HyperFun project: a framework for collaborative multidimensional F-rep modeling

Valery Adzhiev [1,2], Richard Cartwright [3], Eric Fausett [1],
Anatoli Ossipov [2], Alexander Pasko [1], Vladimir Savchenko [1]

Abstract

This paper presents a project devoted to developing an open system architecture for functionally based (implicit or more generally F-rep) shape modeling and its applications. The software tools are built around the shape models written in a high-level programming language called HyperFun. A model in HyperFun can serve as a protocol for exchanging F-rep models between users, modeling systems, or networked computers. HyperFun models can be collected in application-specific libraries. We describe the basic set of system components: an interpreter for parsing and function evaluation; F-rep system libraries; a modeler with an extendable graphical user interface; a multidimensional modeler with a symbolic user interface providing means for interpreting multidimensional coordinates and constructing scenes; applications for visualization (polygonization, VRML generation, ray-tracing), animation, voxelization and others; a collaborative Internet-based modeler including a HyperFun-to-Java translator and advanced interactive techniques based on the "empirical modeling" paradigm. These components are intended to be public domain to stimulate collaborative development efforts.

CR Categories and Subject Descriptors: [Computer Graphics]: Computational Geometry and Object Modeling – Curve, Surface, and Object Representations.

Additional Keywords: shape modeling, implicit surfaces, F-rep, multidimensional modeling, programming language, HyperFun, Java, empirical modeling.

1: Shape Modeling Lab
Department of Computer Software
University of Aizu, Aizu-Wakamatsu City,
Fukushima Prefecture 965-8580 Japan
{pasko, savchen, s1042083}@u-aizu.ac.jp

2: valery@acm.org, tolos@glasnet.ru

3: richard.cartwright@rd.bbc.co.uk

Published in
Eurographics/ACM SIGGRAPH
Workshop Implicit Surfaces '99, Bordeaux, France

1 INTRODUCTION

Most of the currently known implicit modeling systems are oriented towards specific subsets of objects and operations such as traditional skeletal models [12, 28], convolution surfaces [7, 21], distance-based models [13], or Constructive Solid Geometry (CSG) [8]. Although all these models are of the same mathematical nature, it is still not possible to exchange models between the systems, and therefore between the users. In this paper, we present a project devoted to developing an open system based on the more general function representation (F-rep) [17].

In F-rep, a complex object is defined by a single continuous function of several variables (point coordinates in multidimensional space). The representation can be constructed by applying different operations to primitive objects. A primitive is considered as a "black box" with the defining function given by the function evaluation procedure. There is a rich set of operations closed on the representation, i.e., resulting in a continuous real function [17, 20, 22]: set-theoretic operations and Cartesian product defined with R-functions, blending, offsetting, sweeping, projection, hypertexturing, metamorphosis, and extended space mapping, which combines space and functional transformations (mappings). Therefore, F-rep is a more generalized model with respect to traditional skeletal implicits, convolution surfaces, distance-based models, CSG, sweeps, and voxel models, and can therefore unify them.

The proposed system architecture is built around the shape models in HyperFun, which is a high-level programming language for specifying F-rep models. The motivation for this project stems from considering the following issues:

- Exchange protocol. There are several well-known protocols for exchanging geometric data such as polygonal file formats (Autodesk DXF, etc.), Alias/Wavefront object files for parametric surfaces, PADL-2 for CSG [9], and STEP for B-rep and CSG [15]. The VRML extension proposal [27] was the first attempt to introduce a protocol for skeletal implicit surfaces with a limited set of operations (warping, Boolean). A more general protocol for exchanging F-rep models is needed.
- Multidimensionality. F-rep naturally supports multidimensional modeling using functions of several variables $F(x1, x2, \ldots, xn) \geq 0$, where xi are point coordinates in n-dimensional Euclidean space. Practical multidimensional modeling should be supported by the language. Further interpretation of multidimensional models in terms of multimedia and animation should be considered.
- Building applications. F-rep models should be easily available to and processed in application software. This can be provided by a plug-in type language interpreter intended for parsing and function evaluation. In this way, different applications can incorporate F-rep models

developed independently and received, for example, through an exchange protocol.

- Extendibility and openness. Extendibility is the core feature of an F-rep modeling system. New primitives, operations, and relations can be introduced by users in different categories (end user, application software developer, kernel modeling system developer). The system should be extendable on the levels of symbolic (textual) and graphical user interface. The system's openness means the standardized ways of including new interface, modeling, and application components. This also supposes open source and collaborative development.
- Component libraries. Creation of application-specific libraries of reusable F-rep components on different levels (HyperFun, C, Java) provides the modeling system with adaptability to different application domains, and customizability to meet the needs of particular users.
- Multimodal interface. Different types of user interfaces should be supported including symbolic, graphical, and haptic. Eventually, the interfaces serve to create HyperFun models.
- Platform-independence. All of the system components may run on different computer platforms and to communicate through the HyperFun based protocol.
- Internet-based modeling. Collaborative distributed modeling can be supported with HyperFun files stored on an Internet server and exchanged between clients in the same manner as other Internet resources. Here, the HyperFun language can serve as the basis for a lightweight transmission protocol for the collaborative exchange of geometric models.
- Advanced interactivity. The user would like to experiment with models by specifying shape re-definitions, and observing their behavior on the fly. In this case, functional dependencies between objects have to be maintained by the system with a mechanism similar to an electronic spreadsheet.
- Education. A system providing means for the easy specification and operation with implicit surfaces would be very useful in teaching a number of courses such as analytical geometry, computer graphics, animation, or visualization.

In this paper, we present the state-of-the-art HyperFun project contributed to by the University of Aizu (Japan), the University of Warwick (UK), and the Moscow Engineering Physics Institute (Russia). The paper is structured as follows. Section 2 outlines the general scheme of the proposed system. The HyperFun language and its interpreter are described in Section 3. Sections 4-6 present modelers of different types. Applications are briefly characterized in Section 7. Conclusions and future directions described in Section 8.

