Function-based shape modeling: mathematical framework and specialized language

Alexander Pasko[1] and Valery Adzhiev[2]

[1] Hosei University, Tokyo, Japan
pasko@k.hosei.ac.jp
[2] Bournemouth University, Poole, UK
vadzhiev@bournemouth.ac.uk

Abstract. In this survey, we describe the following different aspects of modeling multidimensional point sets (shapes) using real-valued functions of several variables: algebraic system as a formal framework; representation of shapes, operations, and relations using real-valued functions, internal representation of the modeling system; specialized language for function-based modeling, and model extension to point sets with attributes (hypervolumes).

1 Introduction

Description of a point set (shape) by a single real-valued function of point coordinates is a traditional problem of analytical geometry. Note that the direct problem of analytical geometry concerns the analysis of the shape for the given analytical expression. The most well studied shapes are algebraic surfaces given as zero sets of quadratic polynomials. The inverse problem of analytical geometry is to find an analytical description for the given shape. This problem statement can be extended to algorithmic definitions of functions and to multidimensional point sets. Thus, the subject of this paper is computer-aided modelling of multidimensional shapes using real-valued functions of several variables.

The idea of using a single real-valued function of three variables in computer-aided geometric modelling to define an arbitrary complex solid by a single continuous function of point coordinates and the object surface as a zero set of such a function (so-called *implicit surface*) has been exploited in solid modelling and in computer graphics (see Related works). There existed several research results on applying implicit surfaces to solve such important problems of computer-aided geometric design and animation as blending, offsetting, collision detection, and metamorphosis. However, by early 1990-s these models were not seriously considered in the area of solid modelling concentrated on such models as the boundary representation (BRep) based on parametric surface patches and Constructive Solid Geometry (CSG) based on set-theoretic operations [1]. The other separate area with similar models and algorithms was volume graphics using discrete scalar fields (voxel objects). A unifying represen-

tation was necessary to overcome the separation between relative models, to fully exploit the potential of the shape representation by a single function, and to extend it to time-dependent and other multidimensional shapes. In this survey, we describe the mathematical framework of the unifying function representation (FRep), give examples of some non-traditional primitives and operations, and describe the modelling system design including the internal representation and the specialized high-level modelling language.

2 Related works

The idea of using a single real-valued function of three variables in computer-aided geometric modelling to define an arbitrary constructive solid as $f(x, y, z) \geq 0$ and its surface as a zero set $f(x, y, z) = 0$ (so-called *implicit surface*) was expressed independently by Rvachev [2, 3, 4] and Ricci [5]. Both authors have introduced analytical expressions for set-theoretic operations. Ricci proposed using C^1 discontinuous min/max operations for exact descriptions and also approximate descriptions for getting smooth blending properties of the resulting surfaces. The work by Rvachev provided much more general approach called the theory of R-functions and introduced C^k continuous functions for the exact description of set-theoretic operations. We provide more details in the corresponding section.

The approach to modelling complex objects using a single real-valued function was exploited in computer graphics for describing skeleton-based implicit surfaces such as blobby models [6] and other models presented in the book [7]. The advantages of this approach are simple point membership classification, natural blending of shapes, conceptually simple algorithms for such problems as collision detection or metamorphosis. These advantages have motivated several research groups in solid modelling and computer graphics to do systematic research and to develop special software systems supporting this modelling paradigm.

In the solid modeling area, Constructive Solid Geometry (CSG) systems are traditionally based on solid primitives bounded by implicit surfaces. For example, the SVLIS modeling system [8] is based on CSG and allows for applying different algebraic operations to the defining functions of primitives. The only serious restriction is that such operations are not allowed on the level of set-theoretic solids, which means the single function representation is supported only on the level of primitives and their algebraic compositions. Applications of the theory of R-functions in different areas of solid modeling and mechanical design are considered by Shapiro [9, 10]. For example, the interactive system SAGE [11] is oriented towards solving numerical simulation problems on the base of the models built using R-functions and without finite-element mesh generation. The FRep model presented in this survey also has R-functions as the basic mathematical technique, which was used to generalize existing models and operations and to develop original ones. Following the introduction of FRep, in computer graphics, a BlobTree model was introduced in [12] to unify skeleton-based implicit models with CSG and global deformations. The models supported by SVLIS and BlobTree systems are subsets of FRep with the mentioned above restrictions al-

lowing for more efficient performing on some application operations such as polygonization or ray-casting. The HyperFun language [13] presented in this paper fully supports FRep and thus allows for the construction of more broader spectrum of functionally defined shapes.

3 Algebraic system

The geometric concepts of the function representation (FRep) [14, 15] can be presented as an algebraic system

$$(M, \Phi, W)$$

where M is a set of geometric objects, Φ is a set of geometric operations, and W is a set of relations for the set of objects. Here, we characterize the elements of the algebraic system, and the next section provides details on the representation of them using real-valued functions.

We consider geometric objects as closed subsets of n-dimensional Euclidean space E^n with the definition

$$f(x_1, x_2, \ldots, x_n) \geq 0$$

where f is a real continuous function defined on E^n. We call f a *defining function*. The inequality is called a *function representation* (or *F-rep*) of a geometric object. The function can be defined analytically, or with a function evaluation algorithm, or with tabulated values and an appropriate interpolation procedure. The major requirement to the function is to have at least C^0 continuity. The above inequality defines a closed n-dimensional object in E^n space with the following characteristics:

$f(X) > 0$ - for points inside the object;
$f(X) = 0$ - for points on the object's boundary;
$f(X) < 0$ - for points outside the object,

where $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ is a point in E^n. In the three-dimensional case, the boundary of such an object is usually so-called an "implicit surface". We should note that the use of the term "implicit surface" here can be considered a historical accident. The considered objects in 3D space are defined by explicit functions of three variables *f(x,y,z)* with zero-value isosurfaces *f(x,y,z)=0* as boundaries. This definition has nothing in common with implicit functions of two variables except the visual form of the equation *f(x,y,z)=0* used in the latter case to implicitly define, for example, *z* variable as a function of variables *x* and *y*.

Two major types of elements of the set M are simple geometric objects (primitives) and complex geometric objects. Each geometric primitive is described by a concrete type of a function chosen from the finite set of such types. A complex geometric object is a result of operations on primitives. In general, geometric objects defined by the above inequality are not regularized solids required in CSG. Applying operations to an object can result in zero values of the defining function not only on the boundary, but also inside the object. An object can also have a boundary with dangling portions that are not adjacent to the interior. If it is necessary to provide Frep for objects of lower

dimension in the given space, the main idea is that the function *f*(X) has to take zero values only at the points of this object and be negative everywhere else.

The set of geometric operations Φ includes unary, binary, and k-ary operations closed on the object representation:

$$\Phi_i : M^1 + M^2 + \ldots + M^n \rightarrow M$$

where *n* is a number of operands of an operation. The result of an operation is also an object from the set *M* that ensures the closure property of FRep. Let object G_1 have the definition $f_1(\mathbf{X}) \geq 0$. The term "unary operation" on the object G_1 means the operation $G_2 = \Phi_i(G_1)$ with the definition $f_2 = \Psi(f_1(\mathbf{X})) \geq 0$, where Ψ is a continuous real function of one variable. The binary operation on objects G_1 and G_2 means the operation $G_3 = \Phi_i(G_1, G_2)$ with the definition $f_3 = \Psi(f_1(\mathbf{X}), f_2(\mathbf{X})) \geq 0$, where Ψ is a continuous real function of two variables.

Relations can be considered subsets of the Cartesian product of the set *M* on itself. Relations are defined on the set of objects using predicates. For example, a binary relation is a subset of the set $M^2 = M \times M$. It can be defined by a predicate

$$S: M \times M \rightarrow I,$$

where *I* is a set of integer values corresponding to some k-valued logic.

4 FRep components

In this section, we discuss specifics of the algebraic system elements, namely, objects, operations, and relations, and provide detailed examples for some of them.

4.1 Objects

A primitive is considered a "black box" with the defining function given by a known function evaluation procedure. A complex object can be constructed by applying different operations to primitive objects. An FRep modeling system can support different types of primitives from simple to relatively complex ones:

- algebraic solids expressed in terms of polynomials (sphere, superellipsoid, torus, etc.);
- voxel (discrete scalar field) data with trilinear or higher order interpolation;
- skeleton-based “implicits”: blobby, soft, metaballs, and convolution objects [7, 16];
- solid noise and extruded noise [17, 18];
- two-dimensional polygons converted to FRep (see 3.1.1);
- bivariate parametric patches and trivariate parametric volumes (see 3.1.2);
- objects reconstructed from scattered surface points or from the series of cross-sections using radial-basis functions [19].

The possibility should be given to the system developer or the user to extend this set of primitives by providing an analytical or procedural description of a primitive.

This flexibility is one of the major advantages of an FRep based shape modeling system.

We give here examples of two types of primitives illustrating connections between FRep and other representations: "implicit" polygons illustrating boundary-to-function conversion in 2D case, and "implicit" curves and surfaces defined using parametric patches and volumes.

