

Scanned section "Internal system specification" (pages 10-14) not published in other papers.

Technical Report 93-1-008

**MULTIDIMENSIONAL GEOMETRIC MODELING
AND VISUALIZATION BASED ON THE
FUNCTION REPRESENTATION OF OBJECTS**

Alexander A. Pasko, Vladimir V. Savchenko,
Valery D. Adzhiev (University of Warwick, UK),
Alexei I. Sourin (Nanyang Technological University, Singapore)

September 22, 1993



Department of Computer Software
The University of Aizu
Tsuruga, Ikki-Machi, Aizu-Wakamatsu City
Fukushima, 965 Japan

<p>Title: Multidimensional Geometric Modeling and Visualization Based on the Function Representation of Objects</p>	
<p>Authors: Alexander A. Pasko, Vladimir V. Savchenko, Valery D. Adzhiev, Alexei I. Sourin</p>	
<p>Key Words and Phrases: geometric modeling, function representation, function of several variables, set-theoretic operations, CSG, sweeping, animation, symbolic expression, visualization, hypersurface, implicit surface, isosurface, soft object, blobby model, implicit modeling, VDM specification</p>	
<p>Abstract: We represent multidimensional geometric modeling with using of a continuous function of several variables which has positive value inside, negative value outside and zero value on the object boundary. We use the theory of R-functions to implement set-theoretic operations. Function transformations are described for bijective mapping, projection, cartesian product, offsetting, blending, hypertexturing and morphing operations. This representation unifies CSG, sweeping, implicit models, voxel-based objects, deformable and other animated objects. Visualization of descriptive functions on the base of the inductive approach is discussed. Application for parallel simulation of collisions of irregularly shaped particles is presented.</p>	
<p>Report Date: 9/22/1993</p>	<p>Written Language: English</p>
<p>Any Other Identifying Information of this Report: To appear in part in the proceedings of Parallel Computing and Transputer Conference, Queensland University of Technology, Brisbane, Australia, 3th-4th November 1993</p>	
<p>Distribution Statement: First Issue: 100 copies</p>	
<p>Supplementary Notes: no</p>	

Shape Modeling Laboratory
The University of Aizu
Aizu-Wakamatsu City
Fukushima 965 Japan

Content

Preface

Chapter 1. Multidimensional geometric modeling based on the function representation: concepts and modeling system specification

1. Introduction
 2. Requirements to the F-rep based system
 3. User's geometric concepts and their function representations
 4. Internal system specification
 5. Implementation
 6. Advantages and future developments
- References

Chapter 2. Defining and visualizing descriptive functions

1. Symbolic and procedural definition of the descriptive function
 2. Methods of multidimensional visualization
 3. Inductive approach to functions visualization
 4. Ray-marching algorithm for a complex scene
- References

Chapter 3. Examples of geometric operations

1. Offsetting
 2. Blending
 3. Hypertexturing
 4. Morphing
- References

Chapter 4. Simulation of dynamic interaction between rigid bodies

1. Methods of collisions modeling
 2. Parallel algorithm description
 3. Particles collisions
 4. Example
- References

Conclusion

Appendix. Basics of the theory of R-functions

$F_Rep_Machine = Geom_Env \times Geom_Store \times Point \times Store$
 $Geom_Env = G_Ident \xrightarrow{m} Pob \mid Gob \mid Tr \mid Rel$
 $Geom_Store = G_Ident \times Number \xrightarrow{m} Arg^*$

Point :: $s_point_dim: N$
 $s_point_coord: Arg^*$

Store = $Ident \xrightarrow{m} Val$

Pob :: $s_pob_id: Pob_Ident$
 $s_pob_dim: N$
 $s_pob_par: Par^*$
 $s_pob_coord: X^*$
 $s_pob_rep: F_Rep$

Tr :: $s_tr_id: Tr_Ident$
 $s_tr_dim: N$
 $s_tr_par: Par^*$
 $s_tr_coord: X^*$
 $s_tr_rep: F_Transf$

