Interactive geometric modelling based on R-functions: an agent-oriented approach

V.D. Adzhiev W.M. Beynon A.A. Pasko

Abstract

This paper describes a new approach to interactive geometric modelling, based on the application of an agent-oriented modelling technique. Our methods are conceived with exploratory modelling that involves experiment and observation particularly in mind. Our geometric modelling system is based on CSG principles and uses R-functions for implementing sets of operations over functionally defined objects. This supplies a high-level symbolic description of the n-dimensional geometric model being updated during the modelling process. Our approach to building a definitive geometric language to allow incremental and extensible modelling is described. An agent-oriented notation is used to describe both the interaction of the user with the geometric modelling system and the interactions between the principal components of the system itself. A case-study that involves exploratory modelling of morphing between two CSG objects is introduced. The future development of an agent-oriented approach to geometric modelling is discussed.

Introduction

Established applications of traditional CSG systems model a static 3-dimensional object from a limited rather low-level and incomplete set of standard primitives using built-in Boolean operators. The user can typically edit the CSG representation of an object via a graphical interface by modifying or replacing one of the leaves or subtrees of its associated binary tree.

Certain categories of users feel themselves restricted when working with such systems. Without forfeiting the well-known merits of CSG representations, they would like to use a wider and more general class of primitives (e.g. swept objects, blobby models, convolution surfaces etc.) and operations (e.g. blending, morphing etc.) and—moreover—would like to introduce them during the modelling. There is also interest in increasing the dimensionality of the modelling space, as, for instance, when developing time-dependent models of interacting objects.

Finally, such advanced users are interested in modelling environments of a different character, in which it is easy to create new geometric objects and transformations, and subsequently explore their characteristics and behaviour. In this process, the modeller applies principles similar to those used in a traditional scientific investigation of a physical phenomenon [1], performing experiments with the geometric model and observing its behaviour under changeable conditions.

To meet these requirements, it is essential that the user can establish a close and intelligible relationship between the specification of an object and its observed characteristics. This requires both a high-level "user representation" and means to develop the geometric model incrementally and interactively through observation of its visualization at every step.

R-Functions as a high-level user representation for geometric objects

"Generative Modelling" [12] is a promising new approach to unified shape representation, based on multi- dimensional parametric functions, that allows textual specification of a geometric shape using a set of symbolic operators. The approach to geometric modelling described in this paper also uses a high-level representation, but is based on using so-called "R-functions" for the description of operations over geometric objects with implicitly defined boundaries.

The set of geometric objects we specify comprises topologically closed subsets of n-dimensional Euclidean space E_n defined by an inequality

$$f(x_1, x_2, \ldots, x_n) \geq 0 \qquad (1)$$

where f is a real continuous function of coordinates of the point in E_n. We refer to f as the *descriptive function* defining a certain geometric object gob_f. A descriptive function can be defined by an analytic formula, by an algorithm for evaluation or by interpolation from tabulated data. Operations on geometric objects specified as in (1) are derived from operations on their associated descriptive functions. For instance, set-theoretic operations over geometric objects for which the resulting object includes its boundary points correspond to 3-valued logic operations over point-membership predicates. Analytic definitions of such operations have been proposed and studied by V. Rvachev [10],[11] and are represented by R-functions. There is a variety of systems of R-functions, each of which has the closure property. The most frequently used system includes the following analytic expressions for R-functions corresponding to set-theoretic union, intersection and complementation:

$$f \vee_\alpha g = (f + g + \sqrt{(f^2 + g^2 - 2\alpha fg)})/(1 + \alpha)$$

$$f \wedge_\alpha g = (f + g - \sqrt{(f^2 + g^2 - 2\alpha fg)})/(1 + \alpha)$$

$$\neg_\alpha f = -f$$

where $\alpha = \alpha(f, g)$ is certain function satisfying following conditions:

$$-1 < \alpha(f, g) \leq 1,\ \alpha(f, g) = \alpha(g, f) = \alpha(-f, g) = \alpha(f, -g).$$

The form most useful in practice corresponds to $\alpha \equiv 0$.

There is also an R-function system that provides C^m continuity. As shown in [8], this representation is unified in that it is possible to specify not only the above-mentioned primary set-theoretic operations but also such operations as cartesian product, projection, bijective mapping (in particular affine mapping), some kinds of blending [9] etc. Note also that, with the help of R-functions, both algebraic and semi-algebraic geometric objects can be represented using a single descriptive function.