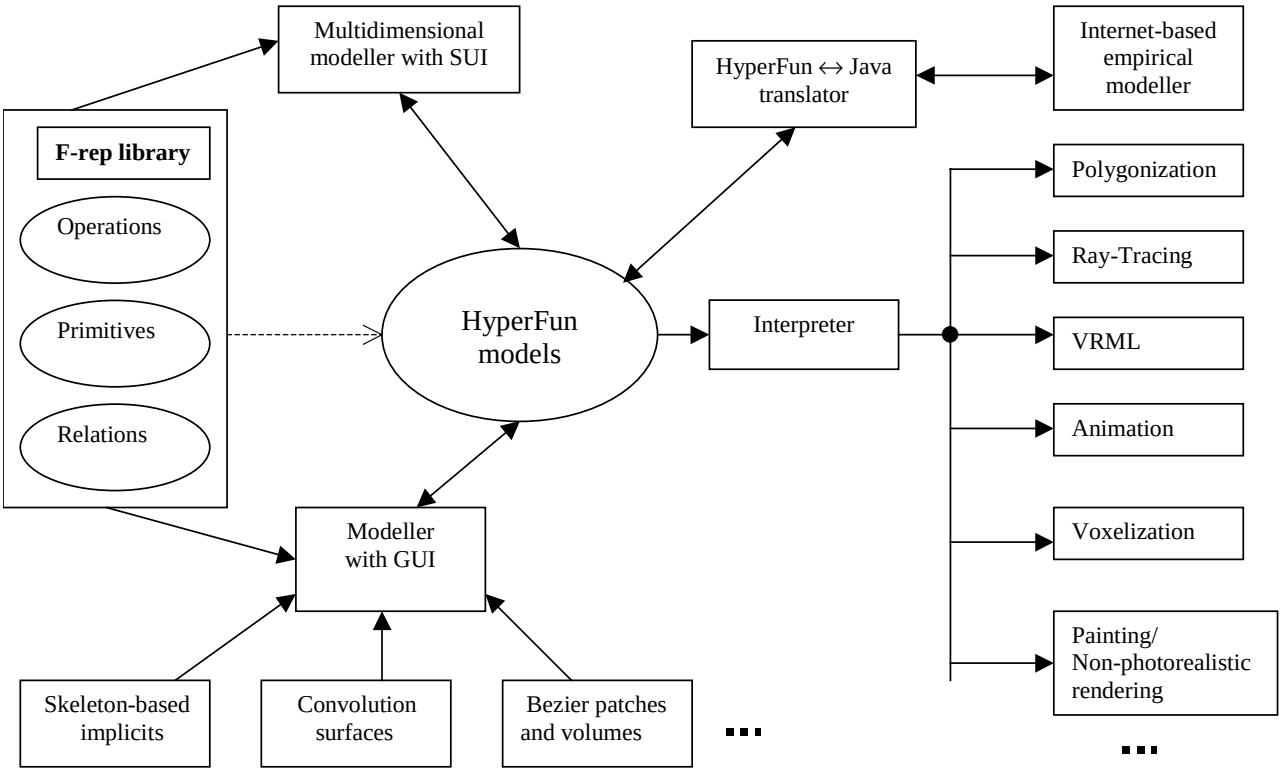
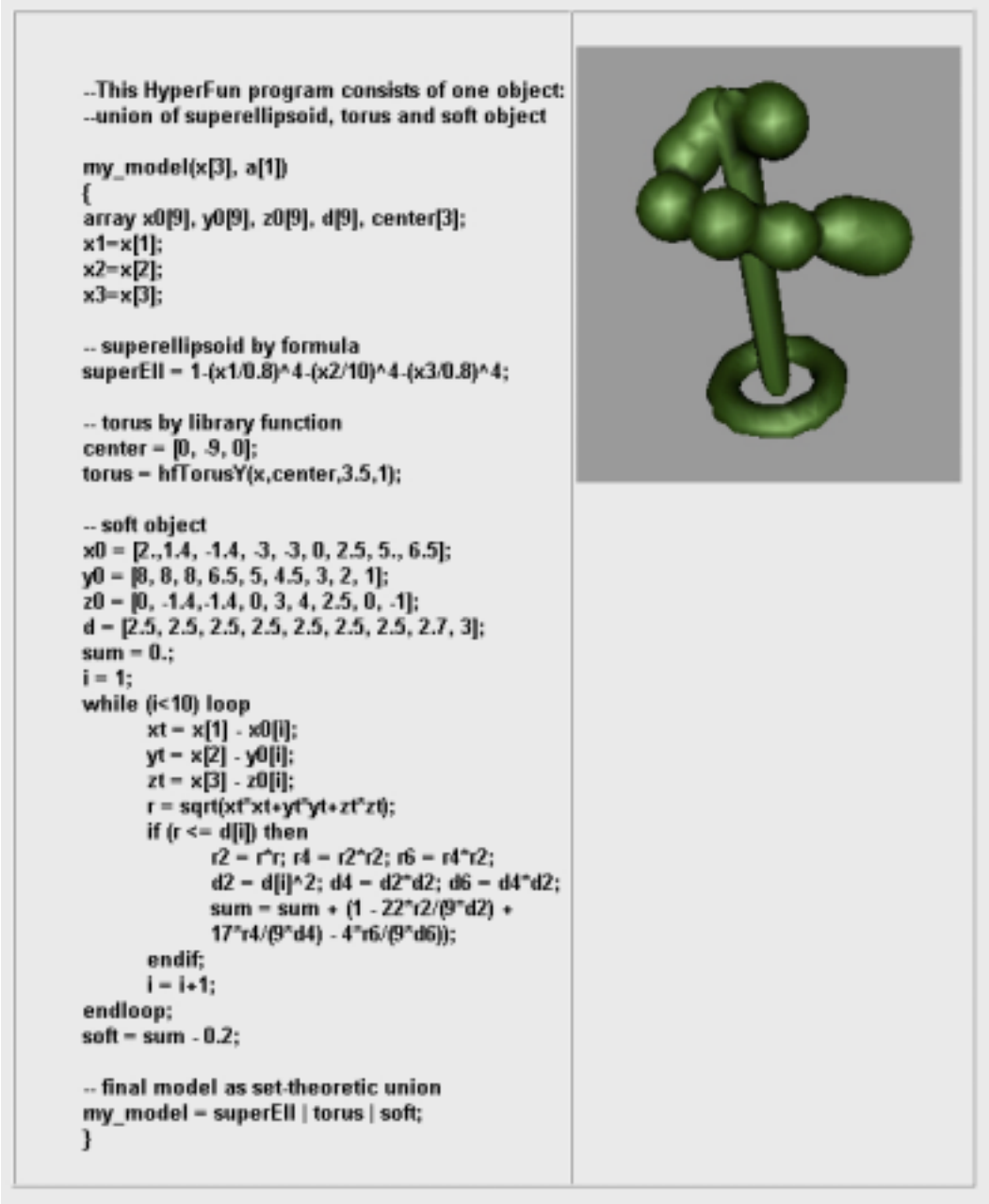