4.1.1 "Implicit" polygons

An arbitrary 2D polygon (convex or concave) can be represented by a real function *f(x,y)* taking zero value at polygon edges. The polygon-to-function conversion problem is stated as follows. A two-dimensional simple polygon is bounded by a finite set of segments. The segments are the edges and their extremes are the vertices of the polygon. A polygon is simple if there is no pair of nonadjacent edges sharing a point, and convex if its interior is a convex set. The polygon-to-function conversion algorithm should satisfy the following requirements:

- It should provide an exact polygon boundary description as the zero set of a real-valued function;
- No points with zero function value should exist inside or outside of the polygon;
- It should allow for the processing of any arbitrary simple polygon without any additional information.

Rvachev [3] proposed representing a concave polygon with a set-theoretic formula where each of the supporting half-planes appears exactly once and no additional half-plane is used. It is illustrated in Fig. 1. A counter-clockwise ordered sequence of coordinates of polygon vertices $A_1(x_1, y_1)$, $A_2(x_2, y_2)$,..., $A_n(x_n, y_n)$ serves as the input. Also, coordinates are assigned to a point $A_{n+1}(x_{n+1},y_{n+1})$ coincident with $A_1(x_1,y_1)$. It is obvious that the equation

$$f_i \equiv -x(y_{i+1} - y_i) + y(x_{i+1} - x_i) - x_{i+1}y_i + x_i y_{i+1} = 0$$

defines a line passing through the points $A_i(x_i,y_i)$ and $A_{i+1}(x_{i+1}, y_{i+1})$; f_i is a function positive in an open region Ω_i^+ and negative in an open region Ω_i^- located respectively to the left and to the right of the line. When the region Ω^+ is bounded by a convex polygon, it can be given by the logical formula

$$\Omega^+ = \Omega_1^+ \bigcap \Omega_2^+ \bigcap ... \bigcap \Omega_n^+ .$$

If Ω^+ is an external region of a convex polygon, then

$$\Omega^+ = \Omega_1^+ \bigcup \Omega_2^+ \bigcup ... \bigcup \Omega_n^+ .$$

Let us consider the concave polygon shown in Fig. 1a and a tree representing its monotone formula in Fig. 1b. Actually, the approach to the monotone formula construction is similar to the convex decomposition discussed in the previous section. Note that in the tree in Fig. 1b, the convex polygon $A_1A_2A_{10}A_{11}$ is a root (level 0), the polygon $A_2A_6A_{10}$ is level 1, and the polygons $A_3A_4A_5$ and $A_7A_8A_9$ are level 2. The internal region Ω^+ of the initial polygon is defined by the following formula:

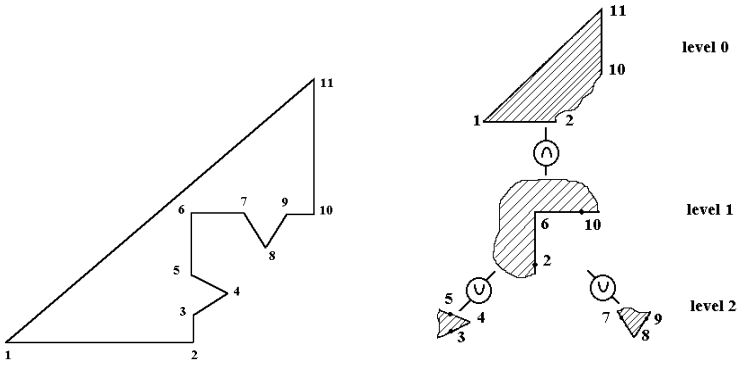


Fig. 1. Polygon-to-function conversion: a) concave polygon; b) tree structure representing the monotone set-theoretic formula.

$$\Omega^+ = \Omega_1^+ \cap \left(\Omega_2^+ \cup \left(\Omega_3^+ \cap \Omega_4^+\right) \cup \Omega_5^+ \cup \Omega_6^+ \cup \left(\Omega_7^+ \cap \Omega_8^+\right) \cup \Omega_9^+\right) \cap \Omega_{10}^+ \cap \Omega_{11}^+$$

This formula is especially nice in that each region is present in it only once. It is worth to emphasize that the set-theoretic operation applied to a region is determined by the tree level to which this region belongs. Because $\Omega_2^+ \equiv \Omega_5^+$ and $\Omega_6^+ \equiv \Omega_9^+$ (see Fig. 1a), it can be simplified as follows:

$$\Omega^+ = \Omega_1^+ \cap \left(\Omega_2^+ \cup \left(\Omega_3^+ \cap \Omega_4^+\right) \cup \Omega_6^+ \cup \left(\Omega_7^+ \cap \Omega_8^+\right)\right) \cap \Omega_{10}^+ \cap \Omega_{11}^+ .$$

The final formula for the defining function is obtained by replacing the symbols O_i^+ by f_i, symbol $\cap$ by $\wedge$ and symbol $\cup$ by $\vee$ in the monotone formula. For our example (Fig. 1a), the defining function for the polygon is

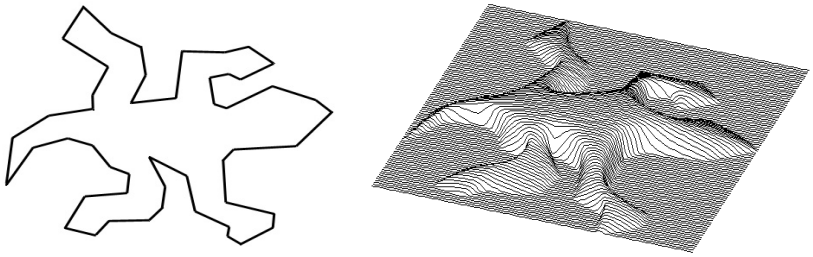
$$F = f_1 \wedge (f_2 \vee (f_3 \wedge f_4) \vee f_6 \vee (f_7 \wedge f_8)) \wedge f_{10} \wedge f_{11},$$

where symbols $\wedge$ and $\vee$ correspond to R-functions providing exact functional description for the results of intersection and union (see details in 4.2.1).

In practice, the monotone formula construction results in a tree structure (see Fig. 1b). To evaluate the single defining function for the entire polygon at a given point, the algorithm traces the tree from the leaves to the root, evaluates the defining functions of half-planes and applies the corresponding R-functions to them.

The conversion procedure is illustrated in Fig. 2 with an input concave polygon (Fig. 2a) and the defining function surface (Fig. 2b), where only positive values of the function (i.e., the points inside the polygon) are shown and the negative values outside the polygon are set to zero.

Once the defining function for the given polygon is obtained, it can be used as other 2D primitives, for example, for modeling 3D objects by sweeping. Note that the similar conversion procedure for an arbitrary 3D polyhedron is still an open research problem.



a b

Fig. 2. Example of an "implicit" polygon: a) initial concave polygon; b) depth data inside the polygon generated by the polygon-to-function conversion procedure.

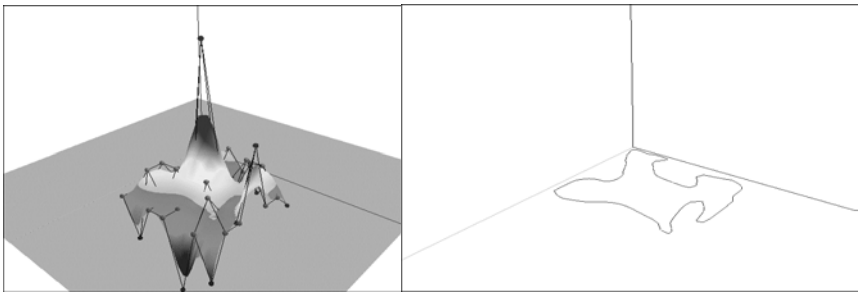


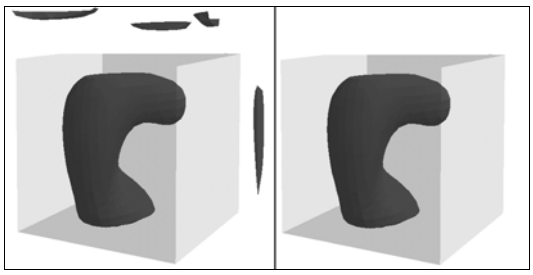
Fig. 3. Parametric function (B-spline) in the extended 3D space and the corresponding 2D solid in the 2D space.

4.1.2 Parametric patches and volumes

Here, we discuss modeling of "implicit" 2D curves and 3D surfaces and solids using bivariate and trivariate parametric splines [20, 21]. Let us consider modeling of a 2D solid as it is shown in Fig. 3. The surface S is a B-spline (parametric) function. It is defined in the (x, y, ξ) space, where $\xi = f(x, y)$. The 2D solid belongs to the (x, y) plane and is bounded by the zero-contour line of the surface with its inside part being the projection of the positive part of the 3D surface onto the plane. The surface is defined parametrically, then, by moving its control points vertically along the ξ axis, one can transform the corresponding 2D solid.