Computing tools for geometric modelling and visualization [7] that use R-functions in conjunction with more low-level CSG representations have already been developed. In this paper, we describe preliminary work aimed at exploiting R-function representations in

a highly flexible interactive geometric modelling system meeting the requirements set out above. To this end, we invoke a new programming paradigm that has already been successfully applied to interactive modelling and visualization with graphical images of a simpler nature [1].

R-function representations and definitive scripts

Figures 1 and 2 are specifications and images of two geometric objects. The objects are *prechair*—an approximation to a chair such as can readily be described using elementary CSG principles, as in the archetypal language PADL-2 [5], and *chair*—chosen to illustrate the complexity of object representations that can be specified using R-functions. The components of the chair, synthesized from superellipsoids, have themselves been constructed by a process of incremental modelling with intermediate step-by-step visualization.

These specifications take the form of definitive scripts [1], in which the variables represent parametrized geometric objects within a modelling space. The algebraic expression that appears on the RHS of a formula defining a geometric object, such as *gob_seat*, is its associated descriptive function. The user modifies the specification of a geometric object, changes the mode of presentation, or simulates changes to its state (as in movement of the chair), by redefining variables in the script. The characteristic property of a definitive script is that such redefinition of a variable automatically affects the values of any dependent variables. For instance, redefinition of r affects the descriptive function that defines *gob_wheels* which in turn affects the value of *gob_chair*. Note that the specification of *prechair* is essentially equivalent to a traditional CSG description of an object, such as might be formulated in PADL-2, and that redefinition within such a definitive script is in essence equivalent to editing a CSG tree.

A geometric modelling system based on definitive scripts provides an interactive environment that supports a very flexible modelling process. By redefining variables in scripts, the user can

- define the modelling space in a symbolic manner, giving its dimensionality, coordinate variable ranges, geometric types etc,
- introduce the names and descriptive functions of geometric

```
/* Built-in set-theoretic operations based on R-functions:
"|" — union, "&" — intersection, "\" — subtraction. */

x1_min = −8.0;
x1_max = 8.0;
x2_min = −12.0;
x2_max = 12.0;
x3_min = −8.0;
x3_max = 8.0;

gob_preseat = ((x1+6) & (−x1+6)) & ((x2+2) & (−x2+2))
                                  & ((x3+6) & (−x3+6));
gob_preback = ((x1+4) & (−x1+4)) & (x2 & (−x2+11))
                                  & ((x3+7) & (−x3−3));
gob_pedestal = ((x1+7) & (−x1+7)) & ((x2+11) & (−x2−7.5))
                                  & ((x3+7) & (−x3+7));
gob_preleg = ((x1+2) & (−x1+2)) & ((x2+7) & (−x2−1.5))
                                  & ((x3+2) & (−x3+2));
gob_prehole = ((x2+1) & (−x2+1)) & ((x3+4) & (−x3+4));
gob_prechair = (gob_preseat \ gob_prehole) | gob_preback
                                  | gob_pedestal | gob_preleg;
```