Figure 1: System architecture

2 SYSTEM ARCHITECTURE

The proposed system architecture is shown in Fig. 1. It consists of the following basic groups of components: HyperFun models, F-rep library, modelers, interpreters/compilers, and applications. HyperFun models are text files that can be stored and transmitted over a network. The F-rep library contains definitions of primitives, operations, and relations used in HyperFun models. This library is currently implemented in C, Java, and HyperFun. The selection of the library depends on the application area and computing environment. While system libraries are written in C and Java, libraries in HyperFun are created by end users and are usually oriented to a specific application area. The modelers are interactive tools for building F-rep models and exporting them as HyperFun files. The HyperFun interpreter provides program parsing and function evaluation at the given point, and can be plugged into applications to process HyperFun models. The special Internet-based modeler accesses HyperFun models through the HyperFun-to-Java translator and supports collaborative model development using definitive scripts. The above mentioned components are described in the following sections.



```
--This HyperFun program consists of one object:
--union of superellipsoid, torus and soft object

my_model(x[3], a[1])
{
array x0[9], y0[9], z0[9], d[9], center[3];
x1=x[1];
x2=x[2];
x3=x[3];

-- superellipsoid by formula
superEll = 1-(x1/0.8)^4-(x2/10)^4-(x3/0.8)^4;

-- torus by library function
center = [0, -9, 0];
torus = hfTorusY(x,center,3.5,1);

-- soft object
x0 = [2.,1.4, -1.4, -3, -3, 0, 2.5, 5., 6.5];
y0 = [8, 8, 8, 6.5, 5, 4.5, 3, 2, 1];
z0 = [0, -1.4,-1.4, 0, 3, 4, 2.5, 0, -1];
d = [2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.7, 3];
sum = 0.;
i = 1;
while (i<10) loop
        xt = x[1] - x0[i];
        yt = x[2] - y0[i];
        zt = x[3] - z0[i];
        r = sqrt(xt*xt+yt*yt+zt*zt);
        if (r <= d[i]) then
                r2 = r*r; r4 = r2*r2; r6 = r4*r2;
                d2 = d[i]^2; d4 = d2*d2; d6 = d4*d2;
                sum = sum + (1 - 22*r2/(9*d2) +
                17*r4/(9*d4) - 4*r6/(9*d6));
        endif;
        i = i+1;
endloop;
soft = sum - 0.2;

-- final model as set-theoretic union
my_model = superEll | torus | soft;
}
```

Figure 2: Sample HyperFun model and its polygonized surface

3 HYPERFUN LANGUAGE AND INTERPRETER

3.1 Language Overview

HyperFun is a modeling language designed to be a high-level tool suitable for specifying functionally based models. The language was intentionally kept as simple as possible to provide for ease of mastery. While being minimalist, it should not limit the user in creating quite complex geometric models. It supports all main notions in F-rep, particularly "geometric objects" and "geometric operations".

A model in HyperFun can contain the specification of several geometric objects. Each object is defined by a function parametrized by input arrays of point coordinates and free numerical parameters. The number of coordinate variables can be greater than three to allow definition of higher dimensional objects. 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'). Functional expressions are built 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).

The languages principle feature is the ability to use the system F-rep library that contains functions representing geometric primitives and transformations. The library is extendible, and its composition can be changed depending on a particular domain. The library version in general use contains the most common primitives ('Sphere', 'Torus', 'Ellipsoid', 'Cylinder', 'Blobby object', 'Metaball object', etc.) and transformations ('Blending union/intersection', 'Rotation', 'Scaling', 'Twisting', etc). Functional expressions can also include references to previously defined geometric objects. The user can create his/her own library of objects for later reuse.

A sample of a HyperFun model can be found in Fig. 2. The on-line manual for the HyperFun language, containing a description of the current version of F-rep library as well as numerous examples of programs in HyperFun, can be found at the devoted Web-site [14].

3.2 Interpreter of the Language

The HyperFun interpreter is implemented as a suite of functions in ANSI C. The interpreter provides parsing with syntax and semantic analysis for a program in HyperFun. We will now describe the two most important functions that correspondingly build an internal representation and perform the function evaluation at the given point for the given parameters.

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. Another possible result is a list of messages about any errors together with their locations and specifics. As to the form of the object's internal representation, it is supposed to be optimal for the subsequent function evaluation (we need to evaluate the function at given points in the modeling space, and the number of them can be very large). Accordingly, the internal representation can be considered as a tree structure ready to effectively evaluate all expressions defining the complete function.

The 'Calc' function performs the function evaluation for the given object at the given point and for the given external parameters. The object's internal representation serves as an input parameter for the function that returns the value of the evaluated function at the given point; if a certain error appears, the function generates the corresponding message about the error and its location.

The formal specification of the internal representation and the function evaluation procedure was given in [19]. On the whole, the interpreter provides an effective procedure for building an internal representation and for function evaluation. It should be emphasized that the 'Parse' module is invoked just once while processing the Hyperfun model; the internal representation it creates can be treated as Java-like "byte-code" (note that it is platform-independent) and can in principle serve as a protocol for data exchange between system components. In fact, these two procedures form an application programming interface (API) in contrast to traditional geometric APIs consisting of large number of modules.

