"HyperJazz" project: development of geometric modelling systems with inherent symbolic interactivity

V.D.Adzhiev A.A. Pasko A.V.Sarkisov

Abstract

The paper presents a project called 'HyperJazz' devoted to development and implementation of the 'empirical' approach to solid modelling, which stresses the importance of interactive experiments of the user with the model. The approach is based on two foundations: geometric modelling based on the function representation and definitive programming environment. We propose a technology of user's interactive work with the modeller and a technology of creating interactive geometric modelling systems that can be adapted to different application domains and can be customized for needs of particular users. The basic set of API-procedures and the Kernel Geometric Interactive Environment are introduced. Their charasteristic feature called "inherent symbolic interactivity" is based on a high-level definitive language of point-sets. The technology of building a particular CSG system as well as an example of behaviour modelling are presented.

Introduction

It was recently pointed out once more in [5], "there are still many useful shapes and processes wholly or partially beyond the reach of existing solid modelling techniques". It becomes especially evident while solid modelling techniques in general, and CSG ones in particular, expand more and more to be exploited within applications beyond mechanical CAD. Examples of this can be found in animation, virtual reality, physically-based modelling, particles systems, computer art, and entertainment applications. The specifics of such applications dictate capturing objects' complex behaviour with spatial interference varying under changeable external conditions. A particularly important case of such behaviour is the variation of objects' geometric shapes dependent on modelling contexts.

To accept this challenge, it is necessary to provide the means for extending the standard set of primitives and operations that can be very diverse for different applications. However, this is not enough. That is why, meeting new demands, the very solid modelling itself is in transition from an established design paradigm to a new one [11].

Classical design paradigms concentrated on obtaining one specific final shape become supplanted by feature-based, constrained-based ones oriented more towards defining families of shape instances. Such new "parametric" modelling systems emphasize importance of the design process and suppose presence of high-level, parameterized, user-modifiable, non-evaluated definitions that make up a representation that called "editable" [17, 10]. Some authors (e.g., Chen and Hoffmann [7]) argue that the value of such representations is determined by extent of independence of a specific core modeller and by the flexibility for adapting to new application domains.

As distinct from the mentioned papers, mainly considering mathematical, algorithmical and structural problems, we will concentrate on the very process of modelling. Traditionally, it supposes that first the user conceives a comprehensive model specification and transfers it to the computer through a textual or graphic interface; then the modeller (integrated with built-in computing and rendering procedures), working actually as a calculator, produces a result which usually appears to be quite as expected. There may be some minor defects that can be eliminated by changing some numerical parameters or adjusting simple constraints [16].

Even modern parametric modelling systems follow this scheme directed (as noticed by Shapiro and Vossler [17]) mainly at the means of creating (but not updating) models. It is especially evident in the case of structurally or dynamically complex models, when the user is not able to predict all details of model's behaviour and even to define exactly which variables will later be a subject to change.

Taking into account these problems, we are principally developing another approach to the modelling process that we call "empirical (or exploratory) geometric modelling" [2], which stresses the importance of interactive experiments of the user with the model. The experiments interlace two viewpoints of modelling through observations of the model's state. One is the user's intentional specification which can be incompletely preconceived and ready to be refined during a modelling process in a step-by-step manner. The other viewpoint is the modeller's external representation of an object. Within such a framework one should consider interactions during the modelling session on different levels of the system

"user + modeller + model" between its all constituents.

This paper is the next one in a sequence of our works argueing combining the empirical interactive framework with the modeller based on function representation (F-rep) [15]. The paper [13] contains a description of an interactively extended geometric modeller; in [1] we provided the interactive modelling system specification made using agent-oriented LSD notation, describing the interactions of the user with the F-rep modeller and between principal components of the modeller itself; in [15] we have assessed principal features of a user representation based on a high-level geometric language.

Here we propose a technology of user's interactive work with the modeller and a technology of creating interactive geometric modelling systems that can be adapted to different application domains and can be customized for needs of particular users. Both technologies heavily use the feature called "inherent symbolic interactivity" and are being implemented within the project "HyperJazz". The central point is building a Kernel Geometric Interactive Environment with an underlying editable non-evaluated representation based on a high-level definitive language of point-sets. The corresponding implementation and examples of applications are described.