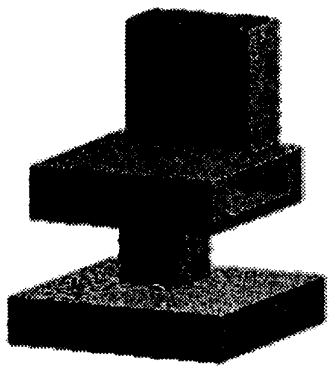

Figure 1. A geometric object constructed from blocks

```
gob_seat = 1-(x1/6)**4-((x2+3)/2)**4-(x3/6)**4;
gob_back = 1-(x1/4)**4-((x2-5.5)/5.5)**4-((x3+5)/1.5)**4;
gob_dent = 1-(x1/6)**4-((x2-6.5)/7.8)**2-((x3-1.3)/5.15)**2;
gob_arms = (((1-((x1-6.5)/0.5)**4-(x2/1.2)**4-(x3/4)**4)
          | (1-((x1+6.5)/0.5)**4-(x2/1.2)**4-(x3/4)**4))
          \ (1-(x2/0.8)**2-(x3/3.5)**2))
          | (1-((x1-6.5)/1)**4-((x2-1.6)/0.4)**4-(x3/5.5)**2)
          | (1-((x1+6.5)/1)**4-((x2-1.6)/0.4)**4-(x3/5.5)**2);
gob_props = (1-(x1/6.5)**8-((x2+5)/0.3)**8-(x3/0.5)**8)
          | (1-((x1-6.5)/0.25)**8-((x2+3)/2)**8-(x3/0.5)**8)
          | (1-((x1+6.5)/0.25)**8-((x2+3)/2)**8-(x3/0.5)**8);
gob_strut = 1-(x1/0.8)**4-((x2-4)/5.8)**4-((x3+5)/0.5)**4;
gob_leg = (1-(x1/0.5)**2-(x3/0.5)**2) & (x2+9) & (-x2-4);
gob_cross = ((1-(x1/0.5)**2-((x2+8)/0.5)**2) & (x3+7) & (7-x3))
          | ((1-((x2+8)/0.5)**2-(x3/0.5)**2) & (x1+7) & (7-x1));
gob_wheels = (1-((x1-7)/r)**2-((x2-x2k)/r)**2-(x3/s)**6)
           | (1-(x1/s)**6-((x2-x2k)/r)**2-((x3+7)/r)**2 )
           | (1-((x1+7)/s)**6-((x2-x2k)/r)**2-(x3/r)**2)
           | (1-(x1/r)**2-((x2-x2k)/r)**2-((x3-7)/s)**6);
x2k=-9.4; r=0.6; s=0.4;
gob_chair = ((gob_seat | gob_back) \ gob_dent) | gob_arms
          | gob_props | gob_strut | gob_leg | gob_cross | gob_wheels;
```

Figure 2. A geometric object synthesized from superellipsoids

objects using mathematical formulae constructed from traditional algebraic functions and R-functions.

- assign numerical values to parameters.

As explained elsewhere [1], the expressive power of representations based upon definitive scripts stems from the close correspondence that can be established between values of variables in the script and observations of the object being modelled. As the specification of *chair* illustrates, the perturbation of parameters in descriptive formulae leads to subtle changes in shape of the geometric object of the kind that a designer might wish to perform. In contrast, however accurately an object is represented by a boolean combination of basic CSG objects, there is in general no easy way to perform transformations of this nature. On this basis, the variables in an R-function representation can reflect the components of a complex geometric object much more faithfully than those in a conventional CSG representation. Practical experience with the prototype Hypersurf system also shows that the modeller can develop insight into the way in which the choice of parametrization and the redefinition of parameters affects the geometry of objects. The experimental process of redefinition and evaluation through inspection of the visual image is ideally suited for modelling tasks that involve aesthetic judgements.

Agent-oriented modelling for interaction

Our geometric modelling system is being developed within a broader framework established by our previous work on interfaces for engineering design [2], [3]. We use an agent-oriented modelling technique to specify this system. The user is an agent who constructs and observes visual images of a geometric model, and who incrementally comprehends the correspondence between the abstract mathematical model and its visualization through an exploratory modelling process. The components of the modelling system are treated as agents that interact with the user and with each other. The perspective of each agent is characterized by a particular mode of observation and particular privileges to augment and modify the model. The mode of observation determines the nature of the variables in

the definitive script. whilst the protocol for redefinition reflects the privileges.

The specification method resembles extensions to object-oriented programming for concurrent execution. such as the ACTORS model [6]. but adopts agents and observations rather than objects as its fundamental abstractions. Our approach differs fundamentally from an object-oriented paradigm in that

- message passing is only one mode of agent interaction.
- direct action of one agent upon another is possible.
- single actions typically effect state changes in several agents in one indivisible transition.

The abstract characteristics of an agent are specified using a special-purpose notation called LSD [2]. An LSD specificaton is oriented towards understanding the interaction between agents in a system by applying principles intimately connected with experiments and observations.

The variables in an LSD specification represent observations of the system such as might have to be monitored in explaining the interaction between agents. The variables associated with an agent are classified according to their status as observations with respect to the agent. These classifications are:

- STATE variables that the agent owns;
- ORACLE variables to which it responds;
- HANDLE variables that are conditionally under its control;
- DERIVATE variables representing indivisibly coupled stimulus-response relations.

The same variable can be classified in different ways by different agents. Each agent has a PROTOCOL that specifies its privileges to change the system state. subject to certain enabling conditions being met.