4 MULTIDIMENSIONAL MODELER WITH SYMBOLIC USER INTERFACE

Obviously, modern geometric modeling systems should provide a GUI enabling the user to specify the model using direct manipulation and other conventional interactive techniques. However, we see the principal drawbacks of a GUI, in particular its limited design vocabulary and problems it has dealing with multidimensional models. It is very useful to have a modeler with a "symbolic" user interface (SUI) based on a high-level geometric language (HyperFun in our case). The following benefits of a SUI can be mentioned:

- more rich functionality provided by direct textual input of equations for primitives and operations of arbitrary complexity;
- dealing with multidimensional models with further interpretation of them;
- support for advanced users, who are especially inclined to use the SUI;
- using a SUI in education lets students understand and master the underlying laws of geometric modeling;
- possibility for generating symbolic definitions using genetic programming techniques.

We have implemented a multidimensional modeler with SUI that allows the user to:

- Specify a functionally-based model in HyperFun using the built-in editor and interpreter;
- Use the standard 'F-rep library' of geometric objects and transformations as well as the user's own library of geometric objects written in HyperFun;
- Define mappings of geometric space to multimedia space by assigning *multimedia types* to geometric coordinates [1];
- Compose scenes consisting of a few objects, each defined in its own modeling space;Generate images of polygonized or ray-traced elementary shapes, animation sequences, 1D and 2D spreadsheets in accordance with assigned multimedia types.

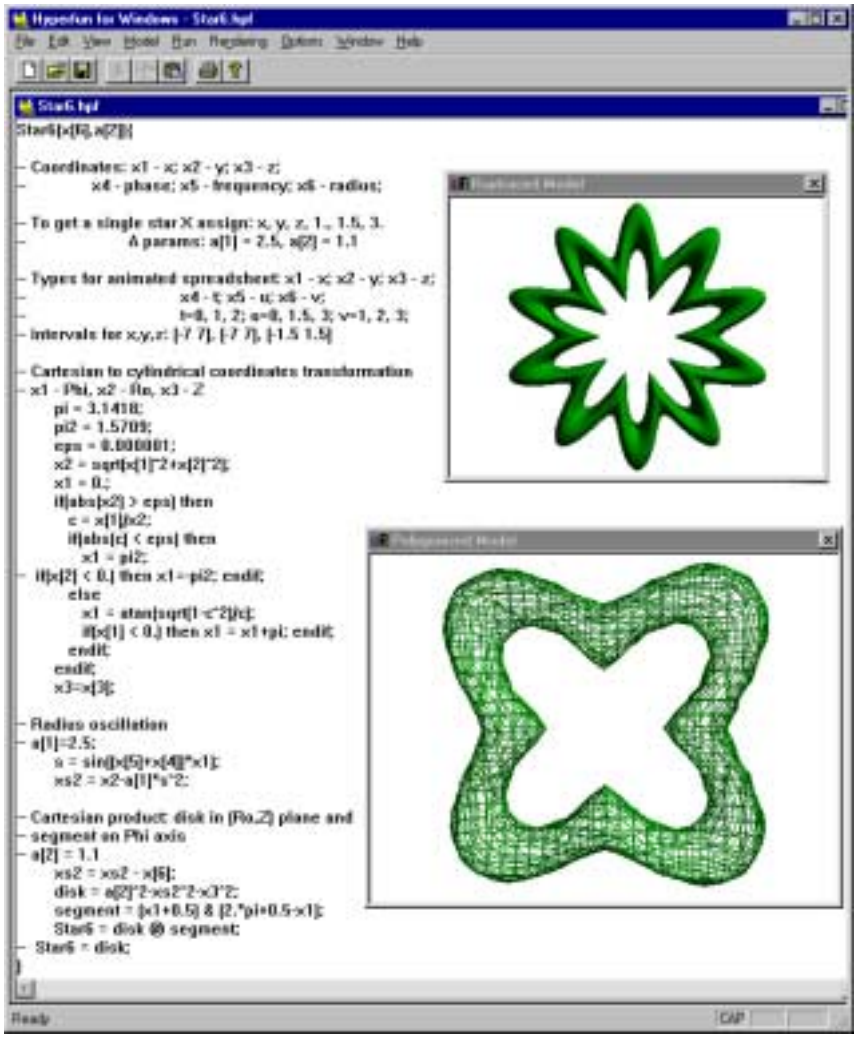


Figure 3: Screenshot of the multidimensional modeler with a symbolic user interface.

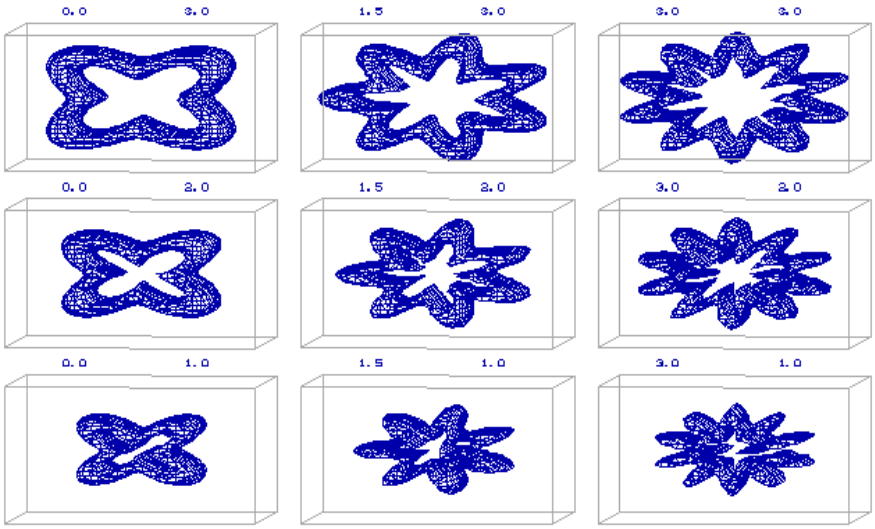


Figure 4: Frame of an animated spreadsheet

Fig. 3 demonstrates a screenshot of the system in the process of modeling. The current version of the modeler called "HyperFun for Windows" is implemented on the PC platform using Windows NT.