Related approaches

The following approaches to specifying geometric models and to organizing the corresponding "user-modeller" interactive process are significant in some aspects, and so have influenced our work.

"Djinn" [5] is a project concerning geometric application programming interfaces for solid modelling. It is declared to be independent of specifics of underlying modellers by using geometric operations on point-sets, whilst hiding data-structures and providing methods to interrogate them, which allow the meeting of requirements of a wide range of applications. The Djinn API, being a set of procedures supporting calls to the underlying modeller, deals mainly with functionality where geometric scope is limited by 3D point-sets. The project itself has nothing to do with the interactive technology of user's work, obviously supposing the use of traditional one with conventional textual or graphical interfaces.

Contrary to such a "procedural" approach, "Generative Modelling" supports a symbolic approach to modelling a variety of shapes, which allows the model to "capture the designer's intent without approximation error" [18]. The corresponding prototype system GEOMOD has a rich set of symbolic operators on parametric functions.

The built-in C interpreter supposes the use of a traditional interactive technology, which has well-known problems concerned with capturing and handling the operation of objects' behaviour.

Hoffmann and collaborators [10, 7] present a feature-based and constraint-based solid design system based on a concept of a layered representation of a design. It means that design intent is stored in a high-level unevaluated "Editable Representation" (Erep). The geometry to which Erep corresponds, for specific constraints values, is stored in an evaluated representation, for instance as B-rep. The user interacts through GUI with a visual presentation of the evaluated model; the design operations are abstracted and recorded in Erep.

van Emmerik et al [9] describe the "hypertext approach" to interactive solid modelling combining a top-down conceptual design process and iterative bottom-up adjustment of design details and parameters on the base of three paradigms: integration of data structures and control structures, multiple view interface, and use of hypertext techniques for fast traversal and selective display of data. The interface enables the user to view and edit both the cell (the single modelling entity) and the hierarchical structure of the model providing integrated textual, 2D graphical, and 3D direct-manipulation techniques. The versatility of the approach is illustrated by the fact that the system can be used for designing the system's internal direct manipulation devices (this has to be done by the system implementer rather than by the end user!). Note also that objects can be instantiated before their definition is given.

Definitive SUI vs GUI

Interactive graphical interfaces (GUI) that enable the user to specify the model by direct manipulation, supplied with facilities for giving geometric constraints, have been supplanting symbolic user interfaces (SUI) being usually implemented on the base of a procedural modelling language. It has been determined by well-known drawback being considered as inherent for SUI, namely - the lack of genuine interactivity (more detail in [8, 9]). Nevertheless, we see a potential for SUI that is based both on needs of new applications and on using the definitive programming paradigm enabled to provide desirable interactivity.

The major drawback of GUI (even if integrated with a subordinated textual view) is a limited design vocabulary. It is suitable for limited range of primitives (e.g., the standard set of CSG primitives), transformations (affine or such as extrusion) and constraints (simple geometric relations such as lines' parallelism). This is quite natural because GUI must be based on simple graphical metaphors.

This problem becomes especially significant, if it is necessary to define complex behaviour of non-trivial (e.g., time-dependent) objects as some applications require.

As to SUI with a high-level language, it can in principle have richer functionality because it can rely on the underlying algebraic system with arbitrary objects, transformations and relations (F-rep is especially suitable for defining such entities, including interactive extending built-in ones [15]).

Then, a symbolic representation can be used as a high-level geometric exchange protocol independent of specific modellers. Another perspective application is concerned with "distributed shape modelling". Here, interactively entered lines of the symbolic description can serve as a network geometric protocol between the user and the remote modeller within WWW (VRML language can be used for output model coding).

Our experience shows that advanced users (modelling systems designers, mathematicians, etc.) are especially inclined to using SUI. It also seems to be valuable in the educational perspective because working with SUI lets students understand and master underlying laws of geometric computing.

Of course, ordinary users will prefer GUI that should be provided if possible. But here, the symbolic representation would serve as interface between GUI and the modeller.

As to the ideology of "symbolic interaction", it relies upon the definitive programming paradigm [4]. It is based on the use of definitions for interactions. The network of definitions ("definitive script") is very suitable for representing the changeable state of dialogue. The combination of implicit and explicit definitions allows both persistent relationships and transient values to be recorded and recovered by interrogating the variables to obtain their current definitions. The expressive power of representations based upon definitive scripts stems from the close correspondence between values of variables in the script and observations of the object being modelled. We consider this paradigm as a proper base for the implementation of the conception of empirical geometric modelling.