The nature of LSD specification can be illustrated with reference to a particular agent *visualization* that governs the way in which a

geometric object within our exploratory modelling system is displayed on the screen. Listing 1 is a simplified LSD specification for *visualization*. The period of existence of the agent *visualization* is specified by the special DERIVATE LIVE whose value is TRUE whilst the user is in the process of getting a visual image. The *visualization* agent responds to variables such as *rotate_image* and *film* as ORACLEs whose value is set by the user to determine the mode of visualization. The STATE variables include the viewing angles *alpha* and *beta*, whose very existence is indivisibly linked to visualization activity. The primary HANDLE variables are parameters *alpha*, *beta* and *x_time* that are manipulated by *visualization* in its PROTOCOL, and these indivisibly change the image of the geometric object currently under consideration as expressed by another DERIVATE.

```
AGENT visualization {
  STATE (degree)alpha=−30, beta=30, (bool)film, ...
  ORACLE rotate_image, x_time_delta, point_membership_value ...
  HANDLE
     visual_image[gname_curr],
     x_time
  DERIVATE
     (bool)LIVE = ( mode=="get_visual_image" ),
     visual_image[gname_curr] = FUNC_visualize(...)
  PROTOCOL
     ( film ) → FUNC_save_frame( ...),
     ( x[i]_geom_type == "t" ) ∧ ( xmin[i] ≤ x_time ≤ xmax[i] )
        → x_time = |x_time| + x_time_delta,
     ( rotate_image == "right" ) → alpha = alpha − delta_alpha,
     ( rotate_image == "down" ) → beta = beta + delta_beta,
     ...
}
```

Listing 1. Outline LSD specification for *visualization*

Note the use of special-purpose functions identified by the prefix "FUNC_"; these describe computational activity associated with *visualization* that establishes functional relationships in a manner that is invisible at the appropriate level of abstraction. In Listing 1, as in the other listings, it is inconvenient to itemize all the

arguments that such functions require. For instance, the parameters upon which the value of the variable *visual_image* depends include: *csg_tree[gname_curr], [x[i]_geom_type (i=1,2,.....n)], x_time, alpha, beta,* and *colour*. Note also that the redefinitions in a PROTOCOL can refer to the current value $|v|$ of a variable v (see for instance the action that updates *x_time*). In this way, redefinition is a true generalization of reassignment.

A brief review of the LSD specification

In our interactive geometric modelling system, there are two principal agents on the same level of abstraction: *user* (see Listing 4) and *modelling* (see Listings 2 and 3). At any stage, the nature of the modelling activity in progress is reflected by the variable *mode*. The *mode* variable is a STATE for *modelling* and simultaneously an ORACLE and a DERIVATE for *user*; its value is functionally determined by the user's STATE variable *user_conclusion* that is set through a special procedure, representing a human thought process, that cannot be formalized. Other significant STATE variables of *modelling* include *built_in_algebraic_functions* and *built_in_R_functions* specifying the functions that can be used in symbolic descriptive functions.

The user and modelling agents have hierarchical structures of different kinds. The following sub-agents of *modelling* can be active simultaneously: *point_in_space, geometric_object, R_csg_conversion, point_membership_relation,* and *visualization*.

