Pre-final version before editing by the IEEE CG&A magazine editors,
appeared in vol. 25, No. 2, 2005, pp. 60-69.

Web-based Shape Modeling with HyperFun

Richard Cartwright[α], Valery Adzhiev[β], Alexander Pasko[χ,δ], Yuichiro Goto[δ], Tosiyasu L. Kunii[δ]

(α) BBC Research & Development, UK
(β) The National Centre for Computer Animation, Bournemouth University, UK
(χ) Hosei University, Japan; author for correspondence: pasko@k.hosei.ac.jp
(δ) IT Institute, Kanazawa Institute of Technology, Japan

We present a project on research and development of the high level language HyperFun for shape modeling using implicit surfaces and the more general function representation (FRep). An approach to collaborative Web-based shape modeling using HyperFun is described in detail. The presented EmpiricalHyperFun shape modeling system based on Empirical Modeling principles provides users with an unusual degree of mutual interaction through the Web.

Key Words: Shape Modeling, Web-based Modeling, Implicit Surfaces, Function Representation, Empirical Modeling, HyperFun, Java, Applet

INTRODUCTION

Growing attention is paid recently to shape modeling and rendering using so-called "implicit surfaces" [3] and a more general function representation (FRep) (see the side bar) because of their well-known properties such as compact mathematical description of complex shapes, natural blending, covering both free-form organic shapes and mechanical parts with sharp edges, and a rich system of developed operations.

The increasing number of applications facilitates the importance of methods and software tools for Web-based shape modeling. These applications include Web authoring, multi-user gaming, distant learning, data visualization, 3D retail environments in e-commerce, collaborative computer-aided design and engineering, and others. Many of these application areas have their own proprietary data representations of shape, making sharing of these shapes for collaborative engagement with a team of users a task for dedicated software with restricted distribution and applicability.

There are several well-known protocols for exchanging geometric data based mainly on polygonal and parametric surfaces models, which are quite verbose. A more high-level and concise format for geometric data transmission would enable truly collaborative and creative shape modeling on the Internet through browsers. This can be achieved by employing shape models of a higher level of abstraction than is provided by polygonal meshes. Implicit surfaces and FRep are among the best candidates for solving this problem.

This paper presents a project devoted to developing an open system architecture and Web-based shape modeling applications using HyperFun, which is a high-level programming language for specifying implicit surfaces and FRep objects. We first give motivation for and the general characteristics of the project, and then discuss in detail its Web-based modeling aspects such as a HyperFun to Java Bytecode runtime compiler, a simple HyperFun applet, and a more general collaborative modeling system called EmpiricalHyperFun (EHF). This system based on Empirical Modeling principles (see the side bar) allows for implementation of realistic behavior of shape models that are situated in multi-agent environments, providing open-ended exploration and experimentation with shared models.

RELATED WORKS

Most of the known implicit modeling systems are oriented towards specific subsets of objects and operations such as traditional skeletal models [15], convolution surfaces [11], distance-based models [6], or Constructive Solid Geometry (CSG) [4]. 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 (FRep) - see the side bar.

There are several well-known protocols for exchanging geometric data such as polygonal file formats (e.g., Autodesk DXF), Alias/Wavefront object files for parametric surfaces, PADL-2 for Constructive Solid Geometry (CSG), and STEP for the boundary representation (B-rep) and CSG. The Virtual Reality Modeling Language (VRML) [13] is an open standard that can be used as a means to distribute a shape to special Internet browsers that can be explored by a distributed set of users. VRML besides the description of the shapes contains additional information about the graphical scene such as light sources, background, texture, and other data. The recently proposed X3D standard [16] allows extensions with plug-in components for creation of standardized profiles of 3D Web clients. One of the X3D profiles fully supports the VRML97 specification. However, the expressive power of VRML is restricted by the polygonal representation of shapes and the set of additional predefined parameterized shape primitives such as a sphere or a cube. This provides only ever an approximation to the real shape model. VRML representations of shapes are typically very large, which is an obstacle to collaboration in the design

process through e-mail, bulletin boards or dedicated software. The VRML extension proposal [14] was the first attempt to introduce a protocol for skeletal implicit surfaces with a limited set of operations (warping, Boolean). However, a more general protocol for the full support of FRep models is needed.