F-rep modeller and previous systems

We use a F-rep modeller, which, as we substantiated in [15], provides a higher abstraction level as regards to other known representations. It supports combinations of different representational styles such as constructive geometry, sweeping, soft objects, voxel-based objects, deformable and other animated objects. All objects irrespective of their nature or complexity are represented by the in-

equality $f(x_1, x_2, ..., x_n) \geq 0$, where f is a real continuous function of Cartesian coordinates of a point.

F-rep, having the closure property, naturally captures multidimensionality allowing, in particular, uniform description of both static and time-dependent objects. Giving "geometric types" to coordinate variables x_i provides interactive change of the interpretation of their semantics. In particular, this is important for producing different kinds of visual images.

As to our previous interactive geometric modelling systems, Hypersurf [13] had a built-in point-sets oriented high-level language that was, however, not definitive. There have been a few projects of applying a definitive approach to graphics aside using F-rep, e.g., in [3], but geometric modelling functionality remained rather poor. In [1] we showed that a F-rep geometric modeller and a definitive interactive environment can be integrated with joint benefits. The CADNORT project [6] based on the universal definitive environment EDEN was the first experience of developing a definitive language for F-rep geometric modelling, with limited functionality. This paper represents our current project called "HyperJazz".

Towards a Kernel Geometric Interactive Environment

Within our project, we consider as significant the following requirements for empirical geometric modelling:

• a particular application can require a specialised modelling system with a unique set of objects (primitives), operations (transformations) and relations;

• it would be convenient, if diverse modelling systems were generated on the base of the same kernel modeller that, in its turn, is built of limited (though extendible) collection of instrumental procedures;

• the user should be able to specify diverse geometric shapes in terms of point-sets. This necessarily supposes a high-level "user representation" in the form of an extendible, highly interactive programming language;

• the interactive interface technology for the work of different users with the system should be similar; it should be possible to reach not only the kernel modeller but also a general geometric and interactive environment in which the kernel modeller is immersed;

• the users' views can differ from each other not only because of application specifics, but also because of users' distinct roles, intents, and privileges within possibly the same application field. In particular, one can single out the following types of users: an

"implementer" who builds a quite generic (kernel) modelling system on the base of a general geometric environment; an "analyst" who is an expert in a certain application domain and creates a specific modelling system on the base of generic one; an "end user" who just uses the system as created, though can in principle change it in some item.

Ultimately, every user can have and exploit his or her personal modelling system. The process of creating such a system can be treated as a continual, subtle fine tuning of a geometric "instrument", starting from an initial geometric modelling environment containing a minimum set of built-in operators and relations, towards a specialist system.

Proceeding from these premises, we introduce the notion of a "kernel geometric interactive environment" as an instrumental tool for developing diverse modelling systems. The environment has an inherent potential for integrating functionality of the geometric modeller, user's interactive activity and application context on the base of a unified conceptual and technological framework. As a "kernel", the environment should be as much as possible independent of any concrete application or computational platform.

It can be reached, if the environment inherits from instrumental API-procedures (with help of those it is built) the feature that we call "symbolic interactivity". It means that all significant interactions take place on the base of entering redefinitions as symbolic high-level statements. They result in indivisible propagation of modification of all dependent model's constituents that is reflected in the internal lower-level, yet unevaluated (though ready to effective evaluation) representation in the form of a generalized CSG-tree. It is true even for the level of instrumental API-procedures. If the kernel geometric interactive environment has been created on the base of these procedures, updating an internal tree can result in automatic evaluating corresponding real functions with generating an updated visual image.

So, the basis for symbolic interactivity is a new definitive geometric language. It is an instrument not only for describing the user's model but also for generating diverse modelling systems.

Definitive language HyperJazz

HyperJazz, treated as a high-level "user representation", is intended for specifying a user's geometric model as a definitive script. Simultaneously, it is a tool for the implementation of experiments that are not preconceived with the model. It has two main features:

1) It covers all the concepts of the F-rep framework [15]. Accordingly, there are such entities as geometric objects, geometric

transformations (operations), and geometric relations described by their defining functions with a mechanism of complex objects synthesis. The notions of "modelling space" and "geometric types" are also used in the language.