A similar approach can be applied to define and to deform 3D solids using trivariate parametric splines. A trivariate spline is defined with 3-dimensional control points, i.e. the x, y and z coordinates. We use the FRep definition, where a solid is defined by its defining function f with the inequality $f(x, y, z) \geq 0$, and its boundary is defined by the equality $f(x, y, z) = 0$. In the parameter space defined by $\{(x,y,z) : 0 \leq (x,y,z) \leq 1\}$, we assume that the x,y and z coordinates of the surface $S(u,v,w)$ can be expressed as a

regular grid, i.e. $P^x_{ijk} = \frac{i}{l}, P^y_{ijk} = \frac{j}{m}$ *and* $P^z_{ijk} = \frac{k}{n}$, where P is a control point of the spline, and l,m,n are the number of control points on each axis. We add one more dimension to those points, the ξ coordinate, corresponding to the function value. Then, we define a 3D solid as $f(x,y,z) = S^{\xi}(u,v,w)$. Hence, we can modify the 3D solid by changing the ξ coordinate of control points of the trivariate spline. Bezier splines were used in [20] and B-splines in [21].



a b

Fig. 4. Functional clipping for trivariate splines: (a) the unit cube, the desired solid inside it, and "ghost" solids outside; (b) the same solid after functional clipping

One undesirable property of Bezier splines and B-splines is that there is no control over the behaviour of the volume outside the unit cube domain. In order to overcome this, the functional clipping was introduced. Fig. 4a shows a solid based on a trivariate B-spline, which is bounded by the unit cube. The aim was to create a 3D solid inside that domain. As it can be observed, the behaviour of the parametric function outside the domain contradicts the requirement that there should be only negative values of a defining function for points outside the defined object. This results in undesirable ("ghost") solids. Let $S(u,v,w)$ be a function defining a trivariate spline where $S(u,v,w) = S^{\xi}(u,v,w)$. Consider the intersection between the unit cube and the 3D solid defined by the function:

$$S_{clip}(u,v,w) = S(u,v,w) \wedge_{\alpha} F_s(u,v,w)$$

where $F_s(u,v,w) = F_b(u)\ \&\ F_b(v)\ \&\ F_b(w)$ is a defining function of the unit cube with F_b defining the unit strip by each variable:

$$F_b(t) = (1-t)t\,,$$

and $\wedge_{\alpha}$ is the symbol of the intersection operation defined by an R-function (see details in 3.2.1).

The result of this procedure is illustrated in Fig. 4b. The defining function is negative outside the domain (no "ghosts") and the desirable 3D solid is unchanged. The procedures defining spline based objects can encapsulate the functional clipping. This allows the user to consider such an object as a standard FRep primitive and to apply to it any operation provided by a modeling system.

4.2 Operations

There is a rich set of operations closed on FRep, i.e., resulting in a continuous real-valued function [15]. Unary operations can be classified as space mappings – transformations of point coordinates (affine transformations, standard deformations such as twisting, tapering, or bending, and feature-based nonlinear deformations), functions mappings – transformations of function values at given points (offsetting, solid sweeping [22], and projection [23]). Binary operations include set-theoretic operations (union, intersection, difference) and Cartesian product defined using R-functions (see 4.2.1), Minkowski operations [24], metamorphosis and others. Bounded blending operations (see 4.2.2) take three objects as arguments, thus illustrating k-ary operations in the FRep framework. Similar to primitives, the user of an FRep based modeling system should be able to introduce any desirable operation by its analytical or procedural description and thus extend the list of operations.

As the combination of continuous shape models with the discrete logic based modeling is one of the key points in FRep, we would like to pay special attention to R-functions enabling this combination and to the bounded blending operation as one of applications of R-functions.

4.2.1 R-functions and set-theoretic operations

Following [2-4], we provide here informal description and analytical definitions of several systems of R-functions, and discuss application of them for describing complex geometric objects composed using set-theoretic operations.

Informal description of R-functions

There are such functions whose some "quality" is completely defined by "qualities" of their arguments. The sign of the function is a typical example of such a “quality”. Given the following functions:

$$u_1 = x_1 \cdot x_2 \cdot x_3 \cdot v(x_1, x_2, x_3), (v > 0)$$

$$u_2 = x_1 + x_2 + \sqrt{x_1^2 + x_2^2 + x_1 \cdot x_2}\ ,$$

we can observe that the signs of these functions are completely defined by signs of their arguments and do not depend on arguments values. Note, that "qualities" of arguments and the corresponding "quality" of a function can be different. But anyway, appropriate dependence remains. Such functions are called "R-functions".

If mentioned "qualities" are enumerated in some way, then there is correspondence between any set of "qualities" and a set of their numbers. So, by introducing some R-function, we simultaneously define the rule which gives some "quality" of R-function for any such set and therefore the number of this "quality". This means that there is some logic function matching the R-function with the same number of arguments. Therefore, there is a deep coupling between R-functions and the logic functions that allow for adaptation of the methods inherent in discrete mathematics to classical continuous analysis.

The subject of a special interest is real continuous R-functions and also R-functions with C^m continuity. R-functions belonging to certain defining systems and being built with help of special techniques can have the particular differential and metric properties which can be useful in some applications.

Defining functions of geometric objects

The property of R-functions to inherit and maintain such a quality of their arguments as signs is used in description of complex geometric objects being created from more simple objects using operations defined by R-functions. The geometric interpretation of logic functions is well-known and is widely used in classification of space points in relative to geometric objects (which are considered as points sets). Euler Circles (or Venn Diagrams) can serve as a visual illustration of it.

Suppose we have a set of geometric objects and each of them is given using the inequality of the form $f_i \geq 0,$ where $f_i = f_i(x_1, x_2, ..., x_k)$ is a continuous real-valued function taking positive values at the points inside the object G_i, and zero values at the boundary points of G_i. Let us introduce 3-valued predicates associated with each of f_i and accordingly which each of geometric objects G_i :

$$S_3(f_i) = \begin{cases} 0, & f_i < 0 \\ 1, & f_i = 0 \\ 2, & f_i > 0 \end{cases}$$

To define a new more complex geometric object G as a result of some set-theoretic operations on initial geometric objects $\{Gi\}$, let us introduce the following predicate equation built on the base of 3-valued logic:

$$F[S_3(f_1), S_3(f_2), ..., S_3(f_m)] = A,$$

where F is a certain 3-valued logic function, and $A = \{ 0, 1, 2 \}$. It is required to define the continuous function $f_{m+1} (f_1 , f_2 , ..., f_m)$, which will define the resulting object G.

This function is such that

$$S_3(f_{m+1}(f_1, f_2, ..., f_m)) = F[S_3(f_1), S_3(f_2), ..., S_3(f_m)]$$

The process of constructing the defining function f consist of the following steps:

- representing F as a composition (superposition) of basic 3-valued logic functions (disjunction, conjunction, inversion or complement, and so on);
- performing the formal replacement of logic functions symbols by symbols of the corresponding R-functions;
- performing the formal replacement of '$S_3(f_i)$' symbols by 'f_i'.

After the final step, the defining function f_{m+1} is specified.

The proper choice of F from a set of 3-valued logic functions guarantees that the predicate equations $F = 0, F = 1, F = 2$ are equivalent to inequalities $f < 0, f = 0, f > 0$ and a geometric object with $f = 0$ includes only boundary points and does not include internal points.

Note, that R-function can describe both "algebraic" (that is traditional) geometric objects and "semi-algebraic" ones. It is important that a semi-algebraic object is represented with help of R-functions in the form of a single inequality. Usually such an object (e.g., a rectangle) is represented as a system of equations and inequalities. In principle, the geometric object description is ambiguous as there can be an infinite set of defining functions for the same object. This lets us create defining functions with certain additional properties. In particular, even for semi-algebraic objects, smooth defining functions can be constructed: in particular, such sharp edged object as rectangle can be represented as a result of intersection of a smooth surface with a plane). This is important for computer-based applications (in particular, for visualization).

An object resulting from set-theoretic operations can be described by the following defining functions:

$f_3 = f_1 \vee_\alpha f_2$ for the union;

$f_3 = f_1 \wedge_\alpha f_2$ for the intersection;

$f_3 = f_1 \setminus_\alpha f_2$ for the subtraction,

where f_1 and f_2 are defining functions of initial objects and $\vee_\alpha, \wedge_\alpha, \setminus_\alpha$ are signs of R-functions. Of all the possible descriptions of the R-functions, we use the following analytical formulae:

$$f_1 \vee_\alpha f_2 = \frac{1}{1+\alpha}(f_1 + f_2 + \sqrt{f_1^2 + f_2^2 - 2\alpha f_1 f_2})$$

$$f_1 \wedge_\alpha f_2 = \frac{1}{1+\alpha}(f_1 + f_2 - \sqrt{f_1^2 + f_2^2 - 2\alpha f_1 f_2})$$

where $\alpha = \alpha(f_1, f_2)$ is an arbitrary continuous function satisfying the following conditions:

$$-1 < \alpha(f_1, f_2) < 1,$$

$$\alpha(f_1, f_2) = \alpha(f_2, f_1) = \alpha(-f_1, f_2) = \alpha(f_1, -f_2)$$

The expression for the subtraction operation is

$$f_1 \setminus_\alpha f_2 = f_1 \wedge_\alpha (-f_2)$$

Note that with this definition of the subtraction, the resulting object includes its boundary.