The sub-agent *point_in_space* serves to represent a given point of the modelling space and to specify the characteristics of its coordinate variables. It includes STATE variables specifying the dimensionality n of the modelling space, values *x[i]* and ranges *xmin[i], xmax[i]* for the coordinate variables, and special geometric types establishing the conventions governing the semantics of coordinate variables, in particular, for visualization. The default values for coordinate variables in 4D space are: "*x*", "*y*", "*z*","*t*", but certain other types (for example, mapping values to colours) can also be introduced. The STATE variable *x_t_delta* determines the incremental interval used in animating a coordinate variable with geometric type "*t*". The DERIVATE variable *point_instance* is a list of val-

```
AGENT modelling {
 STATE
    (string)mode, (string)R_model = "α=0.0",
    (list)built_in_algebraic_functions = [ "sqrt","exp","log".... ],
    (list)built_in_R_functions = [ "|","&","\","^","@" ],
    (list)gname_list, (string)gname_curr,
    (bool)gob_exist[⋆] = FALSE, (string)message
 ORACLE
    modelling_in_progress, point_valid, fun_valid[gname]....
 HANDLE
    gob_exist[⋆], csg_tree_exist[⋆], message....
 DERIVATE
    (bool)LIVE = modelling_in_progress,
    (string)message = FUNC_message(point_valid,fun_valid....)
 PROTOCOL
    (mode == "define_new_gob") ∧ ¬ (gob_exist[gname_curr])
    → gob_exist[gname_curr] = TRUE;
       gname_list = FUNC_append(|gname_list|,gname_curr),
    (mode == "define_new_gob") ∧ (gob_exist[gname_curr])
    → csg_tree_exist[gname_curr] = FALSE,
    (mode == "eliminate_gob") ∧ (gob_exist[gname_curr])
    → gob_exist[gname_curr] = FALSE;
       gname_list = FUNC_remove(|gname_list|,gname_curr), ...
 ...
 AGENT point_in_space {
    STATE
       [ (real)xmin[i] = 0 (i=1,2,....n)],
       [ (real)xmax[i] = 1 (i=1,2,....n)],
       [ (real)x[i] = xmin[i] (i=1,2,....n)],
       (integer)n = 4,
       (point_type)point_instance, (bool)point_valid,
       x[1]_geom_type = "x", x[2]_geom_type = "y",
       x[3]_geom_type = "z", x[4]_geom_type = "t",
       x_time = xmin[4], x_time_delta = (xmax[4]−xmin[4])/10
    DERIVATE
       point_instance = [ x[1], x[2].... x[n] ],
       point_valid = ⋀ ( xmin[i] ≤ x[i] ≤ xmax[i] for i=1,2,....n)
 }
}
```

Listing 2. Outline LSD specification for *modelling*

```
AGENT geometric_object [gname] {
  STATE
    (symbolic_rep)gob_descr_fun[gname],
    (real)gob_par_list[gname],
    (csg_tree_rep)csg_tree[gname],
    (screen_picture)visual_image[gname],
    (real)gob_value[gname],
    (bool)csg_tree_exist[gname] = FALSE
  ORACLE point_instance
  DERIVATE
    (bool)LIVE = gob_exist[gname],
    gob_value[gname] =
       FUNC_descr_fun_eval(point_instance,csg_tree[gname],....)
}

AGENT R_csg_conversion {
  ...
  DERIVATE
    (bool)LIVE = (mode == "define_new_gob"),
    fun_valid[gname_curr] = FUNC_test(gob_descr_fun[gname_curr])
  PROTOCOL
    (fun_valid[gname_curr] ) ∧ ¬ (csg_tree_exist[gname_curr] )
    → csg_tree[gname_curr] =
          FUNC_create_csg_tree(gob_descr_fun[gname_curr],....);
       csg_tree_exist[gname] = TRUE
}

AGENT point_membership_relation {
  STATE (string)point_membership_value
  ORACLE gob_value[gname_curr]
  DERIVATE
    (bool)LIVE = ( mode == "evaluate_po_memb_rel" )
                 ∨ ( mode == "get_visual_image" ),
    point_membership_value =
                 FUNC_predicate_eval(gob_value[gname_curr])
}
```

Listing 3. Outline LSD specification for *modelling* sub-agents

ues of coordinate variables that is re-evaluated whenever any *x[i]* is re-defined.

Within the modelling process, many instances of the sub-agent *geometric_object* can exist simultaneously. Each instance is identified by its associated *gname*. A *geometric_object* instance has STATE variables specifying various representations of it: a symbolic *descriptive_function*, a more low-level *csg_tree*, and a *visual_image* whose value is a picture on the display screen.

The agent *R_csg_conversion* is invoked as required to create a *csg_tree* representation from a valid descriptive function, and the agent *visualization* maintains *visual_image* as a DERIVATE as explained above.

The *user* agent is composed of several sub-agents that correspond to the different modelling activities being performed according to the value of *mode*. The sub-agent *user_interpretation*, which is key for making observations and subsequent decisions on the part of *user* (see its ORACLE variables), is active at all times, and the other sub-agents can act in parallel with it. Only one such sub-agent *user_define_gob* is included in Listing 2 but others, such as *user_set_modelling_space*, *user_assign_geometric_types*, *user_set_gob_parameters*, and *user_delete_gob*, appear in our complete specification.

We illustrate how the LSD specification is to be interpreted by following a typical sequence of processes during a geometric modelling session. If *user* wants to introduce a new geometric object at some stage, it reaches the corresponding *user_conclusion* and the DERIVATE variable *mode* gets the value *define_new_gob*. At once, the sub-agent *user_define_gob* becomes active, and *user* can input the name of a new geometric object *gname_curr* and its symbolic descriptive function *gob_descr_fun*. (Note that the processes by which an actual user might convey a change of mode to the geometric modelling system—as in *choosing a menu option*, or supply input—as in *filling in a dialogue box*, are hidden in FUNC... operators at this abstract level of specification.)