2) It is definitive, incorporating the means to set persistent relationships between the language's variables (that are all geometric entities or any their constituents), and indivisible propagation of change through all dependent variables of a certain variable that is redefined.

To specify the model, the user first devises an "underlying algebra" of geometric objects, operations and relations, which reflects the universe of discourse. Then, corresponding generic entities and their instances, with variables that are the subject for experiments are introduced in a HyperJazz script.

As we parametrise generic geometric objects, transformations and relations passing parameters by value, we introduce instances of them having their own names derived from the generic ones. A generic entity contains a few sections:

• `geom:` - variables definitions that are associated with the particular instance and are independent of coordinate variables (that is to say they are constant for the instance given);

• `inst:` - creating instances of other generic entities;

• `coord:` is used if a transformation changes coordinate variables;

• `def:` - this being obligatory is a sequence of definitions of local variables eventually providing the float value of the defining function for the entity depending on coordinate variable X. The RHS of a local variable definition is an arbitrary expression that is built using other variables and instances of geometric objects. It includes built-in set-theoretic operators ('|', '&', '\', '@'), standard mathematical functions ('max', 'min', 'sin', 'log', 'exp', etc.) and instances of generic geometric transformations. Here in addition, operators such as 'if-then-else', as well as evaluating instances of geometric relations can be used. To capture more computational power, there is also the possibility of calls of external procedures written in C;

• `attr:` - non-geometric attributes attached to the entity (e.g., 'colour', 'text', etc.).

All the generic entities are defined within a common "model" environment. Each generic entity has its own local environment. There are simple syntactical conventions about using variables' names that allow all the variables, including defined as local, to have global scope. Some variables can be introduced as "undefined" to be defined later, beneficial in the process of interaction.

HyperJazz is actually a language of point-sets. Being a high-level

user representation, it avoids explicit reference to model data structures (cf. [5]). The underlying internal representation is created as a collection of structures representing instances, each of these referencing the corresponding generic entity. Resulting defining functions of each generic entity are represented in a generalised CSG-tree. The tree's leaves can contain references to instances of other geometric objects. The tree's nodes correspond to transformations or external procedures. A VDM specification [14] contains more detail about the discipline of processing such a tree with transformations of different types (set-theoretic, affine, bijective mapping, projection etc.). This internal tree reflects the unlimited hierarchy of geometric objects (not to be confused with the only two-level hierarchy of model and local environments!).

Any local variable of generic entities can be considered as "definitive". This means that it is possible to redefine it. Note that we can redefine both variables that are defined within a generic entity (all instances of this generic object will change) and those that are defined within the concrete instance (other instances of the parent-generic as well as generic entity itself will not change). Moreover, these redefinitions themselves can be within a common "model" environment and within any local environments. Note that the `def:` section of the corresponding generic entity can contain evaluations of certain relations, depending on which defining functions of other objects can be redefined. This provides the geometric objects influencing each other as active agents depending on the current modelling context.

The central point of the entire approach is that each redefinition results in an indivisible chain of modifications of internal representations by rebuilding references to all dependent variables. If this modification becomes operationally visible to the user, we can talk about "inherent interactivity" as a property of all HyperJazz-based environments. The language is a quite powerful tool giving the user a carte blanche for the improvised playing with the model (see the language name!).

Fig.1 contains a HyperJazz protocol of an interactive process for a simple illustrative example. Initially we define a generic cube and then its two intances. Let us suppose that we want to get smooth edges of the cubes. We define a generic transformation "blending intersection"[15] and its two instances. Then, we redefine the local variable `px` of the instance `Gcube_1`. Another instance cannot be modified because the generic cube was not affected. The second redefinition changes the local variable `pxy` of the generic cube. Now both instances will be modified.

```
Gcube (a,x0,y0,z0) {
 def:
  px = (x1-x0+a) & (-x1+x0+a);
  pxy = px & (x2-y0+a) & (-x2+y0+a);
  f = pxy & (x3-z0+a) & (-x3+z0+a)
}
Gcube_1(5,0,0,0);
Gcube_2(10,20,0,0);
...
Tblendint (a1,a2,a3) : Gname1,Gname2 {
 def:
  d = a1/(1 + (Gname1/a2)^2 + Gname2/a3)^2);
  f = Gname1 + Gname2 - sqrt(Gname1^2+Gname2^2) + d
}
Tblendint_1(1,0.5,-0.5);
Tblendint_2(1,-0.3,0.5);
...
Gcube_1.px = Tblendint_1(x1-x0+a, x1+x0+a);
...
Gcube.pxy = Tblendint_2(px, (x2-y0+a)&(-x2+y0+a) );
...
```