If α=1, the above functions take the form:

$$f_1 \wedge_1 f_2 = \min(f_1, f_2)$$

$$f_1 \vee_1 f_2 = \max(f_1, f_2)$$

These R-functions are very convenient for calculations but have C^1 discontinuity when $f_1 = f_2$. If α=0, the above general formulation takes the most useful in practice form:

$$f_1 \vee_0 f_2 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2}$$

$$f_1 \wedge_0 f_2 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2}$$

These functions have C^1 discontinuity only in points where both arguments are equal to zero. If C^m continuity is to be provided, one may use another set of R-functions:

$$f_1 \vee_m f_2 = (f_1 + f_2 + \sqrt{f_1^2 + f_2^2})(f_1^2 + f_2^2)^{\frac{m}{2}}$$

$$f_1 \wedge_m f_2 = (f_1 + f_2 - \sqrt{f_1^2 + f_2^2})(f_1^2 + f_2^2)^{\frac{m}{2}}$$

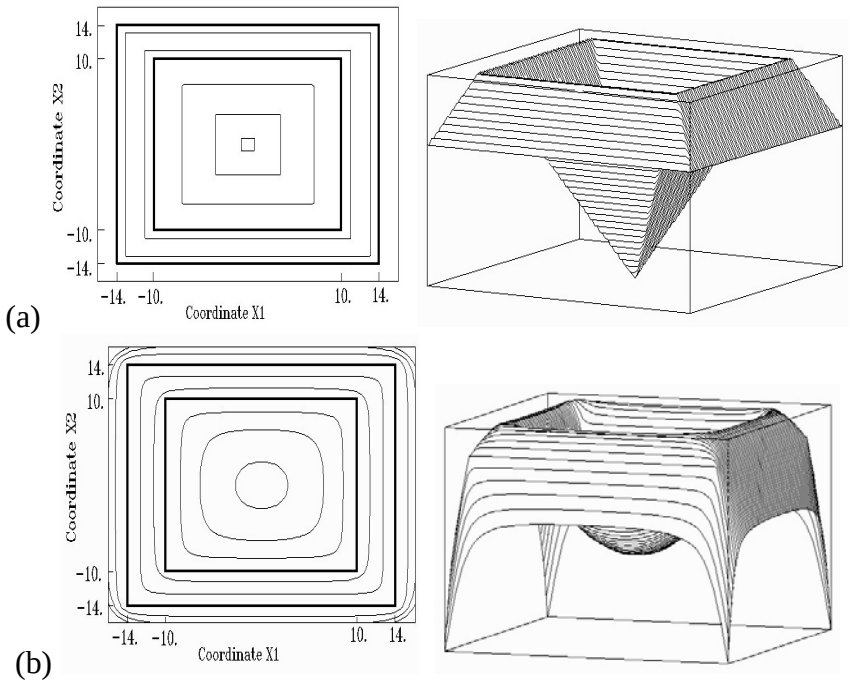


Fig. 5. Contour maps and surfaces of defining functions for a 2D square with a square hole (bold lines): (a) R-functions $\wedge_1$; (b) R-functions $\wedge_0$.

Fig. 5 illustrates the properties of R-functions. The contour maps and surfaces of the function $f_3(x,y)$ are shown for different values of parameter α. Here, $f_3 = f_1 \wedge_\alpha (-f_2)$, where f_1 and f_2 define bigger and smaller square areas respectively. The contour lines $f_3(x,y)=0$ drawn in bold are boundaries of the defined 2D

square with a square hole. Note the smooth non-zero contours for R-functions $\wedge_0$ in Fig. 5b. This property is used in the formulation of several other operations, for example, blending versions of set-theoretic operations.

4.2.2 Bounded blending operations

A blending operation in shape modeling generates smooth transition between two surfaces. Such operations are usually used in computer-aided design for modeling fillets and chamfers. Blending versions of set-theoretic operations (intersection, union, and difference) on solids approximate exact results of these operations by rounding sharp edges and vertices.

The R-functions with $\alpha=0$ have zero contour lines with sharp vertices (bold line in Fig. 5b). Other contour lines are smooth in the entire domain. This property brings the idea that some displacement of the exact R-function can result in the blending effect. The following definition of a blending set-theoretic operation was proposed in [25]:

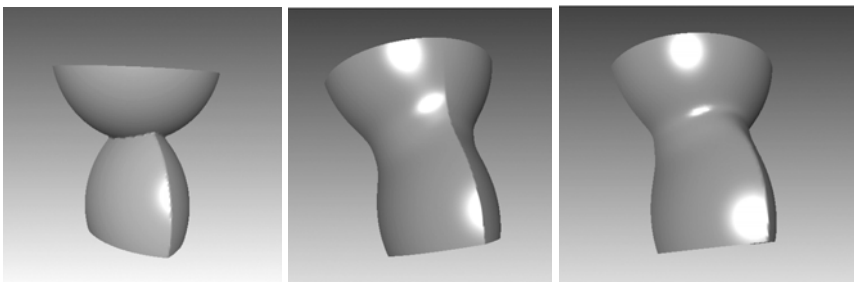
$$F_b(f_1, f_2) = R(f_1, f_2) + disp_b(f_1, f_2),$$

where $R(f_1, f_2)$ is an R-function corresponding to the type of the operation, the arguments of the operation $f_1(X)$ and $f_2(X)$ are defining functions of two initial solids, and $disp(f_1, f_2)$ is a Gaussian-type displacement function. The following expression for the displacement function was used:

$$disp_b(f_1, f_2) = \frac{a_0}{1+\left(\frac{f_1}{a_1}\right)^2+\left(\frac{f_2}{a_2}\right)^2},$$

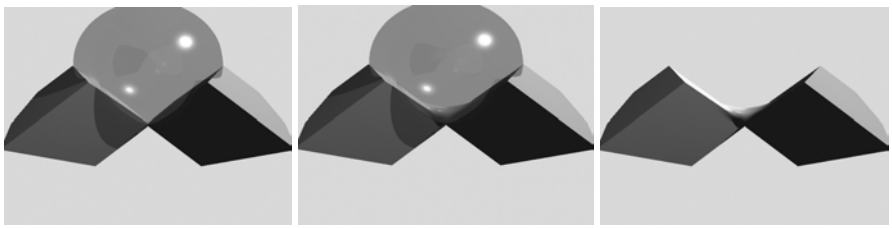
where a_0, a_1, an a_2 are parameters controlling the shape of the blend. The proposed definition is suitable for blending union, intersection, and difference and allows for generating added and subtracted material, as well as symmetric and asymmetric blends.

Blending to the edge is one of the challenging operations. Fig. 6 shows blending union of a sphere (top object) to the edge produced by intersection of two other spheres (bottom object). In the case when the bottom object is constructed using min function (Fig. 6b), the edge is present on the blend surface because of C^1 discontinuity of the min function on a plane passing through the initial edge. In the case when the bottom object is constructed using the R-function $\wedge_0$, the blend surface is smooth (Fig. 6c).



a b c

Fig. 6. Blending to the edge: a) sphere (top object) is blended to the intersection of two other spheres (bottom object); b) min function is used for the bottom object construction; c) R-function is used for the bottom object construction.



a b c

Fig. 7. Shape and position of 3D blend are controlled by the bounding ellipsoid

However, the above displacement function does not get zero value anywhere in the space. This is the reason of the main disadvantage of this definition - the blend has global character and cannot be localized using its parameters. In [26], a different displacement function was proposed, which allows for localization of the blend using an additional bounding solid:

$$disp_{bb}(r) = \begin{cases} \dfrac{(1-r^2)^3}{1+r^2}, r < 1 \\ 0, r \geq 1 \end{cases}$$

where r is a generalized distance constructed using defining functions of two initial solids (f_1 and f_2) and a bounding solid (function f_3):

$$r^2 = \begin{cases} \dfrac{r_1^{\,2}}{r_1^{\,2} + r_2^{\,2}}, r_2 > 0 \\ 1, r_2 = 0 \end{cases}, \quad \text{where}$$

$$r_1^2(f_1, f_2) = \left(\frac{f_1}{a_1}\right)^2 + \left(\frac{f_2}{a_2}\right)^2,$$

$$\text{and } r_2^2(f_3) = \begin{cases} \left(\frac{f_3}{a_3}\right)^2, & f_3 > 0 \\ 0, & f_3 \le 0 \end{cases},$$

with numerical parameters a_1 and a_2 controlling the blend symmetry, and a_3 allowing the user to interactively control the influence of the function f_3 on the overall shape of the blend.

The shape and position of the bounded blend are controlled by its parameters and by the position and shape of the bounding solid. Control of blend shape and position is illustrated by Fig. 7. The pure union of two polyhedral shapes (Fig 7a) is changed to the bounded blending union using the bounding ellipsoid (transparent shape). The resulting blend is located strictly inside the bounding ellipsoid (Fig. 7b), which produces an unusual blending shape localized at the top part of the initial union of polyhedrons (Fig. 7c).

Other unusual applications of bounded blending illustrated in [26] are multiple blending with a bounding solid consisting of several disjoint components, partial edge blending with added and subtracted material, and blend on blend. Note that the bounded blending operation requires three objects as an argument and thus it formally belongs to the class of 3-ary operations of FRep.

4.3 Relations

Let us give here examples of binary relations for point membership and intersection relations between two objects. Similarly to the primitives and operations, the user should be given a possibility to extend the set of relations by providing symbolic or procedural definitions of their predicates.