HYPERFUN PROJECT

Motivation

In this paper we present a project devoted to developing an open system architecture based on FRep, which, in particular, provides a means for a higher-level shape representation on the Web than currently accepted polygonal models. The motivation for this project stems from considering the following issues:

- **Modeling language.** In principle, one can utilize a universal programming language like C or Java as a modeling language. However, such languages are too complex and have a lot of unnecessary features for application to shape representation. Their generalized compilation tools make the specialized task of constructing shape models unnecessarily difficult. The proposed system architecture is built around the shape models in HyperFun, which is a specialized high-level programming language for specifying FRep models. The language is designed to include all conventional programming constructs and specialized operators necessary for modeling complex objects, yet without any redundant features. Language simplicity is an important criterion, because education is considered to be one of its main application areas.

Function Representation

Function representation (FRep) [1] was introduced in [2] as a uniform representation of multidimensional geometric objects. An object (point set) in multidimensional space is defined by a single continuous real-valued function of point coordinates F(X) which is evaluated at the given point by a procedure traversing a tree structure with primitives in the leaves and operations in the nodes of the tree. The points with $F(X) \geq 0$ belong to the object, and the points with $F(X) < 0$ are outside of the object. The geometric domain of FRep in 3D space includes solids with non-manifold boundaries and lower dimensional entities (surfaces, curves, points) defined by zero value of the function.

A primitive can be defined by an equation or by a "black box" procedure converting point coordinates into the function value. Solids bounded by algebraic surfaces, skeleton-based implicit surfaces, and convolution surfaces, as well as procedural objects (such as solid noise), and voxel objects can be used as primitives (leaves of the construction tree). In the case of a voxel object (discrete field), it should be converted to a continuous real function, for example, by applying the trilinear or higher-order interpolation.

Many operations such as set-theoretic, blending, offsetting, projection, non-linear deformations, metamorphosis, sweeping, hypertexturing, and others, have been formulated for this representation in such a manner that they yield continuous real-valued functions as output [2, 3], thus guaranteeing the closure property of the representation. R-functions originally introduced in [4] provide C^k continuity for the functions exactly defining the set-theoretic operations (min/max functions are a particular case). Because of this property, the result of any supported operation can be treated as the input for a subsequent operation; thus very complex models can be created in this way from a single functional expression.

Recently, a more general “constructive hypervolume” model has been introduced [5], which allows for modeling multidimensional point sets with attributes. Point set geometry and attributes have an independent representation but are treated uniformly. A point set in a geometric space of an arbitrary dimension is an FRep based geometric model of a real object. An attribute that is also represented by a real-valued function (not necessarily continuous) is a mathematical model of an object property of an arbitrary nature (material, photometric, physical etc.). The concept of “implicit complex” proposed in [6] provides a framework for including geometric elements of different dimensionality by combining polygonal, parametric, and FRep components into a single cellular-functional model of a heterogeneous object.

References

1. *Shape Modeling and Computer Graphics with Real Functions,* FRep Home Page: cis.k.hosei.ac.jp/~F-rep/index.html
2. A. Pasko, V. Adzhiev, A. Sourin, V. Savchenko, "Function representation in geometric modeling: concepts, implementation and applications", *The Visual Computer*, vol.11, no.8, 1995, pp.429-446.
3. V. Savchenko, A. Pasko, "Transformation of functionally defined shapes by extended space mappings", *The Visual Computer*, vol. 14, no. 5/6, 1998, pp. 257-270.
4. V.L. Rvachev, “On the analytical description of some geometric objects”, *Reports of Ukrainian Academy of Sciences*, vol. 153, no. 4, 1963, pp. 765-767 (in Russian).
5. A. Pasko, V. Adzhiev, B. Schmitt, C. Schlick, “Constructive hypervolume modelling”, *Graphical Models*, a special issue on volume modeling, 63(6), 2001, pp. 413-442.
6. V. Adzhiev, E. Kartasheva, T. Kunii, A. Pasko, B. Schmitt, Cellular-functional modeling of heterogeneous objects, *Proc. 7th ACM Symposium on Solid Modeling and Applications*, Saarbrucken, Germany, ACM Press, 2002, pp. 192-203.

- **Exchange protocol**. As we mentioned above, polygonal shape representation is dominating the Web now and a more general protocol (in the form of HyperFun programs) supporting exchange of FRep models is needed.
- **Multidimensionality**. FRep naturally supports multidimensional modeling using functions of several variables $F(x_1, x_2, \ldots, x_n) \geq 0$, where x_i 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**. FRep 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, which can be a kind of a simple API. An alternative useful method is developing compilers of HyperFun to standard programming languages such as C or Java with subsequent use of standard conventional means. Direct generation of Java bytecode from HyperFun source code looks especially effective for Web-based applications. In this way, different applications can incorporate FRep models developed independently and received, for example, through an exchange protocol.