Figure 1: An example of a protocol of an interactive process

Instrumental tools for building kernel geometric interactive environments

This section contains a brief description of a small collection of procedures that can be considered as both API and a F-rep modeller. These procedures provide the means for building kernel geometric interactive environments. With such a system the user will make experiments with the model being created by entering definitions or redefinitions using one or a few statements in the HyperJazz language.

1. `Jazz_create( "(re)definition lines in HyperJazz" )`. This procedure, in correspondence with the symbolic line(s) of HyperJazz (that is its calling parameter) either creates a new element of the internal model representation (if new variable or entity is defined) or modifies an "old" variable or structure with indivisible propagation of changes to all dependent variables.

2. `Jazz_f_eval( "name of geometric object", X )`. This procedure evaluates the value of the real defining function for the geometric object in the given point X.

3. A few procedures providing interrogation of the internal model representation:

- `Jazz_var_def( "name of generic entity/instance" )` returns names and current values of all local variables for the given geometric generic entity or instance;

- `Jazz_var_undef()` returns names of all undefined variables;

- `Jazz_attr( "name of geometric object" )` returns names and values of non-geometric attributes for the given geometric object.

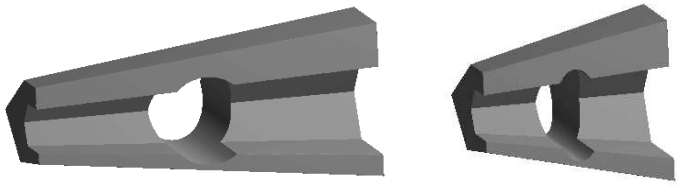
4. Of course, it is necessary to have visualization procedures. Considering that it is a matter of principle for the approach to have quick visual response, we include the procedure of polygonisation of an iso-surface $f(x_1, x_2, ..., x_n) = 0$ [12]. It, in addition, can be easily tuned for parallel architecture.

5. Some computational procedures dealing with the internal representation.

Note that all these procedures are written in C language, and so are independent of any computational platform. They can be used in any application program. Of course, to get real benefits from the "inherent interactivity", it is necessary to build an interactive system. In such cases, procedures handling scripts with tools providing multi-windowing facilities are necessary. Among such procedures, one should mention those for saving symbolic descriptions of generic geometric entities that helps to create libraries of geometric objects, transformations and relations relevant for the particular specific applications.

Having the set of procedures described above (note that it is really a small set, in the utmost case we can leave only two procedures `Jazz_create` and `Jazz_f_eval`), it is possible to easily build a kernel geometric interactive environment that contains no built-in geometric primitives and a few built-in operations, such as set-theoretic, and relations, such as point-membership or inclusion. This environment should provide multi-view facilities. The minimum set is a window for a visual image and a window for entering symbolic lines in HyperJazz. Note, the latter one contains full history of the modelling process. Another textual window can contain only current values of variables, etc.

This is a basis for generating particular geometric modelling systems, each with the same basic property of symbolic interaction stemming from the nature of the definitive programming environment. All procedures mentioned in this section have been implemented in ANSI C. The kernel geometric interactive environment has been implemented under MS Windows. It is expected to be compatible with any further implementation on different platforms because of the use of symbolic HyperJazz scripts.



(a) Initial solid (b) Stretched solid

Figure 2: Stretching a set-theoretic solid

Case study

We illustrate the proposed technology with the help of case study showing a way how "implementer", "analyst" and "end user" create and use a particular application-oriented CSG system. The problem is to support the following sequence of modelling steps (see Fig.2): set-theoretic operations on convex 2D polygons, extrusion of a resulting 2D shape, tapering a resulting 3D solid, 'drilling' cylindrical holes in the tapered solid and finally stretching it. This sequence of operations seems to be very simple, but its implementation with some conventional solid modellers in practice is difficult or not possible [19]. One of the reasons is that operations like stretching and others are not supported on the level of CSG solids. The HyperJazz code for this example is presented in Fig.3. We demonstrate here that arbitrary primitives and operations can be defined by the user in a few lines of code. External C functions can be called also, for example such a function is used here to implement a generic polygon.