Gob :: $s_gob_id: Gob_Ident$
 $s_gob_dim: N$
 $s_gob_tree: Gob_Tree$
 $s_gob_rep: F_Rep$

Rel :: $s_rel_id: Rel_Ident$
 $s_rel_dim: N$
 $s_rel_rep: F_Pred$

Gob_Tree :: $s_tree_left: Gob_Tree \mid nil$
 $s_tree_node: G_Ident \times (Number \mid Undefined)$
 $s_tree_right: Gob_Tree \mid nil$

$G_Ident = Pob_Ident \mid Gob_Ident \mid Tr_Ident \mid Rel_Ident$
 $Tr_Ident = Tr_U_Ident \mid Tr_B_Ident$
 $Tr_U_Ident = Tr_U_Mapping_Ident \mid Tr_U_Modulation_Ident \mid Tr_U_Projection_Ident$
 $Par = Ident$
 $Arg = Val \mid Ident$
 $Val = Undefined \mid Real$
 $Number = Intg \mid Ident$
 $N = Intg \mid Ident$
 $Ident = Token$
 $F_Rep = Token$
 $F_Transf = Token$
 $F_Pred = Token$

Figure 3: State of the F_Rep-machine (Abstract Syntax)

Inclusion relation

This relation is described as $G_2 \subset G_1$ and means that the object G_2 is included in G_1 . If G_2 is the point P the relation can be described by the following bivalued predicate:

$$S_2(P, G_1) = \begin{cases} 0, & \text{if } f_1(X) < 0 \text{ — for } P \notin G_1 \\ 1, & \text{if } f_1(X) \geq 0 \text{ — for } P \subset G_1 \end{cases} \quad (11)$$

Point membership relation

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

$$S_3(P, G_1) = \begin{cases} 0, & \text{if } f_1(\mathbf{X}) < 0 \text{ - for } P \notin G_1 \\ 1, & \text{if } f_1(\mathbf{X}) = 0 \text{ - for } P \in bG_1 \\ 2, & \text{if } f_1(\mathbf{X}) > 0 \text{ - for } P \in iG_1 \end{cases} \quad (12)$$

Note that operations of union, intersection, complement and difference, described in 3.2 correspond to operations of 3-valued logic over predicates S_3 but not to the Boolean logic over S_2 .

Interference relation

The relation is defined by the bivalued predicate:

$$S_c(G_1, G_2) = \begin{cases} 0, & \text{if } G_1 \cap G_2 = \emptyset \\ 1, & \text{if } G_1 \cap G_2 \neq \emptyset \end{cases} \quad (13)$$

There are a number of algorithms for detection of interference (collision) for implicitly defined objects. See, for example, [Duff92,Essa92].

4. Internal system specification

User's concepts defining logic-mathematical content of a system are presented in previous section. The internal specification of the system is needed at stages of system design and implementation. The formal specification given below defines principal abstract data types and operations over them. Being high level and implementation-independent the specification allows understanding the computing processes which are performed in the system.

We use the constructive technique based on Vienna Development Method (VDM) [Bjor82]. We have no possibility to give here more or less complete VDM-specification of the system (complete multilevel specification is described in [Adzh92]). Therefore, we restrict it here by a representative fragment. The limited subset of VDM is used close to the one used in [Duce87] for the small part of GKS specification. This subset has a traditional though somewhat simplified and mnemonically informative notation.

Let us introduce the concept of *F_Rep-machine* as underlying abstract machine and define its state in the form of the Abstract Syntax shown in Fig.3. The concise commentary of the main components of the Abstract Syntax follows.

A geometric environment *Geom_Env* states the sense of objects defined in the system with their names from the class *G_Ident*. These geometrically specific objects are geometric primitives *Pob*, complex geometric objects *Gob*, geometric operations *Tr* and relations *Rel*. Objects *Pob*, *Tr* and *Rel* can originally exist in the system or be defined by a user in symbolic manner during work with it.