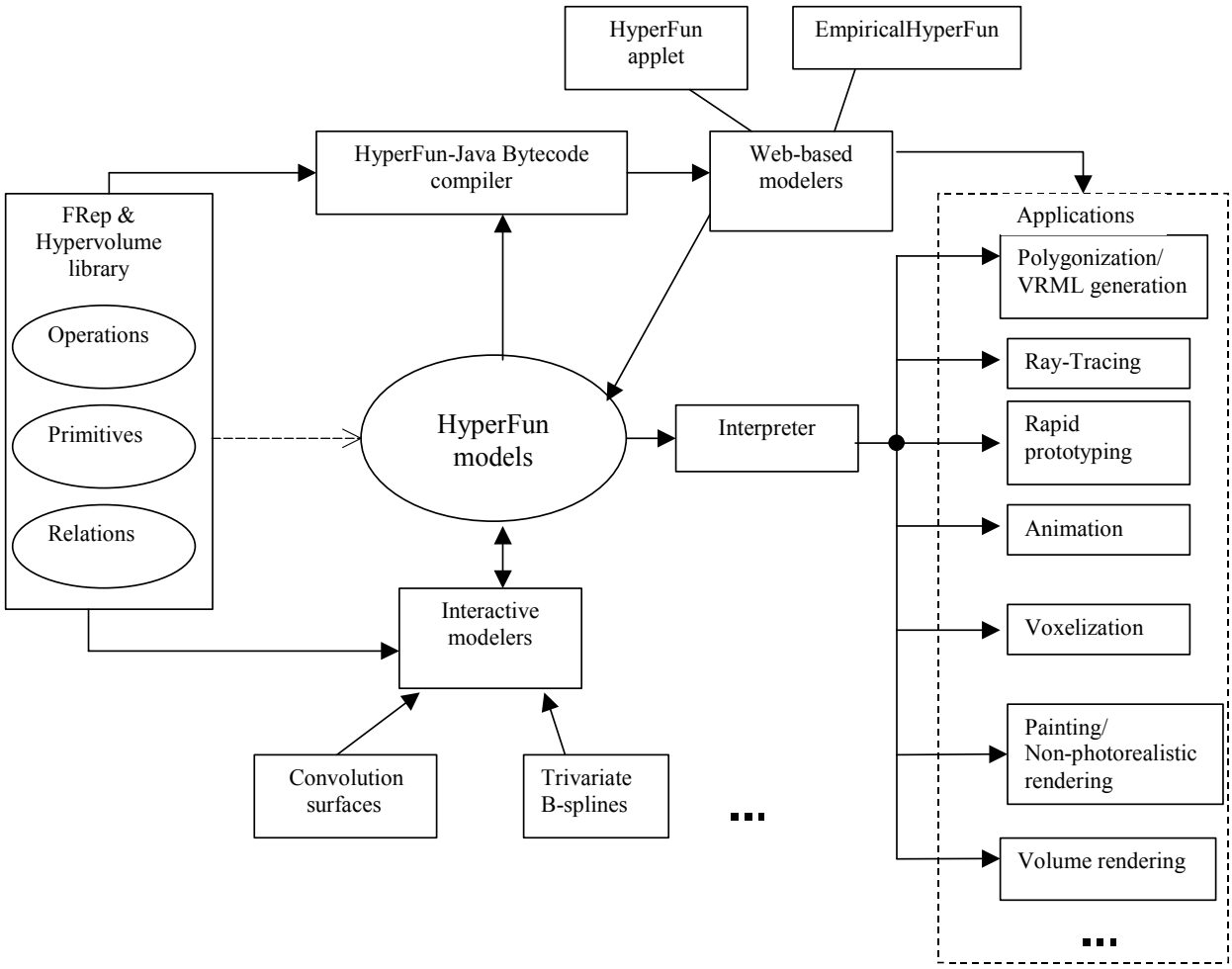


Figure 1: System architecture

- **Extensibility and openness**. Extensibility is the core feature of an FRep 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 interfaces. 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 FRep 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 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 and FRep would be very useful in teaching a number of courses such as analytical geometry, computer graphics, animation, or visualization.