Fig. 4 contains an example of behaviour modelling. An interectively entered HyperJazz script defines the following model: two spheres are given; when a collision event of the spheres is detected, the first sphere transmutes itself into a cube. This example shows that it is possible with a similar model of time-dependent objects to simulate complex behaviour of active geometric agents.

Conclusion

As Djinn's developers [5], we can declare: at the current stage the HyperJazz "is intended to be a tool that will help modelling re-

```
// the implementer provides the interactive system, adding to the "empty" kernel
// environment the C-procedure Fpolygon and generic objects Ginfcylx and Gpolygon2D

// Generic object "infinite cylinder" along x - axis

Ginfcylx(rc)
{ f=rc^2-x3^2-x2^2
}

// Description of the generic "polygon"

Gpolygon2D(Lp)
{ f=Fpolygon2D(Lp)
}

// the analyst defines the generic operations Tstretch3Dz and Ttapering3Dz

// Generic transformation "stretch" along z - axis

Tstretch3Dz(scale,zstr):Gname
{ coord: x3'=zstr+(x3-zstr)/scale;
  def:   f=Gname.X'
}

// Generic transformation "tapering" along z - axis

Ttapering3Dz(ztap1,ztap2):Gname
{
 coord: x1'=2*x1/(1+stap);
        x2'=2*x2/(1+stap);
 def:   stap=(ztap2-x3)/(ztap2-ztap1);
        f=Gname.X'
}

// The end user can load and use now generic objects
// provided by the implementer and the analyst

// Description of the generic final object

Gpart()
{ inst: Ginfcylx_n(2);
        Gpolygon2D_1(.Lp1);
        Gpolygon2D_2(.Lp2);
        Ttapering3Dz_c(.zt1,.zt2);
        Tstretch3Dz_s(.scales,.zt1);

  def:  polygonc=Gpolygon2D_1\Gpolygon2D_2;
        extr=polygonc @ ((x3-.zt1)&(.zt2-x3));
        poltag=Ttapering3Dz_c(extr);

        f=Tstretch3Dz_s(poltag\Ginfcylx_n)
}

// defining variables

Lp1=[-5, 0, -1, -4, 4, -3, 3, 4, 0, 4];
Lp2=[1, -2, 5, -2, 5, 2, 1, 2, 0, 0];
zt1=-8;
zt2=8.;
scales=0.5;

// instance of the final object

Gpart_1();

// now the user can visualize Gpart_1 and make experiments redefining variables
```

Figure 3: A HypeJazz code for an application example

```
// the example of behaviour modelling as a script for
// the interactive session entered line by line

// r is a "global" variable

  Gbody (x0,y0,z0) {
     f = .r^2 - (x1-x0)^2 - (x2-y0)^2 - (x3-z0)^2
   }
// defining two instances as two spheres with the same radius

   Gbody_1(x01,y01,z01);
   Gbody_2(x02,y02,z02);

// defining some value of radius as a definitive parameter

    r = 5;

// defining the origins of the spheres

     x01 = -11;
     y01 = 0;
     z01 = 0;

     x02 = 10;
     y02 = 0;
     z02 = 0;

// redefinition of the defining function of the 1st body with using
// the built-in relation "intersection of two objects" rint

     Gbody_1.f = if (rint(Gbody_1, Gbody_2) = 1) then
                      (x1-x0+.r/2) & (-x1+x0+.r/2) &
                      (x2-y0+.r/2) & (-x2+y0+.r/2) &
                      (x3-z0+.r/2) & (-x3+z0+.r/2)
                 else
                    .r^2 - (x1-x0)^2 - (x2-y0)^2 - (x3-z0)^2;

// the experiments: changing the definitive parameter's value

      r = 10.;

// as we can observe the visual image that appears automatically, we can
//  see that there is no a "collision event"; so, we input another value:

      r = 12.;

// the "collision event" takes place;
// a cube appears instead of the first sphere
```

Figure 4: An example of behaviour modelling

searchers". Also, this "project is itself research". Of course, ordinary end users could be unsatisfied with "symbolic interactivity" at the user interface level and would require GUI. An interesting research topic is how to generate such GUI, content and features of which are concerned with the specifics of a concrete model. We also plan to implement a WWW-based version of the kernel geometric interactive environment with VRML models as output.