Further, in accordance with its PROTOCOL, *modelling* checks whether an object with such a *gname_curr* already exists. If not, the variable *gob_exist[gname_curr]* becomes TRUE and a new instance

```
AGENT user {
  STATE
     (thought)user_conclusion = "let's_start",
     (bool)modelling_in_progress = FALSE
  ORACLE   mode
  HANDLE   modelling_in_progress, mode
  DERIVATE
           mode = FUNC_user_choose_mode(user_conclusion)
  PROTOCOL
     (user_conclusion == "let's_start")
           → modelling_in_progress = TRUE
  ...
  AGENT user_interpretation {
    ORACLE
       mode, message, point_instance,
       point_membership_value, visual_image[gname_curr],
       gname_list, gname_curr,
       gob_descr_fun[*], gob_par_list,
       [ xmin[i], xmax[i] ], i=1.....n,
       x_time, alpha,beta
    HANDLE user_conclusion,
    DERIVATE
       LIVE = (modelling_in_progress),
       user_conclusion = FUNC_user_observe_thinking(...)
  }
  AGENT user_define_gob {
    ORACLE
       mode, built_in_algebraic_functions,
       built_in_R_functions,
       gname_list, gob_descr_fun[*]
    HANDLE
       gname_curr, gob_descr_fun[gname_curr]
    DERIVATE
       LIVE = (mode == "define_new_gob"),
       gname_curr = FUNC_user_input(gname_type),
       gob_descr_fun[gname_curr] = FUNC_user_input(fun_type)
  }
  ...
}
```

Listing 4. Outline LSD specification for the *user* agent

geometric_object[gname_curr] is created. Agent *R_csg_conversion*, which is also active in this mode, simultaneously checks the validity of the *gob_descr_fun* introduced and sets the appropriate value of the variable *fun_valid[gname_curr]*. The DERIVATE *message* which appears on the display screen is set by the agent *modelling* according to the value of this variable. Provided that *fun_valid[gname_curr]* is TRUE, the agent *R_csg_conversion* creates an internal representation of the geometric object *csg_tree[gname_curr]* in accordance with its PROTOCOL. The DERIVATE *gob_value[gname_curr]* that is the value of the descriptive function at the point *point_instance* of the modelling space (the origin of the coordinate system, by default) is simultaneously evaluated. And this is the end of the chain of actions connected with this particular *mode*.

In the sequel, *user_interpretation*, responding to the *message* displayed, can set the next *user_conclusion*. Entering the mode *get_visual_image*, for example, will invoke both *visualization* and *point_membership_relation*, which is implicitly used in evaluating FUNC_*visualize*. The agent *point_membership_relation* can also be used directly when *user* wants to classify a particular point with respect to a particular geometric object. In this case, introduction of the coordinates of the point *x[i]* and the *gname_curr* leads to the indivisible sequence of evaluations: *point_instance* in *point_in_space*, followed by *gob_value[gname_curr]* in *geometric_object[gname_curr]*, then *point_membership_value* in *point_membership_relation*, and finally *message* in *modelling*.

The complete LSD specification provides a clear and concise description both of the interaction between user and modelling system and of the main processes taking place through interaction between the components of the modelling system. In particular, the descriptions of the *user* sub-agents specify what interface to the system is needed in each specific mode. Since the LSD specification indicates what agents act concurrently during the modelling process, it can also be viewed as a preliminary specification for a parallel implementation of the modelling system.

Morphing between CSG-objects as a case-study

The exploratory geometric modelling process described in this paper is illustrated by a case-study concerned with morphing between two CSG objects. Such a morphing is considered as a 4-dimensional operation. If the geometric objects gob_f, gob_g have descriptive functions f, g respectively, the resulting geometric object is gob_h, where h is defined by

$$h(x,y,z,t) = a(x,y,z,t)f(x,y,z,t)(1-t) + b(x,y,z,t)g(x,y,z,t)t$$

and a and b are positive real-valued "modulation functions" that influence features of the visual sequence of frames. Defining these functions for particular objects is a problem that can only be solved effectively through exploratory geometric modelling.