In fact, these objects are generic types defined in the space of n-dimensions and containing the name **_Ident*, a list of formal parameters *Par**, a list of formal variables (point coordinates) *X** and an object defining function. The defining function can be represented in analytical, tabular, algorithmic and other forms not detailed here. In accordance with Section 3, the defining function for *Pob* and *Gob* is *F-rep*, for *Tr* it is a transformation of *F-rep*, for *Rel* it is a predicate.

It is important to note that operations *Tr* are subdivided into unary *Tr_U* and binary *Tr_B* ones. *Tr_U_Mapping*, *Tr_U_Modulation* and *Tr_U_Projection* are distinguished among unary operations *Tr_U* as having different procedures of computing. Note that the unary set theoretical complement is processed as the modulation.

The geometric store *Geom_Store* is a finite function intended for storing of actual parameters list of *Pob*- and *Tr*-instances. An instance is identified by a generic type name and ordinal number. Note, a binding of actual parameters identifiers with their values is fulfilled with help of a numerical *Store*.

Complex geometric objects *Gob* are generated during modeling. They contain analytical expressions (*F-rep*)

```

type inv_gob_tree: Gob_Tree × F_Rep_Machine → Bool
  let gob_tree = mk_gob_tree(gob_tree_left, <g_ident, number>, gob_tree_right) in
  inv_gob_tree(gob_tree)  $\triangleq$ 
  (  $\forall$  <g_id, numb>  $\in$  coll_nodes(gob_tree) )

  ( g_id  $\in$  dom gem_env )
   $\wedge$ 
  ( ( numb  $\neq$  undef ) ( <g_id, numb>  $\in$  dom geom_store ) )
   $\wedge$ 
  ( ( is_Gob_Ident(g_id)  $\vee$  is_Pob_Ideint(g_id) )
    ( ( gob_tree_left = nil )  $\wedge$  ( gob_tree_right = nil ) ) )
   $\wedge$ 
  ( ( ( is_Tr_U_Ident(g_id)
    ( ( gob_tree_left  $\neq$  nil )  $\wedge$  ( gob_tree_right = nil ) ) ) )
   $\wedge$ 
  ( ( ( is_Tr_B_Ident(g_id)
    ( ( gob_tree_left  $\neq$  nil )  $\wedge$  ( gob_tree_right  $\neq$  nil ) ) ) )

```

Figure 4: Well-formedness condition invariant for a complex geometric object

built by the system if necessary and the binary tree defining their structure. Fig.4 shows the invariant which states well-formedness conditions for *Gob*.

Operations over introduced objects are transition functions of *F_Rep-machine*. A list of them is quite large and relatively standard. We will discuss only one of them *f_gob_eval* which evaluates a descriptive function (1) for some instance *gob* and the given space point of generic type *Point*. The definition of this operation is given in Fig.5. It is supposed that the object *gob* is well-formed and its dimension is equal to the dimension of point (other pre-conditions are omitted). The definition of this function is reduced to the definition of the recursive function *f_gob_tree_eval*. Predicates of *is_*_Ident* kind are used to recognize the generic type of the object located in the next node of the binary tree.

If *Gob* is recognized in a leaf node, the function *f_gob_eval* is recursively applied. If *Pob* is recognized in a leaf node, the evaluation is fulfilled by sequential manner: the selection of an actual parameters list of recognized primitive instance from *Geom_Store*; binding of actual parameters with their values by means of auxiliary function *Value_list* (other auxiliary function is not defined); the substitution of values of parameters and point coordinates into the defining function *F_Rep* being selected by the selector *s_pob_rep*.