We will now describe in more detail how the user can deal with multidimensional models. The concept of *multimedia types* is exploited here. It is possible to define geometric objects in HyperFun using the coordinate variables x[1], x[2], …, x[n]. There is a special dialogue window that lets the user associate each geometric coordinate variable with a certain multimedia type. These types establish the conventions governing coordinate variables' semantics by giving a concrete interpretation of x[i]. In the current prototype system implementation, the following multimedia types can be assigned to x[i]:

- "x", "y" and "z" types correspond to world coordinates of "real life" geometry. These can be Cartesian, cylindrical, or other coordinates. The selection of these types defines the "elementary" image or 2D/3D shape within the bounding box in the selected geometric space. An "elementary" shape (i.e., curve, surface, isosurface) is the projection of a cross-section of the initial multidimensional shape.
- "t" corresponds to dynamic coordinates representing continuous values that can be linearly or non-linearly mapped onto physical time. The interval of such a coordinate can mean the life time of the multimedia object. This type can be assigned to one or several geometric coordinate variables. The user can actually give a path in the space of variables with "t" type by filling out a table with their discrete values. Each row of the table corresponds to certain time value. Then, the frames of the animation sequence in the form of the model's cross-sections can be produced. This is the basis for implementing such operations as metamorphosis.
- "u" and "v" types correspond to 2D spreadsheet coordinates that take discrete values in the given bounding box. This type allows for spreadsheet-like spatial organization of elementary images or shapes in the regularly placed cells.
- "c" corresponds to a photometric coordinate, namely the color. Color is interpreted differently depending on the selected type of the elementary image or shape. For example, if the elementary shape is a 3D surface, color can be assigned to a light source in the scene. In fact, color is a vector of three real values (R,G,B) defining red, green, and blue components of RGB color model. Therefore, the mapping should be provided from a geometric coordinate onto all (R,G,B) components.

To define a cross-section of the multidimensional model the user can assign a constant numerical value to the corresponding x[i]. By assigning one or several constant values to the function x[n+1], the user can select a contour map or a set of isosurfaces as an elementary shape.

Let us give an example of assigning multimedia types for the model shown in Fig. 3. The object is defined by a function of six variables. The following types are assigned to the geometric coordinates:

```
x[1] → x
x[2] → y
x[3] → z
x[4] → t
x[5] → u
x[6] → v
```

Assigning a zero value to the function itself defines an isosurface as an elementary shape in the cells of the spreadsheet. The elementary shape illustrates function dependence on three variables x[1], x[2], and x[3]. Changes of isosurfaces along rows and columns of the spreadsheet illustrate function dependence on x[5] and x[6]. Changes of the entire spreadsheet in time show how the function depends on x[4]. The image in Fig. 4 is a frame of the animation corresponding to t = 2. It can be thought as an image of a 5D shape since x[4] has constant value.

Assigning multimedia types to geometric coordinates and the subsequent generation of images and animations allows for:

- interpretation of the multidimensional shape involving more human senses;
- uniform treatment of all kinds of multimedia coordinates (Cartesian coordinates of "real life" 2D and 3D geometry, time, color, sound, etc.).
- new visual representations of multidimensional shapes such as animated spreadsheets of images or 3D objects;
- non-traditional behavior descriptions for animation;
- the emergence of new multimedia applications such as synthesis of music, color, and dynamic 3D shapes based on strict mathematical definitions.

5 MODELER WITH AN EXTENDABLE GRAPHICAL USER INTERFACE

We present a HyperFun modeler with a high-level graphical user interface that will allow users to model 3D shapes using HyperFun primitives and operations. The user creates models using simplified polygonal meshes that represent the underlying F-rep primitives. 3D widgets and other visual cues are displayed to allow the user to interact with and understand HyperFun operations used during the modeling process. At any time during the modeling process, the user can polygonize the current model state to get a more accurate estimation of the actual underlying shape. The fundamental output of the modeler is a text file of the model in HyperFun. This text can then be read by the various applications available for the HyperFun language.

A model consists of an ordered set of objects. Each object is represented as a separate n-ary tree that is built out of nodes (see Fig. 5). Each node is of a certain type, and this type information is held in the node library. When a node is put on a tree, the type information initializes another structure that holds the values of the parameters of the node. Leaves on the tree correspond to F-rep library primitives, previously created objects in the model, inline HyperFun primitives, or user defined HyperFun primitives. Other nodes in the tree represent F-rep library operations, inline HyperFun operations, or user defined HyperFun operations. Inline primitives and operations allow the user to specify on the fly a

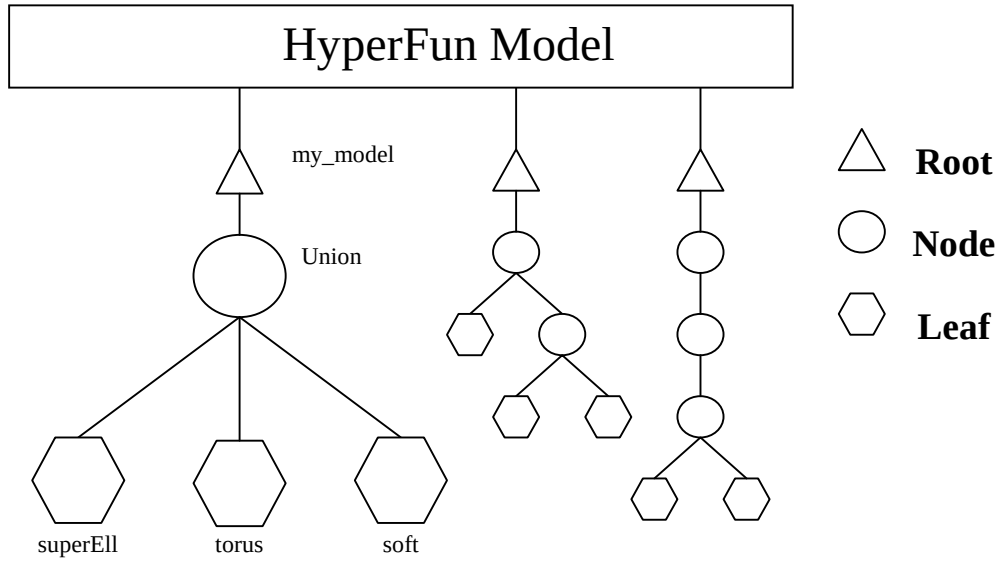


Figure 5: HyperFun model tree structure. The tree on the left is a representation of the model shown in Figure 2 with the soft object represented as a single primitive.

textual definition for the node on the tree. The fundamental difference between a primitive and an operation is that operations must be the roots of sub-trees in the object, and primitives can have no such sub-tree. A tree that has an operation without a sub-tree is incomplete, and therefore meaningless until complete. Nodes can be divided into two types: space mappings and functional mappings (see [22]).