Point membership relation

Let iG_1 be the interior of the object G_1 and bG_1 be the boundary of G_1. The point membership relation is described by the 3-valued predicate:

$$S_3(P, G_1) = \begin{cases} 0, \text{if } f_1(\mathbf{X}) < 0 \quad \text{for } P \notin G_1 \\ 1, \text{if } f_1(\mathbf{X}) = 0 \quad \text{for } P \in bG_1 \\ 2, \text{if } f_1(\mathbf{X}) > 0 \quad \text{for } P \in iG_1 \end{cases}$$

Note that the system of set-theoretic operations based on R-functions corresponds to the operations of 3-valued logic over predicates S_3 but not to the Boolean logic.

Intersection relation

The intersection (or collision) relation indicates if two objects have common points and is defined by the bi-valued predicate

$$S_c(G_1, G_2) = \begin{cases} 0, \text{if } G_1 \cap G_2 = \varnothing \\ 1, \text{if } G_1 \cap G_2 \neq \varnothing \end{cases}$$

The function $f_3(\mathbf{X}) = f_1(\mathbf{X}) \wedge_\alpha f_2(\mathbf{X})$ defining the result of the intersection can be used to evaluate S_c. It can be stated that $S_c = 0$ if $f_3(\mathbf{X}) < 0$ for any point of E^n [4]. This property can be used for the numerical collision detection based on the maximum search for $f_3(\mathbf{X})$.

5 Internal representation

In this section we provide an outline of the formal specification of the FRep based geometric modeling system. We use the constructive technique based on Vienna Development Method (VDM) [27]. Of course, we can give here only a simplified fragment of the whole VDM specification, and many details are omitted. However, we believe that even in such an incomplete form the specification fragment does provide a solid mathematical framework for building high level representative view of the system which can be useful for understanding internal data structures and computing processes.

We use a limited subset of VDM notation which is close to the one used in [28] for specifying GKS. This subset has a traditional though somewhat simplified and mnemonically informative notation. First of all, the specification defines principal abstract data types and operations over them.

Let us introduce the concept of FRep_Machine which embodies state-based model of our geometric modeling system. The state of that underlying abstract machine is defined in the form of the Abstract Syntax (see Fig. 8). Line 1 introduces FRep_Machine definition in terms of four components that are briefly described below.

Geometric environment Geom_Env (line 2) is defined as a finite map and states the meaning of all the geometric entities present in the system based on their names Geom_Ident. Geometric entities (lines 7 – 11) are defined as composite objects having a number of fields taking values from specified domains. The names attached to these fields ('s_*_*') serve as selectors (accessor functions), each from the domain of the composite object to the domain of the relevant field to make all the constituent fields accessible.

For instance, each object with generic type Gob (line 7) contains the fields stating its names with type 'Gob_Ident', dimensionality of integer type N, a list of formal parameters 'Par*', a list of formal coordinate variables 'X*', and its representation in terms of geometric tree 'Gob_Tree'. The geometric tree is itself specified as a composite object (line 11) recursively defining a k-ary constructive tree structure. Note

that primitive geometric objects Pob (line 8) are represented by a structure of ‘FRep_Pob’ type which can embody a functional expression, procedure, etc. A special object is a point in the modeling space ‘Point’ (line 6) being characterized by its dimensionality and a list of its coordinates. Relations (line 10) are represented by a structure embodying predicate.

As to operations (line 9) , they are represented by a structure with type ‘FRep_Transf’ embodying a certain transformation of FRep. We can deal with k-ary operations (lines 12-13) whose generic types can be unary (function mappings ‘Op_F_Map’, space mapping ‘Op_Space_Map’, extended space mapping ‘Op_Extended_Space_map’, projection ‘Op_Projection ’, etc.), binary set-theoretic (‘Op_ST_U’ and ‘OP_ST_Bi’), and ternary bounded blending ‘OP_Blend_Ternary’.

Geometric store ‘Geom_Store’ (line 3) is a finite map that provides lists of actual parameters for all the geometric entities. Numerical store ‘Num_Store’ allows for binding of variable identifiers with their values.

Operations on introduced objects are represented as transition functions of FRep_Machine. Let us present here only one yet the most important operation, namely the one evaluating a defining function for a certain instance of a complex geometric object gob at the given point of the modeling space. The definition of such a function ‘f_gob_eval’ is shown in Fig. 9. It is recursive and makes use a number of standard key words and operators such as ‘**let** x = … **in** …’ introducing local variables in function definitions.

There are a number of other functions in that definition. Auxiliary functions ‘Value_list’ and ‘Value_par’ (lines 5 and 7) provide binding of parameters with their actual values through access to the numerical store and usage of standard functions dealing with lists. The principal function ‘f_gob_eval’ is eventually reduced to ‘f_gob_tree_eval’ (lines 9-56) which deals with the tree structure ‘gob_tree’ corresponding to that instance gob (see line 4). Let us give some comments on that function.

Firstly, the signature of each input and output variables defining the function type (line 9) is defined. The symbol ‘ $\underline{\Delta}$ ’ (line 10) denotes syntactic equivalence. Line 11 contains precondition on input variables; note that we do not define here invariants such as ‘inv_gob_tree’ proving “well-formedness” condition. Then, using selector ‘s_tree_node’ (see Fig. 8) we get an access to the geometric entity ‘geom_ident_node’ in the tree node (line 12) and form a list of the point coordinates (line 13). Predicates ‘is_*’ serve for recognizing the type of ‘geom_ident_node’.

If ‘geom_ident_node’ is recognized as having ‘Gob’ type (line 14), we get all the necessary data about it from the geometric environment (line 15) and the function ‘f_gob_eval’ is recursively applied. If ‘Pob’ type is recognized (line 17), we get all the information about this primitive from the geometric environment and the geometric store (lines 18-19) with its subsequent substitution into the selector function ‘s_pob_rep’ (line 20).

Fig. 10 shows how the function 'f_gob_eval' can be used for defining point membership relation.

1. FRep_Machine = Geom_Env × Geom_Store × Point × Num_Store
2. Geom_Env = Geom_Ident $\xrightarrow{m}$ Geom_Entity
3. Geom_Store = Geom_Ident $\xrightarrow{m}$ Arg*
4. Num_Store = Ident $\xrightarrow{m}$ Val
5. Geom_Entity = Gob | Pob | Op | Rel

6. Point:: s_point_dim: N
   s_point_coord: Arg*

7. Gob:: s_gob_id: Gob_Ident
   s_gob_dim: N
   s_gob_par: Par*
   s_gob_coord: X*
   s_gob_rep: Gob_Tree

8. Pob:: s_pob_id: Pob_Ident
   s_pob_dim: N
   s_pob_par: Par*
   s_pob_coord: X*
   s_pob_rep: FRep_Pob

9. Op:: s_op_id: Op_Ident
   s_op_dim: N
   s_op_par: Par*
   s_op_coord: X*
   s_op_rep: FRep_Transf

10. Rel:: s_rel_id: Rel_Ident
    s_rel_dim: N
    s_rel_par: Par*
    s_rel_rep: FRep_Pred

11. Gob_Tree:: s_tree_arity: N
    s_tree_node: Geom_Ident
    s_tree_leaves: (N × Gob_Tree | **nil)***

12. Op = Op_Unary | OP_ST_Bi | OP_ Blend_Ternary
13. Op_Unary =Op_ST_U ‖ Op_F_Map | Op_Space_Map | Op_Extended_Space_map | Op_Projection | …

14. Geom_Ident = Gob_Ident | Pob_Ident | Op_Ident | Rel_Ident
15. Par = Ident
16. Arg = Val | Ident
17. Val = Real | **Undef**