We consider HyperFun as a proposal of an open standard for collaborative Web-based shape modeling. The open source principles are naturally applied in R&D within the Web-based international HyperFun project. Currently, the work is being done on converting the HyperFun software tools into complete open source products and services under some variation of the CGPL license (www.cgpl.org) combining the principles of the free software GPL license with human and environmental rights.

System architecture

The proposed system architecture is shown in Fig. 1. It consists of the following basic groups of components: HyperFun models, FRep library, modelers, interpreters/compilers, and applications. HyperFun models are text files that can be stored and transmitted over a network. The FRep & Hypervolume library contains definitions of primitives, operations, and relations used in HyperFun for defining shapes and attributes. This library is currently implemented in C and Java. The selection of the library depends on the application area and computing environment. The modelers are interactive tools for building FRep 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 Web-based modelers access HyperFun models through the HyperFun-to-Java Bytecode compiler and support collaborative model development. The above-mentioned components are described in the following sections.

HyperFun language and interpreter

HyperFun [1] is a specialized high-level modeling language which allows for a parameterized description of functionally-based multidimensional geometric shapes. While being minimalist in design and easy to master, it supports all the main concepts of FRep. This language was designed to be as simple as possible in order to allow non-specialist users to create models of complex geometric shapes.

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

In principle, the language is self-contained and allows users to build objects from scratch, without the use of any pre-defined primitives and transformations. However, its expressive power is greatly increased by the availability of the system "FRep library" that is easily extendible. Currently, the version of the FRep library in general use contains the most common primitives and transformations of quite a broad spectrum. The user can create his/her own library of objects for later reuse.

Thus, there are library functions implementing conventional CSG primitives (block, sphere, cylinder, cone, and torus) as well as their more general counterparts (ellipsoid, superellipsoid, elliptical cylinder, elliptical cone). Another group implements popular skeletal implicit surface primitives (blobby objects, soft objects, metaballs) including convolution objects with skeletons of different types (i.e. points, line segments, arcs, triangles, curves, and meshes). Primitives derived from parametric functions (i.e. cubic B-spline and Bézier spline objects) have also been implemented. Traditional transformation operations are available, such as rotation, scaling, translation, twisting, stretching, tapering, blending

union/intersection, as well as some more general operations such as non-linear space mapping driven by arbitrary control points. Some useful library functions can help to compose complex attribute functions concerned with creating different texture patterns and color related operations. The on-line manual for the HyperFun language containing a description of the FRep library as well as numerous examples of programs in HyperFun can be found at the HyperFun Project Web page (www.hyperfun.org).

Fig. 2 shows an example of a HyperFun model illustrating different language features. This simple program describes the model of topologically non-trivial 3D object 'my_model' which has been modeled as follows. First, cylinders 'cylz' and 'cylx' are introduced using library functions 'hfCylinderZ' and 'hfCylinderx'; the variable 'cyl' represents union of those cylinders, and the variable 'spcyl' is union of a sphere (defined by the library function 'hfSphere') and the object 'cyl'. Then, the variable 'inside' describes the object 'spcl' with a cylindrical hole 'hole' made by set-theoretic subtraction operation '\'. The object 'cube' is defined by a functional expression; and finally this object is smoothly intersected (using library function 'hfBlendInt' providing blending intersection of two objects) with the object 'inside'.

Application software deals with HyperFun models through using either a built-in interpreter or HyperFun-to-C/HyperFun-to-Java compilers and utilities of the HyperFun API. The HyperFun interpreter has been implemented as a small set of functions in ANSI C. It is quite easy to integrate them into the application software since the developer needs to deal with only two C-functions: one - to implement parsing of the HyperFun program; another - to evaluate the function at the given point.