We consider morphing between *prechair* and *chair* as a particular case-study. We develop this case-study by extending the definitive script obtained by concatenating the scripts in Figures 1 and 2, and using the default geometric types as described previously. In this context, the above formula takes the form:

```
gob_sculpturing_chair = a1*gob_prechair * (1-x4) + a2*gob_chair*x4;
```

Figure 3—generated using the interactive geometric system Hypersurf [7]—depicts corresponding visual sequences of frames generated for 4 different values of the time parameter $x4$.

When we define the parameters $a1$ and $a2$ to be 1, observation of the sequence of images shows that the morphing is unsatisfactory. Inspection of the specification shows that this is due to the different "densities" of the objects. We accordingly choose $a1$ much larger than $a2$, and redefine $a1$ to be 10. This morphing is more satisfactory, but there are some undesirable features in the region of the chair arms. The form of the corresponding descriptive function and some experiments suggest the following functional form for $a2$:

```
a2 = 1+95*e**(-abs(x2));
```

with peak value at $x2 = 0$. The morphing process is now satisfactory. Of course, the process of experiment and observation can be taken further until an even better result is achieved.

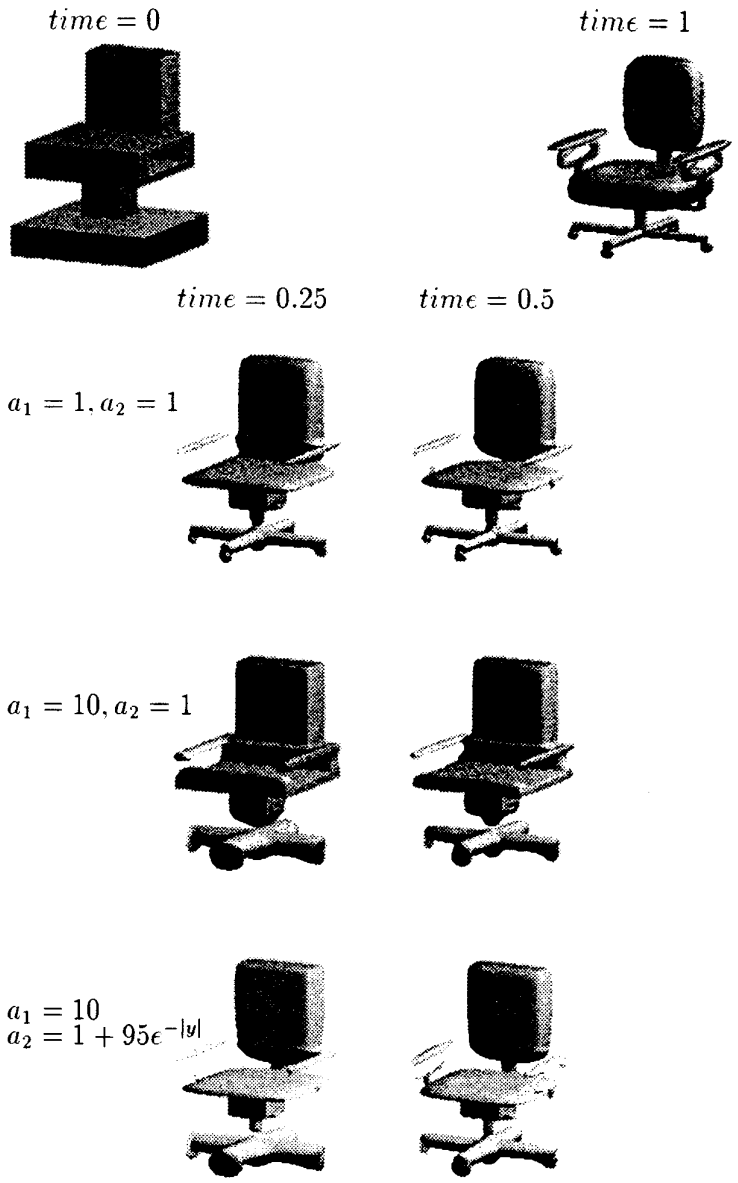

Figure 3. Four steps of the morphing process

Conclusion

In principle, R-function representations can be used to give precise and subtle specifications of complex geometric objects. The methods of exploratory modelling outlined in this paper have already made it possible to develop an impressive range of useful construction techniques. To derive the maximum benefit from the approach, we propose to develop an interactive environment, based on definitive principles, that exploits an extended set of symbolic operators (such as symbolic differentiation, search for global extremum etc.). Operators of this nature have already been introduced by Snyder [12] within a procedural programming framework.