If *Tr_B* is recognized, the evaluation involves the recursive application of *f_gob_tree_eval* to the left and the right subtrees and the substitution of obtained values into the defining function *F_Transf*. If *Tr_U* is recognized, the right subtree is empty and different procedure is applied depending on the kind of an unary operation. For modulation the procedure is similar to binary operations except the right subtree processing. For bijective mapping *Tr_U_Mapping* the defining function *F_Transf* is represented by a list of inverse functions in accordance with (3). They are applied for a new point coordinates calculation. Then *f_gob_tree_eval* is applied to the left subtree with a new point. For projection the procedure is built according to (9). It involves an iterative loop with a defining function evaluation for a varying point with dimension increased by one.

When *f_gob_tree_eval* is defined, it can be used in defining functions *F_Pred* of relations and in application algorithms of visualization and integral properties computing. For example, the defining function of the point membership relation can be written as:

```

type f_gob_eval: Point  $\times$  Gob  $\times$  F_Rep_Machine  $\rightarrow$  Val
f_gob_eval(point, gob, f_rep_machine)  $\triangleq$ 
  pre is_Gob(gob)  $\wedge$  (s_point_dim(point) = s_gob_dim(gob));
  let gob_tree = s_gob_tree(gob) in
    f_gob_tree_eval(point, gob_tree, f_rep_machine);

type f_gob_tree_eval: Point  $\times$  Gob_Tree  $\times$  F_Rep_Machine  $\rightarrow$  Val
f_gob_tree_eval(point, gob_tree, f_rep_machine)  $\triangleq$ 
  let <g_id_node>, <num_node> = s_tree_node(gob_tree) and
      coord_value_list = Value_list(s_point_coord(point)) in

is_Gob_Ident(g_id_node)  $\rightarrow$ 
  let gob_node = geom_env(g_id_node) in
    f_gob_eval(point, gob_node, f_rep_machine);

is_Pob_Ident(g_id_node)  $\rightarrow$ 
  let pob = geom_env(g_id_node) and
      par_value_list = Value_list(geom_store(g_id_node, num_node)) in
    let f_pob_rep = s_pob_rep(pob) in
      f_pob_rep(par_value_list, coord_value_list);

is_Tr_B_Ident(g_id_node)  $\rightarrow$ 
  let tr_b = geom_env(g_id_node) and
      par_value_list = Value_list(geom_store(g_id_node, num_node)) in
    let gob_tree_left = s_tree_left(gob_tree) and
        gob_tree_right = s_tree_right(gob_tree) in
      let f_transf_b = s_tr_rep(tr_b) in
        f_transf_b(par_value_list, coord_value_list, f_gob_tree_eval(point, gob_tree_left, f_rep_machine),
          f_gob_tree_eval(point, gob_tree_right, f_rep_machine));

is_Tr_U_Ident(g_id_node)  $\rightarrow$ 
  let tr_u = geom_env(g_id_node) and
      par_value_list = Value_list(geom_store(g_id_node, num_node)) and
      gob_tree_left = s_tree_left(gob_tree) in

is_Tr_U_Modulation_Ident(g_id_node)  $\rightarrow$ 
  let f_transf_modulation = s_tr_rep(tr_u) in
    f_transf_modulation(par_value_list, coord_value_list, f_gob_tree_eval(point, gob_tree_left, f_rep_machine));

is_Tr_U_Mapping_Ident(g_id_node)  $\rightarrow$ 
  let f_transf_mapping_list = s_tr_rep(tr_u) in
    let new_coord_list = Value_list(f_transf_mapping_list(par_value_list, coord_value_list)) in
      let point = mk_point(s_point_dim(point), new_coord_list) in
        f_gob_tree_eval(point, gob_tree_left, f_rep_machine);

is_Tr_U_Projection(g_id_node)  $\rightarrow$ 
  let f_transf_projection = s_tr_rep(tr_u) and
      <index_i, x_i_min, x_i_max, k_interval> = Par_value(par_value_list) in
    let x_i_delta = Delta_value(index_i, x_i_min, x_i_max, k_interval) and
        x_i = x_i_min and
        old_val_pro = undef in
      for i = 1 to k_interval do
        begin
          let new_coord_list = insert_list(coord_value_list, index_i, x_i) in
            new_point = mk_point(s_point_dim(point) + 1, new_coord_list) in
              let new_val_pro = f_gob_tree_eval(new_point, gob_tree, f_rep_machine) in
                let old_val_pro = f_transf_projection(par_value_list, gob_tree_left, new_val_pro) and
                    x_i = add(x_i, x_i_delta) in
                  end
                old_val_pro;

type Value_list: Arg*  $\times$  Store  $\rightarrow$  Val*
Value_list(arg_list, store)  $\triangleq$ 
  if arg_list = nil then nil
  else conc(Value_par(hd(arg_list), store), Value_list(tl(arg_list), store));

type Value_par: Arg  $\times$  Store  $\rightarrow$  Val
Value_par(arg, store)  $\triangleq$ 
  is_Ident(arg)  $\rightarrow$  store(arg);
  is_Val(arg)  $\rightarrow$  arg;