Modeling is accomplished by adding, deleting, or moving nodes on the tree, and by changing the parameters associated with each node. These parameters are mapped onto their graphical and semantical meaning in the modeler by the node definitions contained in the node library. Due to this mapping, the user is given the ability to interactively change these parameters either graphically or symbolically. Symbolically, the user can directly type in the values for the parameter, or pick an object argument that will be assigned to it. Graphically, the user can use the mouse to manipulate a widget that has been associated with the parameter. Changing parameter values in either way results in changes to the onscreen display. The graphical subsystem that creates the display is based on OpenGL. A multi-platform software toolkit (MAM/VRS) is being used for the OpenGL interface. This toolkit has a Tcl/Tk binding, and so we use Tcl/Tk for creating the GUI. The use of these toolkits in combination allows the software to be compiled on various machines. The same source code is compiled on a Windows based workstation, or a Silicon Graphics workstation. The HyperFun textual output is created simply by sequentially traversing each tree (object) in the forest (model). Text is added to the file stream during both downward and upward traversal of the tree. The output during downward traversal consists of HyperFun text related to space mapping nodes. On the other hand, functional mapping information is output during upward traversal.

One of the essential features of this modeler is its extendibility. The core code of the modeler does not contain definitions of the nodes used in the modeler. These node definitions exist in external data files that specify the type and function of the node, as well as semantic and graphical information that gives the node meaning in the modeler. Upon initialization, the program reads these data files and fills the node library with the data contained therein. In this way, the user is given the ability to extend the modeler's node library to include any new primitive or operation that they may define in the HyperFun language. In the same way, this library may be extended to include future additions to the system F-rep library itself. For more complicated additions to the F-rep library, a different type of interface may be needed. In this case, the modeler may be extended with a new interface that would be used when working with the new function. One example of such an extension is the included interface that allows modeling using Bezier patches and volumes [23].

6 INTERNET-BASED EMPIRICAL MODELER

6.1 HyperFun to Java translator

The HyperFun to Java translator has two main purposes. The first is to provide an alternative method for fast and effective realization of HyperFun shapes, from those already discussed, that is platform independent and appropriate for easy distribution via the Internet (see Section 6.2). The second is for building algebras of shape that can be used in spreadsheet-like applications for geometric modeling and shared between many agents in collaborative and concurrent interaction with, and incremental development of, shape models (see Section 6.3). The second approach is based on Empirical Modeling principles [4, 5] that allow for the description of realistic behavior of shape models that are situated in multi-agent environments, providing animation through open-ended exploration and experimentation with models.

The translator started life as an undergraduate assignment for a course teaching parsing techniques at the University of Warwick. The students were asked to write a lexical analyzer and parser generator [2] for the HyperFun notation. The final part of the assignment asked the students to add semantic actions to their parser that write out Java code corresponding to a HyperFun model presented as input. To support the students work, they

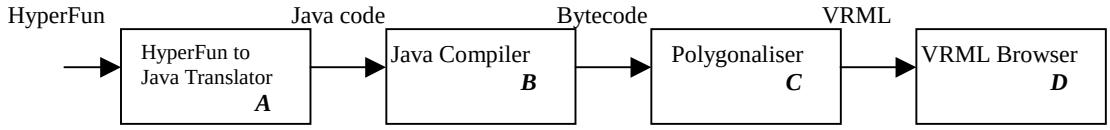


Figure 6. VRML generation using HyperFun to Java translator

were provided with the standard HyperFun library of shapes hand translated into Java and a simple polygonalisation algorithm (similar to Bloomenthal's tetrahedral decomposition [6]) for the realization of HyperFun models as VRML files [26]. Successful students and the author of the model solution could then take a HyperFun model (see Fig. 6), translate it into new Java code (***A***), compile this Java code using the standard Java compiler (***B***), and execute this code to generate a VRML realisation of the HyperFun model (***C***). The shape can be viewed and explored in a VRML browser (***D***) such as CosmoPlayer from Cosmo Software (see `http://cosmosoftware.com/`).

The advantage of this technique is that the HyperFun code, which may be executed thousands of times over to produce one realization of a shape, is compiled into a bytecode format by an optimizing compiler. When executed, the Java just-in-time compiler converts the bytecode into native machine code that can be executed efficiently. Informal benchmark testing shows that Java can achieve similar and sometimes improved performance for repeated mathematical calculations in comparison to equivalent code in the C programming language.

The current implementation status of the translator is as described above and requires the user to type a sequence of commands at a shell prompt. Initial tests on a modest Pentium PC shows that the generation of a VRML model using a cell decomposition of 30 by 30 by 30 from a HyperFun file of 50 lines could be generated in under 10 seconds. The obvious next step in implementation is to integrate the translation, compilation and realisation phases into one combined tool with the replacement of VRML by the recently released Java 3D library [24].

6.2 Internet-based realization

One unique feature of Java is the ability to dynamically load classes at runtime. This means that is possible to build an application that generates Java bytecode and executes that code without the need to stop the execution to re-link the applications code. In more conventional approaches, it would be necessary to implement a runtime interpreter and this cannot benefit from the performance advantage provided by the just-in-time compiler and native code. So by utilising this dynamic feature of Java, it is possible to build an open-ended application that allows HyperFun shape files to be read, edited, compiled, and visualized. Such an application should include a choice of rendering algorithms that offer the user a choice between speed, quality of rendering, and style of interaction with models, with comparable speed to a conventional closed (at compilation) application.

Moreover, it is possible to use an Internet browser that supports Java (such as Netscape Navigator) to provide the same functionality as the application mentioned above. In addition, HyperFun files and their bytecode representation can be stored on an Internet web server and exchanged between any number of clients in the same manner as other Internet-based resources. The clients can communicate through a server that acts as a broker for HyperFun files and related information, providing services such as version control, password protection and persistent storage for models. In this way, the HyperFun notation is being used to provide a lightweight transmission protocol for the collaborative exchange of geometric shape.