Fig. 8. VDM specification: State of the FRep_Machine (fragment of an abstract syntax)

```
1. type f_gob_eval: Gob × Point × FRep_Machine → Val
2. f_gob_eval (gob, point, frep_machine) ≜
3.  pre is_Gob (gob) ∩ (s_point_dim (point) = s_gob_dim (gob));

4.  let gob_tree = s_gob_rep (gob) in
        f_gob_tree_eval (gob_tree, point, frep_machine);

5. type Value_list: Arg* × Num_Store  → Val*
6. Value_list (arg_list, num_store) ≜
  if arg_list = nil then nil
   else conc (Value_par (hd (arg_list, num_store) ), Value_list (tl (arg_list), num_store) );

7. type Value_par: Arg × Num_Store  → Val
8. Value_par (arg, num_store) ≜
        is_Ident (arg) → num_store(arg);
        is_Val (arg) → arg;

9. type f_gob_tree_eval: Gob_Tree × Point × FRep_Machine → Val
10. f_gob_tree_eval (gob_tree, point, frep_machine) ≜
11.  pre inv_gob_tree (gob_tree, frep_machine);
12.  let geom_ident_node = s_tree_node (gob_tree) and
13.      coord_value_list = Value_list ( s_point_coord (point) ) in

14.  is_Gob_Ident (geom_ident_node) →
15.     let gob_node = geom_env (geom_ident_node) in
16.                 f_gob_eval (gob_node, point, frep_machine);

17.  is_Pob_Ident (geom_ident_node)  →
18.     let pob = geom_env (geom_ident_node) and
19.     par_value_list = Value_list (geom_store (geom_ident_node) ) in
20.                 let frep_pob = s_pob_rep (par_value_list, coord_value_list);

21.   is_Op_Unary_Ident (geom_ident_node) →
22.     let op_un = geom_env (geom_ident_node) and
23          par_value_list = Value_list (geom_store (geom_ident_node) ) and
24.                 let gob_tree_left = s_tree_leaves (gob_tree, 1)  in

25.     is_Op_F_Map_Ident (geom_ident_node) →
26.                 let  frep_transf = s_op_rep (op_un) in
27.                             frep_transf (par_value_list, coord_value_list,
28.                       f_gob_tree_eval (gob_tree_left, point, frep_machine));

29.     is_Op_Space_Map_Ident (geom_ident_node) →
30.                 let  frep_transf = s_op_rep (op_un) in
31.                 let inverse_coord_list =
```

```
32.                     Value_list (frep_transf (par_value_list, coord_value_list) ) in
33.                                     let inverse_point =
34.                     mk_point (s_point_dim (point),  inverse_coord_list) in
35.                     f_gob_tree_eval (gob_tree_left, inverse_ point, frep_machine);

36.   is_Op_Extended_Space_Map_Ident (geom_ident_node) →
      <…>
37.   is_Op_ST_Bi_Ident (geom_ident_node) →
38.           let op_st_bi = geom_env (geom_ident_node) and
39.   par_value_list = Value_list (geom_store (geom_ident_node) in
40.           let gob_tree_left = s_tree_leaves (gob_tree, 1) and
41.               gob_tree_right = s_tree_leaves (gob_tree, 2) in
42.                     let frep_transf = s_op_rep (op_st_bi) in
43.                                 frep_transf (par_value_list, coord_value_list,
44.                     f_gob_tree_eval (gob_tree_left, point, frep_machine),
45.                     f_gob_tree_eval (gob_tree_right, point, frep_machine) );

46.    is_Op_Blend_Ternary_Ident (geom_ident_node) →
47.           let op_bl_tern = geom_env (geom_ident_node) and
48.               par_value_list = Value_list (geom_store (geom_ident_node) in
49.                     let gob_tree_1 = s_tree_leaves (gob_tree, 1) and
50.                         gob_tree_2 = s_tree_leaves (gob_tree, 2) and
51.                         gob_tree_3 = s_tree_leaves (gob_tree, 3)  in
52.                               let frep_transf = s_op_rep (op_bl_ter) in
53.                                       frep_transf (par_value_list, coord_value_list,
54.                               f_gob_tree_eval (gob_tree_1, point, frep_machine),
55.                               f_gob_tree_eval (gob_tree_2, point, frep_machine)
56.                               f_gob_tree_eval (gob_tree_3, point, frep_machine) );
```

Fig. 9. VDM specification: Evalation of defining function for geometric object

```
type pred_point_membership: Gob × Point × FRep_Machine → Int
pred_point_membership (gob, point, frep_machine)  ≜
      let f_value = f_gob_eval (gob, point, frep_machine) in
            is_equal_zero (f_value)

type is_equal_zero: Val → Int
      <…>
```

Fig. 10. VDM specification: Point membership relation

Then, we check if ‘geom_ident_node’ is a unary operation (line 21); if so, we get its data from the geometric environment and form the list of its actual parameters (lines 22-23). We get an access to its only subtree ‘gob_tree_left’ and can act depending on the type of that operation. In case this is a “Function mapping” operation (line 25) we just get the corresponding representation ‘frep_transf‘ of this operation (ine 26) and evaluate it with proper parameters (line 27) using recursively applying ‘f_gob_tree_eval’ function (line 28).

If this is a “Space mapping” operation (line 29), one should get an inverse list of coordinates (line 31) with subsequent recursive evaluation of ‘f_gob_tree_eval’ at the point ‘inverse_point’ formed with help of “make” function ‘mk_point’ (lines 33-35). As to an operation of the “Extended space mapping” type combining a function mapping and a space mapping, it can be implemented by combining two previously described ones with two subsequent (down and up) passes of the tree (omitted in the Fig. 9).

Finally, let us consider the case when ‘geom_ident_node’ is a ternary bounded blending operation (line 46-56). The evaluation involves forming the lists of actual parameters and a recursive application of ‘f_gob_tree_eval’ for all three subtrees. A similar procedure is applied in the case of a binary set-theoretic operation (lines 37-45).

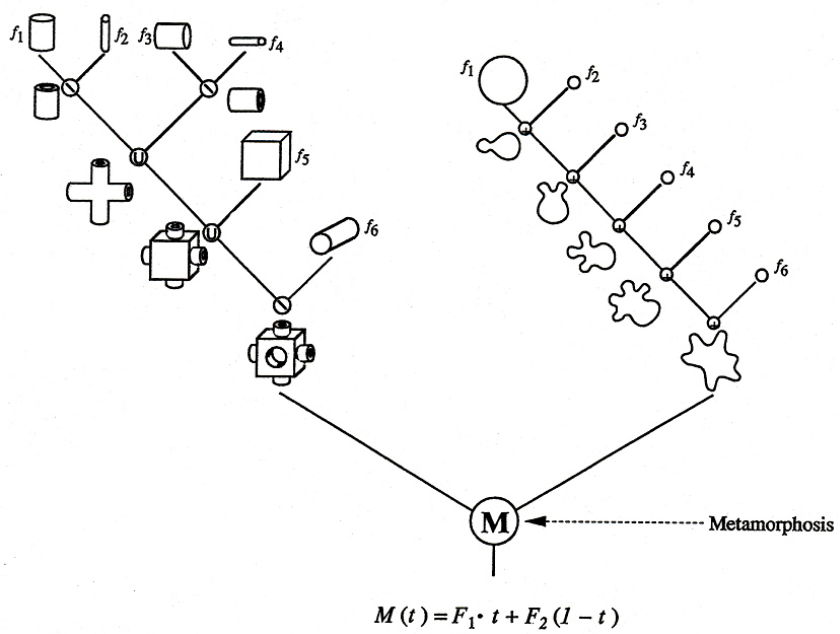


Fig. 11. Tree structure for a metamorphosis operation between a constructive solid and a blobby object defined using algebraic sums of individual blobs.

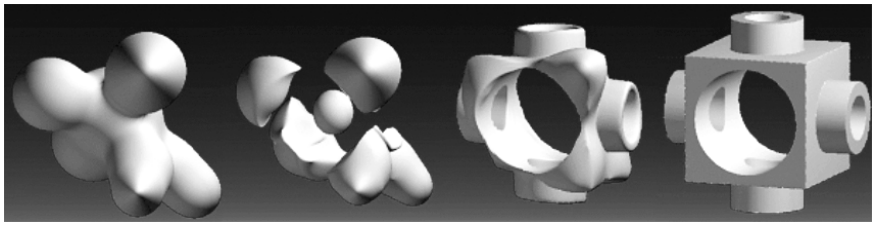


Fig. 12. Stages of metamorphosis between a constructive solid and a blobby object

Fig. 11 shows an example of the internal tree structure for a 4D (time-dependent) object, defined as a binary metamorphosis operation between a constructive solid (left subtree) and a blobby object (right subtree with only algebraic sums in the nodes). Several stages of the object transformation are shown in Fig. 12.

6 Modeling heterogeneous objects

Modeling heterogeneous objects is becoming an important research topic. Heterogeneous objects are considered in such different areas as modeling of objects with multiple materials and varying material distribution in CAD/CAM and rapid prototyping, representing results of physical simulations, geological and medical modeling, volume modeling and rendering. We consider real or abstract heterogeneous objects that have internal structure with non-uniform distribution of material and other attributes of an arbitrary nature (photometric, physical, statistical, etc.), and elements of different dimension. It means these objects are heterogeneous with respect to their structure and dimensionality.

Multidimensional point sets with a fixed dimensionality and with multiple attributes can be quite effectively dealt with using a *constructive hypervolume model* based on real-valued vector-functions (see 6.1). The requirement of dimensional heterogeneity naturally brings the idea of adding a kind of cellular representation to the model while using FRep. Moreover, different applications such as CAD or finite-element analysis require an explicit representation of mixed-dimensional objects along with the functional one. These are the main motivations for introduction of a new hybrid cellular-functional model (see 6.2).

6.1 Constructive hypervolume model

A hypervolume object can be defined as:

$$o = (G, A_1, ..., A_k) : (F(X), S_1(X), ..., S_k(X))$$

where $X = (x_1, \ldots, x_n)$ is a point in n-dimensional Euclidian space E^n, $F: X \to \Re$ is a real-valued defining function of point coordinates representing the point set G, $S_i: X \to \Re$ is a real-valued scalar function representing an attribute A_i that is not

necessarily continuous. The point set G and the attribute functions S_i are defined by real-valued functions using FRep. The function F corresponding to an FRep object is at least C^0 continuous and defined in the Euclidean space E^n. As it was discussed earlier, the function F is evaluated by a procedure traversing a tree structure, where primitives are placed at the leaves, and operations at the nodes. Because the function F is built using a constructive method, the proposed model was called a *constructive hypervolume model*. Similarly, attribute functions S_i are evaluated by traversing the corresponding tree structures. Therefore, we can state that the internal representation of a constructive hypervolume includes a *constructive geometric tree* and *constructive attribute trees*.