```

Figure 5: VDM Operation of a descriptive function evaluation for a complex object

```

type pred_point_membership: Point X Gob X F_Rep_Machine -> Intg
pred_point_membership(point,gob,f_rep_machine)  $\triangleq$ 
let f_val = f_gob_eval(point,gob,f_rep_machine) in
is_equal_zero(f_val).
type is_equal_zero: Val -> Intg

```

The predicate *is_equal_zero* gives the values (0,1,2) in accordance with (12).

A geometric model built by a user is represented as a set of instances of objects *Gob*. Structures of *F_Rep_machine* components allow an easy implementation of such operations as adding, deleting and updating elements of geometric model without modification of the entire model.

The specification discussed here is the basis for the prototype system implementation described in next section.

5. Implementation

The proposed multidimensional geometric modeling system has been implemented on the IBM PS/2. The system includes a basic modeling package, a symbolic interpreter, visualization and user interface subsystems.

The basic modeling package supports complex object model construction, descriptive function evaluation at a given point and predicates evaluation.

The symbolic interpreter provides processing of input description of new primitives and operations. Symbolic expressions define a formula or a Pascal-like program. The interpreter fulfils a syntax analysis only once and transforms a symbolic expression into the internal representation in prefix notation form. Only this form is used for a descriptive function evaluation to increase calculations efficiency.

The visualization subsystem provides a polygonization of an isosurface $f(x1,x2,x3)=0$ using the algorithm of [Pask86]. This algorithm is mathematically proved and free of heuristic and ambiguities essential to other cell based isosurface fitting algorithms.

The interactive user interface is used to define parameters of primitives and operations, to form complex objects step by step, to input symbolic descriptions of new primitives and operations.

The software was applied for a computer film generation by given four "key frames" (Fig.6). "Key frames" were 3D volumes with different "classical" representations converted into the unified function representation:

Fig. 6a The solid defined by Constructive Solid Geometry (CSG). The function representation of this solid built by the system is:

$$((f_{b1} \vee f_{c1}) \wedge_0 (-f_{c2})) \wedge_0 (-f_{b2}) \geq 0$$

where functions of (x,y,z) are descriptions of the following primitives:

f_{b1} - basic block defined as intersection of six halfspaces;

f_{c1} - vertical cylinder;

f_{c2} - horizontal cylinder;

f_{b2} - subtracted smaller block;

\vee и \wedge_0 - R-functions defined by expressions (7).

Fig. 6b The swept object defined by two consequent operations:

1) the cartesian product of 2D and 1D objects:

$$(R^2 - x^2 - y^2) \wedge_0 ((z - z_0) \wedge_0 (z_1 - z)) \geq 0$$

with R-conjunction \wedge_0 used first for the segment $[z_0, z_1]$ description (see Fig.1b) and second for the cartesian product (10).