Our LSD specification can be interpreted operationally using definitive scripts to represent computational state. We are currently developing a sequential implementation in the interactive environment supplied by EDEN [3]. An alternative approach, better suited for a parallel implementation, is to exploit the concurrency in the specification in the computational framework of the Abstract Definitive Machine [4].

A complementary concern is that of exploiting our exploratory modelling technique in a design context. For this purpose, we must provide a practical interface for a designer who has no specialist mathematical knowledge, and make it possible to construct R-function representations from object descriptions better suited to conceptual design. It is particularly important to make provision for references to characteristic points and special features of objects, and to enable the user to develop suitable R-function representations based on these. Issues for future research include effective methods for generating R-function representations:

- to reflect the construction of an object from a component hierarchy—as in separating a chair into the parts that would be separately manufactured,
- to capture the functionality of geometric objects, so that redefinition of parameters in a script can be used to represent physical operations—as in rotating or tilting the seat of a chair,
- to develop exploratory modelling techniques combining shape modelling with complex motion using a set of special time-

dependent transformations—as in investigating the problems of moving a chair from one room to another.

Much previous related work has been done on the use of definitive scripts in conjunction with agent-oriented modelling in engineering design [2]. The advantages of using such a modelling paradigm in conjunction with simple 2-dimensional visualization techniques have been demonstrated in several case-studies, and the possibility of generalizing this work to higher dimensions has already been investigated. The research described in this paper has close points of contact with the design and partial implementation of a definitive notation for geometric modelling CADNO as first introduced in [3]. CADNO is centrally concerned with developing an abstract framework within which many different ways of specifying and referencing geometric objects can be unified. The versatility and homogeneity of R-function representations suggests that they are well-suited to complement the object representations conceived in CADNO.

Acknowledgements

We are grateful to Steve Russ and Yun Pui Yung for useful discussions relating to this paper, and to Alexei Sourin and Nick Holloway for software support. We are indebted to the Royal Society for supporting Valery Adzhiev under its Postdoctoral Fellowship scheme, and to the SERC for financial support under grant GR/J13458.

References

[1] W.M. Beynon, Y.P. Yung, A.J. Cartwright, P.J. Horgan, Scientific visualization: experiments and observations, *Proc. 3rd Eurographics W/S on Visualization in Scientific Computing*, Viareggio, (157–173), 1992.

[2] W.M. Beynon, I. Bridge, Y.P. Yung, Agent-oriented modelling for a vehicle cruise control system, *Proc. ASME Conf. ESDA '92*, Istanbul, (159–165), 1992.

[3] W.M. Beynon, A.J. Cartwright, A definitive programming approach to the implementation of CAD software, *Intell. CAD Systems II: Implementation Issues*, Springer Verlag, (126–145), 1989.

[4] W.M. Beynon, M.D. Slade, Y.W. Yung, Parallel computation in definitive models, *Proc. CONPAR'88. BCS Workshop Series*, CUP, (359-367), 1989

[5] M. Hartquist, *PADL-2 User Manual*, 1983.

[6] C. Hewitt, Viewing control structures as patterns of passing messages *Artificial Intelligence*, 8, (323–364), 1977

[7] A.A. Pasko, V.D. Adzhiev, I.A. Prostakov, Multivariate function visualization: the inductive approach, *Proc. 3rd Eurographics W/S on Visualization in Scientific Computing*, Viareggio, (303–316), 1992.

[8] A.A. Pasko, V.V. Savchenko, V.D. Adzhiev, A.I. Sourin, Multidimensional geometric modeling and visualization based on function representation of objects, *Tech. Report 93-1-008*, Dept. of Computer Software, The University of Aizu, Japan, September 1993.

[9] A.A. Pasko, V.V. Savchenko, Blending operations for functionally based constructive geometry, *ibid.*

[10] V.L. Rvachev, On the analytical description of some geometric objects, *Doklady of Ukrainian Academy of Sciences*, **153**, 4, (765–767), 1963.

[11] V.L. Rvachev, *Methods of Logic Algebra in Mathematical Physics*, Naukova Dumka Publishers, Kiev, 1974.

[12] J.M. Snyder, *Generative Modeling for Computer Graphics and CAD*, Academic Press, 1992.