The proposed model was used in [29] to extend the well-known concept of solid texturing in two directions: constructive modelling of space partitions for texturing and modelling of multidimensional textured objects. Some operations on attributes such as color blending were also discussed. Other applications of constructive hypervolumes include

- Rapid prototyping and fabrication of objects with multiple materials and varying material distribution;
- Physics based simulations for the analysis of physical field distributions over the given geometric areas;
- Analysis of geological structures;
- Medical examination and surgery simulation using data from computer tomography and other scanning devices;
- Modeling and visualization of amorphous and gaseous phenomena.

6.2 Implicit complexes

Let us first consider a 2D space and k-dimensional cells in it:
$k=2$: 2D solid or planar area;
$k=1$: curves and their segments;
$k=0$: points.

It is clear that 2D solids can be defined as $f(x,y) \geq 0$. Rvachev [4] discussed the construction of such a definition for lower dimensional objects. The main idea is that the function $f(x,y)$ has to be zero only at the points of this object and negative everywhere else. For example, if one wants to describe a straight line segment on a 2D plane, an equation of a straight line can be used: $w(x,y)=ax+by+c$. The inequality $w \geq 0$ defines a halfplane. Then, $-w^2 \geq 0$ defines the line itself, where in fact the function $-w^2$ is never positive and only becomes zero on the line. The line can be trimmed using some 2D solid to produce one or several segments. The simplest way to define a segment is $-w^2 \wedge_0 g \geq 0$, where $g(x,y) \geq 0$ is a definition of a 2D solid disk.

In 3D space, we can also define points, curve segments, and surface patches as follows:

- "implicit" definition of a surface patch requires an "implicit" surface and a trimming 3D solid

- a curve can be defined as an intersection of two surfaces, each defined as $-f^2 \geq 0$
- a point can be defined as the intersection of three surfaces, a curve and a surface, or directly as *d(x,y,z)*, where d is a negative distance to the given point.

Rvachev et al. [30] showed that such a function representation of lower dimensional cells was quite useful in solving interpolation and boundary value problems.

In [31], a hybrid cellular-functional model of heterogeneous objects was introduced. It combines a cellular representation and a constructive representation using real-valued functions. A notion of an *implicit complex* was introduced in [31]. Such a complex has the following features:

- it is defined as a union of *properly joined* cellular spaces (which we call *domains* to distinguish them from general cellular spaces);
- subspaces that are shared by two or more domains are represented by explicitly defined cellular subcomplexes; accordingly, some domains can be represented by explicit cellular complexes; for other domains, it is enough to have only a few explicit cells to form a base for the intersection; as to their complete representation, it is defined functionally;
- as all these cellular spaces are properly joined, they ultimately form a complex, which we call an implicit one.

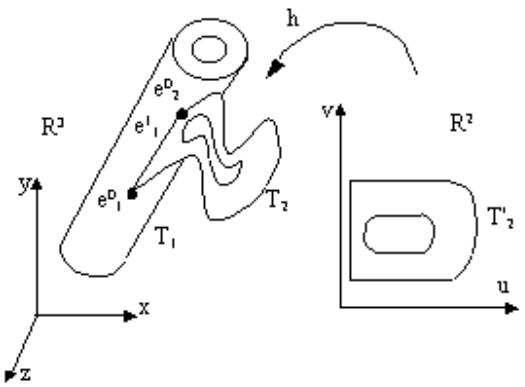


Fig. 13. Example of an implicit complex in E^3.

An example of an implicit complex is given in Fig. 13. Here, using domains with different dimensionalities is necessary. The object consists of a cylinder with a longitudinal hole and a bent surface patch with a hole, which is attached to the cylinder's surface. Here, we use two FRep domains. The first one, D_1 is represented as a 3D cylinder defined functionally as $F_1(x,y,z) \geq 0$. The explicit part of this domain consists of cells corresponding to two "attaching points" $\{e^0_1, e^0_2\}$ and one "attaching" line segment e^1_1. The second FRep domain D_2 presents a more interesting case. The point set T_2^2 has its preimage $(T_2^2)'$, which is defined on the plane *(u,v)* by a function $F_2(u,v) \geq 0$. Then, we must define embedding map h: *E2 -> E3*, which gives us ho-

meomorphism taking $(T_2^2)'$ to T_2^2. The second domain has the same explicit part as the first one.

The hybrid cellular-functional model allows for independent but unifying representation of geometry and attributes of heterogeneous objects, and makes it possible to represent dimensionally non-homogeneous entities and their cellular decompositions. It is a new research area and a lot of work should be done on related operations, conversion to conventional models, as well as on the development of corresponding software tools and applications.

7 Specialized language

In principle, one can utilize a universal programming language like C or Java as an FRep modeling language. However, such languages are too complex and have a lot of unnecessary features for application to shape representation. Their generalised compilation tools make the specialized task of constructing shape models unnecessarily difficult. In [13], we introduced a modeling system architecture built around the shape models in HyperFun, which is a specialized high-level programming language for specifying FRep models.. The language is designed to include all conventional programming constructs and specialized operators necessary for modeling complex objects, yet without any redundant features. While being minimalist, it supports all main notions of FRep. HyperFun is also intended to serve as a lightweight exchange protocol for FRep models to support platform independence and Internet-based collaborative modeling.

A model in HyperFun can contain the specification of several FRep or constructive hypervolume objects parameterized by input arrays of point coordinates x_i and numerical parameters a_i whose values are to be passed from outside the object. The number of coordinate variables can be greater than three to allow for definition of higher dimensional objects. Each object is defined by a function describing its geometry (the function's name coincides with the object's name) accompanied, if necessary, by a set of scalar functions s_i representing its attributes. The function can be quite complex: it is represented with the help of assignment statements (using auxiliary local variables and arrays, if necessary); conditional selection ('if-then-else'), and iterative ('while-loop') statements. Functional expressions are composed using conventional arithmetic and relational operators. It is possible to use standard mathematical functions ('exp', 'log', 'sqrt', 'sin', etc.). Fundamental set-theoretic operations are supported by special built-in operators with reserved symbols ("|" - union, "&" - intersection, "\" - subtraction, "~" - negation, "@" - Cartesian product). Functional expressions can also include references to previously defined geometric objects.

In principle, the language is self-contained and allows users to build objects from scratch, without the use of any pre-defined primitives and transformations. However, its expressive power is greatly increased by the availability of the system "FRep library" that is easily extendable and can be adapted to particular application domains and can even be customised for the needs of particular users. Currently, the version of the FRep library in general use contains the most common primitives and transforma-

tions of quite a broad spectrum. The user can create his/her own library of objects for later reuse. A sample of a HyperFun model can be found in Fig. 14.

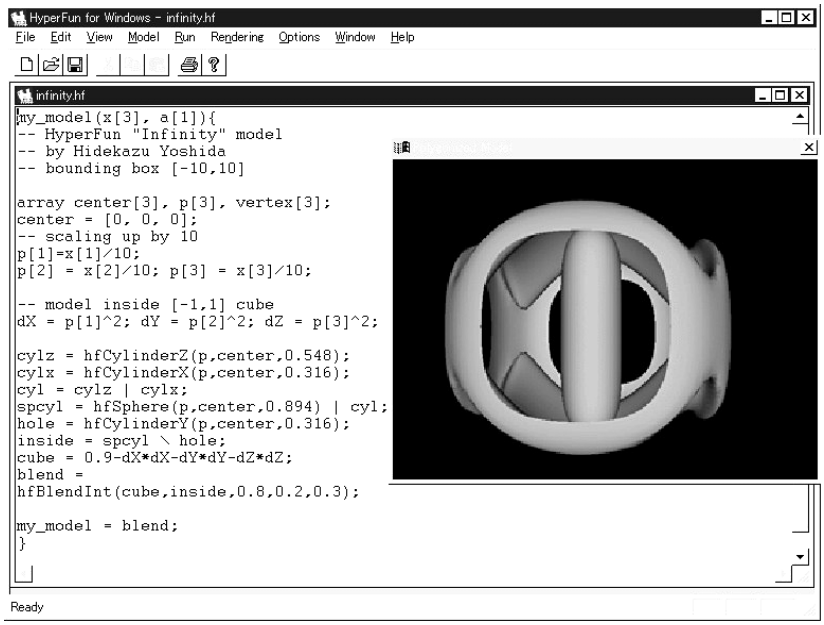


Fig. 14. Example of a HyperFun program and the corresponding image of the model in the HyperFun for Windows modeling environment

Application software deals with HyperFun models through using either a built-in interpreter or HyperFun-to-C/HyperFun-to-Java compilers and utilities of the HyperFun API. The HyperFun interpreter has been implemented as a small set of functions in ANSI C. It is quite easy to integrate them into the application software since the developer needs to deal with only two C-functions. The 'Parse' function performs syntax analysis in accordance with the language grammar and semantic rules. For each object described in the HyperFun program, the function generates an internal representation that is actually a collection of the tree structures optimized for subsequent efficient evaluation. If there are any errors in the program, the function outputs a list containing the location and details of each error found. Another interpreter function ('Calc') is called every time when there is a need to evaluate the function at a given point in the modeling space and for the given external numerical parameters. Externally defined values for attribute scalar functions can be passed too. The object's internal representation serves as an input parameter for 'Calc' function that returns both the value of the "geometric" function and a set of values for "attribute" scalar functions - all evaluated at the given point.

