia*

[10] **Hagen H., ed.** (1992), Topics in Surface Modelling *SIAM, Philadelphia*

[11] **Stoyanov Tz.E.** (1992), Marching along surface/surface intersection curves with an adaptive step length *Computer Aided Geometric Design 9 pp. 485-489*

[12] **Helmsen J.J., Scheckler E.W., Neureuther A.W., Sequin C.H.** (1992), An Efficient Loop Detection and Removal Algorithm for 3D Surface-based Lithography Simulation *In Proc. of NUPAD IV, pp.3-8*