Figure 7. Screenshot of Empirical Worlds in use.

6.3 Algebra and empirical modelling

The Empirical Worlds proof-of-concept application [11] demonstrates how an algebra of shape, including a large family of primitive shape types and operators on those primitives defined using F-rep techniques, can be used as the underlying algebra of a definitive notation [3]. A screenshot of Empirical Worlds in use is given in Fig. 7. In Empirical Worlds, it is possible to create an instance of a shape type and assign it a name. Defining parameters used to create a specific instance of a shape data type are given to the application using syntax similar to that of VRML. As an example, the following two lines of scripts (each assignment is known as a *definition*) create an instance of a solid box and a solid cylinder shape:

```
body = Box { size 2 2 2 }
hole = Cylinder { height 3 radius 0.3 }
```

Using the operator *cut*, it is possible to create a new piece of geometry that is the set theoretical subtraction of the cylinder (hole) from the box (body).

```
cutBody = cut(body, hole)
```

This definition establishes a dependency between the body, hole and cutBody. Whatever values are subsequently defined for body or hole, cutBox is automatically maintained consistent with its definition. For example, if the body is redefined to be a sphere of radius 2, then cutBody is updated to become the set-theoretical subtraction of a cylinder from a sphere. An Empirical Worlds script is similar to a spreadsheet for geometric shape, where identity via grid location has been substituted for by identity using assignment operators and identifiers in a programming notation.

In the current version of Empirical Worlds, the algebra is fixed and a specialist programmer is required to create new primitives. The implementation is currently in Java and the HyperFun to Java translator could allow for new shape data types and operators to be created or edited on-the-fly. These can then be used in Empirical Worlds scripts with the new shape data types having fast and effective realisation through the use of compiled native code. To achieve this, it will be necessary to have a meta-language for describing the mechanism for dynamically linking a new HyperFun generic shape data type into the empirical world notation and/or graphical user interface.

A fully integrated tool supporting all functionality described above could be used to access large libraries of shapes defined using HyperFun in a textual or binary format. Each definition can be attributed to an agent owner who can choose privileges for other agents to reference or redefine it. Definitions can be used as a very lightweight way of transmitting shape models and aspects of behavior in the models between several different agents collaborating over the Internet. This collaboration can be concurrent, as is illustrated by the dtkeden tool that supports two-dimensional line drawing [10]. Moreover, definitive scripts are particularly suited to incremental model development or modification in distributed environments as only the change to the model has to be transmitted to synchronize all agents' views. A dependency maintainer on each client maintains models consistent and up-to-date with their definitions.

Definitions can also be used to represent dependencies between the location of shape as observed in the models referent. For a humanoid shape, it is possible to define the realistic behaviour of a shoulder joint by introducing angular parameters for each degree of freedom for possible movements of the top half of the arm, with the range of parameters limited to prevent the arm embedding itself into the chest. Simple animations can be created by the introduction of a parameter *t*, on which the location and size of shapes can depend. In this way, it is possible to establish multi-agent environments for the exploration of shape models, and their associated behaviours, in a manner that is not preconceived and benefits from effective, distributed realisation.

7 APPLICATION EXAMPLE

Application software provides visual representation (ray-tracing, non-photorealistic rendering), animation, conversion to other representations (polygonization, VRML generation, voxelization), analysis, storage, rapid prototyping, and other types of processing of HyperFun models. An application reads the model and invokes the interpreter to parse it and to evaluate the function at a set of points. Although the detailed description of applications is beyond the scope of this paper, let us discuss one example, that of using ray-tracing to visualize a multidimensional model.

In Fig. 9 we present a ray-traced image illustrating a *bi-directional metamorphosis*, which is a non-traditional operation in computer graphics and animation. It results in a smooth transformation (metamorphosis) between four key 3D shapes modeled using F-rep. Algebraically, the model of the bi-directional metamorphosis is the bilinear interpolation between four real-valued functions by coordinates x_4 and x_5. Geometrically, it is a 5D object defined by the real function as $\phi(x_1, x_2, x_3, x_4, x_5) \geq 0$. The selected key-shapes are, to some extent, "cultural key signs" in Japan:

- "Cat" (upper left) resembles the cult character of children's animation. Its complete model in HyperFun can be found at [14];
- "NiHon" (lower left) is a 3D puzzle representing the word "Japan". First, two 3D Chinese characters "*Ni*" and "*Hon*" are constructed independently as unions of blocks. Then, the solids are oriented along Z and X axes respectively and combined as $NiHon = Ni \cap Hon$, where $\cap$ represents intersection operation. The idea of this puzzle construction is that the resulting 3D solid looks like the single initial 2D character "*Ni*" or "*Hon*" when projected along the Z and X axes respectively onto a plane.
- "Robot" (upper right) and "Rob_let" (lower right) are also modeled using set-theoretic operations with R-functions.

```
Meta5D(x[5],a[1])
{
array xx[3];
xx[1] = x[1]; xx[2] = x[2]; xx[3] =
x[3];

-- (0,0): Cat
  Cat = Cat3D(xx);

-- (0,1): NiHon
  NiHon = NiHon3D(xx);

-- (1,0): Robot
  Robot = Robot3D(xx);

-- (1,1): Rob_let
  Rob_let = Rob_let3D(xx);

-- Bi-directional metamorphosis
Meta5D =
(Cat*(1.-x[4])+Robot*x[4])*(1.-x[5])
+ (NiHon*(1.-x[4])+Rob_let*x[4])*x[5];
}
```

Figure 8: HyperFun model of the bi-directional metamorphosis

The HyperFun model of the bi-directional metamorphosis is shown in Fig. 8. A 2D spreadsheet presented in Fig. 9 is the most adequate visual representation of this 5D object. The following assignment of the multimedia types defines such spreadsheet:

```
x[1] → x
x[2] → y
x[3] → z
x[4] → u,p1
x[5] → v,p2
```

Coordinates x_1, x_2, x_3 (types **x**, **y**, and **z**) continuously change inside the 3D bounding box. Coordinates x_4 and x_5 (types **u** and **v**) take five discrete values along horizontal and vertical axes respectively. Each cell of the spreadsheet contains an elementary shape corresponding to a specific point in the (x_4, x_5) plane. Each of the key shapes is assigned a 3D texture. Four textures are also interpolated in the (x_4, x_5) plane (photometric types **p1** and **p2**). Note that smooth shape and texture transformations can be observed in any linear direction in the image. On the other hand, the spreadsheet can be considered as a set of frames of animation with two dynamic variables. It can serve as a reference for constructing an animation sequence containing particularly interesting shape transformations [16].