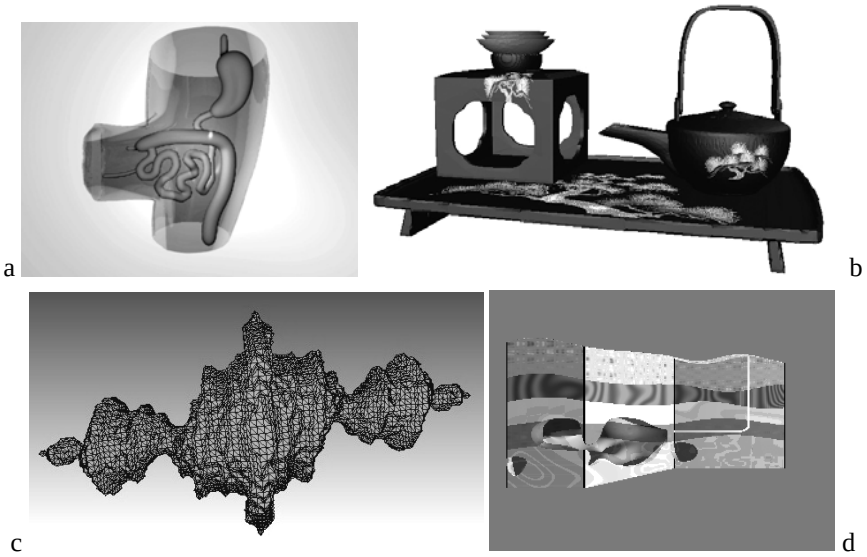


Fig. 15. HyperFun models: a) human embryo digestive system (courtesy of R. Durikovic and S. Czanner); b) Japanese lacquer ware; c) 3D quaternion Julia fractals (courtesy of F. Delhoume); d) multilayer geological structure with internal material attribute distribution, an oil well, and cavities (heterogeneous object).

The formal specification of the internal representation and of the function evaluation procedure was given above. The function 'Parse' is invoked just once while processing the HyperFun program to generate a representation that can be considered as a "byte-code" and can serve as a protocol for data exchange between system components 'Parse' and 'Calc'. In fact, these constitute an application programming interface (API) that is simple to use.

The software tools developed to support HyperFun include a polygonizer (surface mesh generator), a plug-in to a POVRay ray-tracer, and a set of Web-based modeling tools such as translator to Java, polygonizer in Java, and interactive modeler based on empirical modeling principles and provided as an applet. Examples of HyperFun models created and rendered using these tools are shown in Fig. 15.

Main current application areas of FRep and HyperFun include education (geometry and geometric modeling, computer graphics, programming languages), biological modeling, cultural heritage preservation, animation and multimedia, and computer art. The constructive hypervolume models can be applied in multiple material rapid prototyping, geological and biological modeling, physics based simulations, and volume graphics. We are also planning to develop an advanced computer-aided design system based on several geometric representations including FRep and the constructive hypervolume model.

Acknowledgements

We would like to thank all the co-authors of FRep related papers and developers of HyperFun software tools, who contributed their knowledge and efforts to our joint research and development during past 15 years.

References

1. Requicha A., Representations of rigid solids: theory, methods, and systems, Computing Surveys, vol. 12, No. 4 (1980) 437-464.
2. Rvachev V.L., On the analytical description of some geometric objects, Reports of Ukrainian Academy of Sciences, vol. 153, No. 4 (1963) 765-767.
3. Rvachev V.L. Methods of Logic Algebra in Mathematical Physics, Naukova Dumka, Kiev (1974) (in Russian).
4. Rvachev V.L., Theory of R-functions and some applications, Naukova Dumka, Kiev (1982) (in Russian).
5. Ricci A., A constructive geometry for computer graphics, The computer Journal, vol. 16, No. 2 (1973) 157-160.
6. Blinn J., A generalization of algebraic surface drawing, ACM Transactions on Graphics, vol. 1, No. 3 (1982) 235-256.
7. Bloomenthal J. et al., Introduction to Implicit Surfaces, Morgan Kaufmann Publishers, San Francisco (1997).
8. Bowyer A., SVLIS Set-theoretic Kernel Modeller, Introduction and User Manual, Information Geometers, Winchester, UK (1995).
9. Shapiro V., Theory of R-functions and applications: a primer, TR CPA88-3, Cornell University (1988).
10. Shapiro V., Real functions for representation of rigid solids, Computer Aided Geometric Design, 11(2) (1994)153-175.
11. Tsukanov I., V. Shapiro V., The architecture of SAGE - a meshfree system based on RFM, Engineering with Computers, vol. 18, No. 4 (2002) 295-311.
12. Wyvill B., Galin E., Guy A., Extending the CSG tree. warping, blending and Boolean operations in an implicit surface modeling system, Computer Graphics Forum, vol. 18, No. 2 (1999) 149-158.
13. Adzhiev V., Cartwright R., Fausett E., Ossipov A., Pasko A., Savchenko V., HyperFun project: a framework for collaborative multidimensional F-rep modeling, Implicit Surfaces '99 Workshop (Bordeaux, France), J. Hughes and C. Schlick (Eds.) (1999) 59-69, URL http://www.hyperfun.org/
14. Pasko A., Savchenko V., Adzhiev V., Sourin A., Multidimensional geometric modeling and visualization based on the function representation of objects, Technical Report 93-1-008, The University of Aizu, Japan (1993) 47 p.
15. Pasko A., Adzhiev V., Sourin A., Savchenko V., Function representation in geometric modeling: concepts, implementation and applications, The Visual Computer, vol.11, No.8 (1995) 429-446, URL http://wwwcis.k.hosei.ac.jp/~F-rep/
16. McCormack J., Sherstyuk A., Creating and rendering convolution surfaces, Computer Graphics Forum, vol. 17, No. 2 (1998) 113-120.
17. Perlin K., Hoffert E., Hypertexture, *SIGGRAPH'89, Computer Graphics*, vol. 23, No. 4 (1989) 253-262.

18. Sourin A., Pasko A., Savchenko V., Using real functions with application to hair modelling, Computers and Graphics, vol. 20, No. 1 (1996) 11-19.
19. Savchenko V., Pasko A., Okunev O., Kunii T., Function representation of solids reconstructed from scattered surface points and contours, Computer Graphics Forum, vol.14, No.4 (1995) 181-188.
20. Miura K., Pasko A., Savchenko V., Parametric patches and volumes in the functional representation of geometric solids, Set-theoretic Solid Modeling: Techniques and Applications, CSG 96 (Winchester, UK, 17-19 April 1996), Information Geometers, UK (1996) 217-231.
21. Schmitt B., Pasko A., Schlick C., Constructive modelling of FRep solids using spline volumes, Sixth ACM Symposium on Solid Modeling and Applications (June 6-8, 2001, Ann Arbor, USA), D. Anderson, K. Lee (Eds.), ACM Press (2001)321-322.
22. Sourin A., Pasko A., Function representation for sweeping by a moving solid, IEEE Transactions on Visualization and Computer Graphics, vol.2, No.1, (1996) 11-18.
23. Pasko A., Savchenko V., Projection operation for multidimensional geometric modeling with real functions, Geometric Modeling: Theory and Practice, W. Strasser, R. Klein, R. Rau (Eds.), Springer-Verlag, Berlin/Heidelberg (1997) 197-205.
24. Pasko A., Okunev O., Savchenko V., Minkowski sums of point sets defined by inequalities, Computers and Mathematics with Applications, Elsevier Science, vol. 45, No. 10/11 (2003) 1479-1487.
25. Pasko A., Savchenko V., Blending operations for the functionally based constructive geometry, Set-theoretic Solid Modeling: Techniques and Applications, CSG 94 Conference Proceedings, Information Geometers, Winchester, UK (1994) 151-161.
26. Pasko G., Pasko A., Ikeda M., Kunii T., Bounded blending operations, Shape Modeling International 2002, Banff (Canada, May 17-22), IEEE Computer Society (2002) 95-103.
27. Bjorner D., Jones C., Formal specification and software development, Prentice-Hall, Englewood Cliffs, N.J. (1982).
28. Duce D., Fielding E., Formal specification – a comparison of two techniques, The Computer Journal, vol. 30, No. 4 (1987) 316-327.
29. Pasko A., Adzhiev V., Schmitt B., Schlick C., Constructive hypervolume modeling, Graphical Models, special issue on Volume Modeling, vol. 63, No. 6 (2001) 413-442.
30. Rvachev V., Sheiko T., Shapiro V., Tsukanov I., Transfinite interpolation over implicitly defined sets, Computer-Aided Geometric Design, 18 (2001) 195-220.
31. Adzhiev V., Kartasheva E., Kunii T., Pasko A., Schmitt B., Hybrid cellular-functional modeling of heterogeneous objects, Journal of Computing and Information Science in Engineering, Transactions of the ASME, vol. 2, No. 4 (2002) 312-322.