Acknowledgement

We are greatful to Richard Cartwright from the University of Warwick for fruitful discussions. Our thanks to Carl Vilbrandt from Applied CAD/ART for his application example.

References

[1] V.D.Adzhiev, W.M.Beynon, A.A.Pasko "Interactive geometric modelling based on R-functions: an agent-oriented approach", *Set-theoretic Solid Modelling: Techniques and Applications, CSG'94 Conference Proceedings*, Information Geometers, Winchester, UK, (253-272), 1994.

[2] V.D.Adzhiev "Agent-oriented approach to exploratory geometric modelling based on function representation", *Proceedings of the International Workshop "Shape Modelling: Parallelism, Interactivity and Applications"*, The University of Aizu, Japan, (64-68), 1994.

[3] M.Beynon, E. Yung "Implementing a definitive notation for interactive graphics", *New Trends in Computer Graphics*, ed. N.Magnenat-Thalman, D.Thalman, Springer-Verlag, (456-468), 1988.

[4] W.M.Beynon "Evaluating definitive principles for interaction in graphics", *New Advances in Computer Graphics, Proceedings of CG International'89*, (291-303), 1989.

[5] A.Bowyer et al. *Introducing Djinn. A geometric interface for solid modelling*, The Geometric Modelling Society and Information Geometers, 1995.

[6] R.I.Cartwright "CADNORT", *3rd year undergraduate project*, Department of Computer Science, Warwick University, Coventry, UK, 1994.

[7] X.Chen, C.Hoffmann "Design compilation of feature-based and constraints-based CAD", *Proceedings "Solid Modeling'95"*, Salt Lake City, ACM, (13-19), 1995.

[8] M. van Emmerik "Interactive design of 3D models with geometric constraints", *The Visual Computer*, **7**, 5/6 (309-325), 1991.

[9] M. van Emmerik, A.Rappoport, J.Rossignac "Simplifying interactive design of solid models: a hypertext approach", *The Visual Computer*, **9**, 5 (239-254), 1993.

[10] C.Hoffmann, R.Juan "Erep - an editable, high-level representation for geometric design and analysis", in *Geometric Modeling for Product Realization*, Elsevier Science Publishers B.V., North-Holland, (129-164), 1993.

[11] C.Hoffman "Solid modeling", *CRC Handbook of Discrete and Computational Geometry, to appear*, 1995.

[12] A.Pasko, V.Pilyugin, V.Pokrovskiy "Geometric modeling in the analysis of trivariate functions", *Computer and Graphics*, **12**, 3/4 (457-465), 1988.

[13] A.A.Pasko, V.V.Savchenko, V.D.Adzhiev "Educational project: Empty Case technology of geometric modelling", *EDUGRAPHICS'93 and COMPUGRAPHICS'93, Combined Proceedings*, ACM, Portugal, (6-11), 1993.

[14] A. Pasko, V. Savchenko, V. Adzhiev and A. Sourin "Multidimensional geometric modeling and visualization based on the function representation of objects", *Technical Report 93-1-008*, The University of Aizu, Japan, 1993.

[15] A. Pasko, V. Adzhiev, A. Sourin and V. Savchenko, "Function representation in geometric modeling: concepts, implementation and applications", *The Visual Computer*, **11**, 8(429-446), 1995.
http://www.u-aizu.ac.jp/public/www/labs/sw-sm/FrepWWW/F-rep.html

[16] A.A.G.Requicha, J.R.Rossignac "Solid modeling and beyond", *IEEE Computer Graphics and Applications*, **12**, 5 (31-43), 1992.

[17] V.Shapiro, D.Vossler "What is a parametric family of solids", *Proceedings "Solid Modelling'95"*, Salt Lake City, ACM, (43-54), 1995.

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

[19] C.Vilbrandt, Applied CAD/ART (Eugene, Oregon USA), *private communication*, 1995.