The image was rendered using POVRay 3.0 [18] (a freeware raytracer available on various platforms) running on Intel Pentium II Windows NT workstations. A modified version of POVRay with "the POV-Ray isosurface patch" [25] was used to render the isosurfaces.

Figure 9. Spreadsheet of bi-directional metamorphosis between four shapes

8 CONCLUSIONS

We presented a project with a high-level modeling language as its unifying core. An open system architecture was proposed and a description of its components was given. The language is powerful enough to completely support multidimensional F-rep modeling including both traditional implicit models (blobby and soft objects, metaballs and convolution surfaces) and advanced models (sweeps, CSG, non-linear transformations, etc.). On the other hand, our experience shows that undergraduate students can easily master the language and the supporting tools which are used in different courses in our universities.

The language-based protocol can serve well for exchanging models between users, modeling systems and networked computers. The project has also shown its potential to support advanced interactive "empirical modeling" techniques for collaborative work on Internet. The project itself is open for collaboration between research groups. As the first step in this direction, we plan to make the system components public domain.

References

[1] Adzhiev V., Osipov A., Pasko A. "Multidimensional shape modeling in multimedia applications" (submitted to Multimedia Modeling '99).

[2] A. W. Appel. Modern Compiler Implementation in Java. Cambridge University Press, 1998.

[3] W. M. Beynon. Definitive notations for interaction. In Proceedings of HCI 1985, pages 23-34, Cambridge University Press, 1985.

[4] W. M. Beynon Empirical modelling for educational technology. In Proceedings of Cognitive Technology 1997, pages 54-68. University of Aizu, Japan, IEEE, 1998.

[5] W. M. Beynon. Empirical modelling and the foundations of artificial intelligence. In Proceedings of International Workshop on Computation for Metaphors, Analogy and

Agents. Lecture Notes in Artificial Intelligence 1562, pages 322-364, Springer, 1999.

[6] J. Bloomenthal et al. Introduction to Implicit Surfaces. Morgan-Kaufman, 1997.

[7] Bloomenthal J, Lim C. "Skeletal methods of shape manipulation", Shape Modeling International '99 (University of Aizu, Japan), IEEE Computer Society, 1999, pp. 44-47.

[8] Bowyer A. "SvLis -- Introduction and User Manual", Information Geometers Ltd and University of Bath, 1999.

[9] Brown C. "PADL-2: a technical summary", IEEE Computer Graphics and Applications, vol. 2, No.2, 1982, pp. 69-84.

[10] W. M. Beynon and P. H. Sun. Computer-mediated communication: a distributed empirical modelling perspective. In Proceedings of Cognitive Technology 1999, San Francisco, August 1999, to appear.

[11] R. I. Cartwright. Geometric Aspects of Empirical Modelling: Issues in Design and Implementation. PhD thesis, Department of Computer Science, University of Warwick, September 1998. Available at http://www.dcs.warwick.ac.uk/people/academic/Richard.Cartwright/thesis/phd.html

[12] Desbrun M., Gascuel J.-D., Cani-Gascuel M.-P., Tsingos N. "Fabule - a multi-purpose animation system for simulation and virtual reality", ERCIM News No.31, 1997.

[13] Hart J. "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces", The Visual Computer, vol. 12, No. 10, 1996, pp. 527-545.

[14] "HyperFun: Language for F-rep Geometric Modeling", http://www.u-aizu.ac.jp/labs/sw-sm/FrepWWW/HF_descr.html

[15] Owen J. "STEP: An Introduction", Information Geometers, Winchester, UK, 1997.

[16] Pasko A., Adzhiev V., Fausett E. "Multidimensional shape modeling for animation", to appear in EUROGRAPHICS'99, short presentations.

[17] 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, pp.429-446.

[18] "POV-Ray – the Persistence of Vision Raytracer", http://www.povray.org/

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

[20] Pasko A., Savchenko V. "Constructing functionally defined surfaces", Implicit Surfaces '95, Eurographics Workshop (Grenoble, France, 18-19 April 1995), B. Wyvill and M.-P. Gascuel (Eds.), pp. 97-106.

[21] Sherstyuk A. "Interactive shape design with convolution surfaces", Shape Modeling International '99 (University of Aizu, Japan), IEEE Computer Society, 1999, pp. 56-65.

[22] Savchenko V., Pasko A. "Transformation of functionally defined shapes by extended space mappings", The Visual Computer, vol. 14, No. 5/6, 1998, pp. 257-270.

[23] Schmitt B., Pasko A., Savchenko V. "Extended space mapping with Bezier patches and volumes", submitted to Implicit Surfaces'99.

[24] H. Sowizral, K. Rushforth and M. Deering. The Java 3D API specification. The Java Series, Addison-Wesley Longman, December 1997. See also http://java.sun.com/products/java-media/3d/

[25] Suzuki R. "POVRay 3.0 isosurface patch", http://www.public.usit.net/rsuzuki/e/povray/iso/

[26] International Organization for Standardisation and International Eltrotechnical Comission. The virtual reality modelling language, version 2.0. Draft specification ISO/IEC CD 147720, ISO/IEC, August 1996. Available at http://vrml.sgi.com/moving-worlds/spec/.

[27] Wyvill B., Guy A. "The Blob Tree. Implicit modelling and VRML", International conference From the Desktop to the Webtop: Virtual Environments on the Internet, WWW and Networks, NMPFT, Bradford, April 1997.

[28] Wyvill B., Guy A., Galin E. "Extending the CSG tree. Warping, blending and Boolean operations in an implicit surface modeling system", Implicit Surfaces '98, Eurographics/ACM SIGGRAPH Workshop, Bloomenthal J. and Saupe D. (Eds.), University of Washington, Seattle, USA, 1998, pp. 113-121.