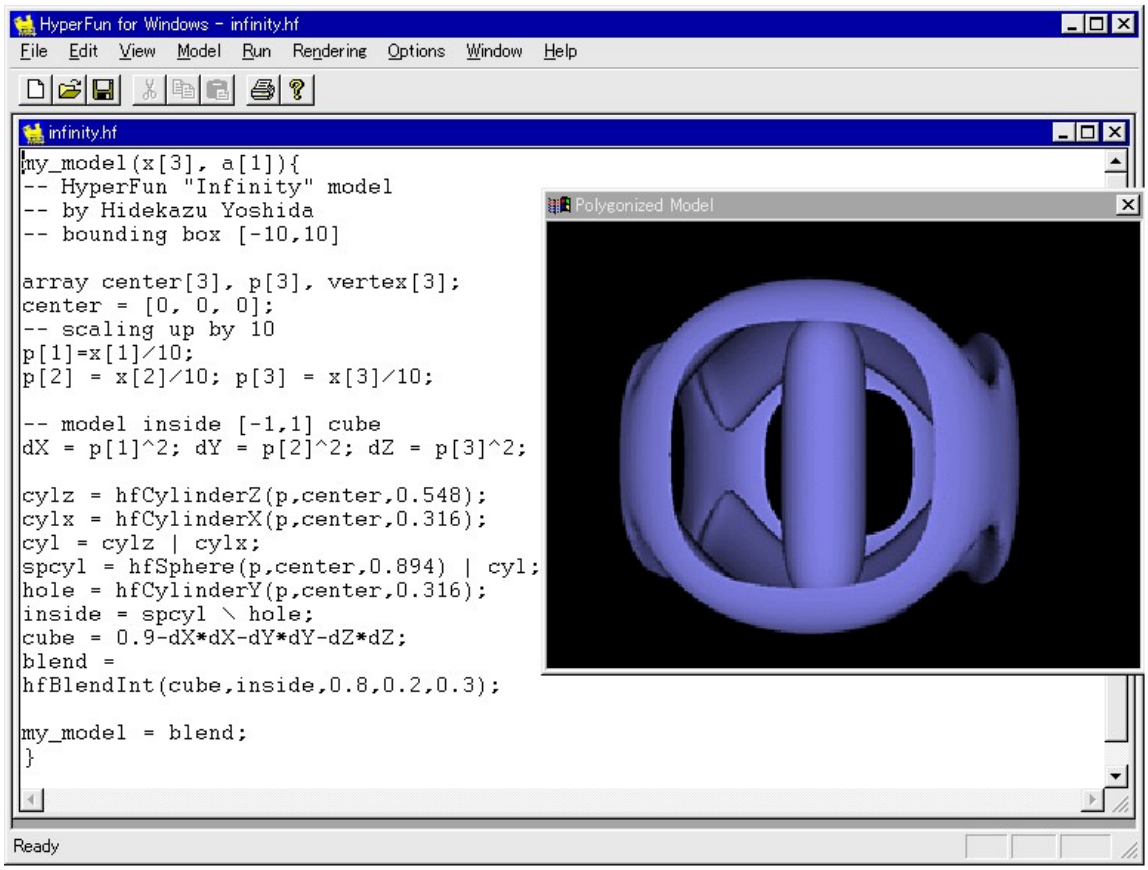


Figure 2: Example of a HyperFun program and the corresponding image of the model in the HyperFun for Windows modeling environment

HyperFun Software Tools

Let us briefly describe the following software tools that are freely available for downloading from the HyperFun Project Web site.

- The *HyperFun Polygonizer* (commonly abbreviated as *"HFP"*). This program polygonizes and displays an object input from a HyperFun file. It has a command line interface allowing the user to define a number of modeling and rendering options. Support for higher dimensional models is also available. The program also makes it possible to output the results in VRML format.

- The *HyperFun for POVRay* (*"HFpov"*) is a plug-in to a popular ray-tracer POVRay (www.povray.org) which makes it possible to generate high quality photorealistic images on an ordinary PC. HyperFun objects can be manipulated as POVRay objects. All ray-tracing options are set using POVRay scene descriptions. Animation capabilities are also available.
- The *HyperFun for Windows* (*"HFW"*) is an interactive system allowing the user to master the FRep modeling concepts using the HyperFun language while working in a conventional MS Windows environment. This program has an interactive windows graphical user interface with pop-up menus and toolbars. This program allows the user to specify an FRep model (using a built-in text editor) in the HyperFun language, to compose complex scenes with multiple objects, to specify visual parameters for subsequent rendering and to generate animation sequences. This tool has been used for student exercises in a number of computer graphics courses in universities in Japan and Russia (see a system screenshot in Fig. 2).
- *Experimental modelers with GUI* support interactive modeling of convolution surfaces and trivariate B-splines with output of HyperFun models containing corresponding primitives.
- *Volume rendering* software for direct rendering of a 3D shape with attributes as a particular case of the hypervolume [9].

The HyperFun toolkit supports a concept of "multimedia coordinates" [2] allowing for a richer interpretation of multidimensional models through generation of variety of visual representations (such as animated spreadsheets) along with audio and other multimedia features. The examples of HyperFun models shown in Fig. 3 have been created and rendered using the described tools to illustrate a variety of applications.

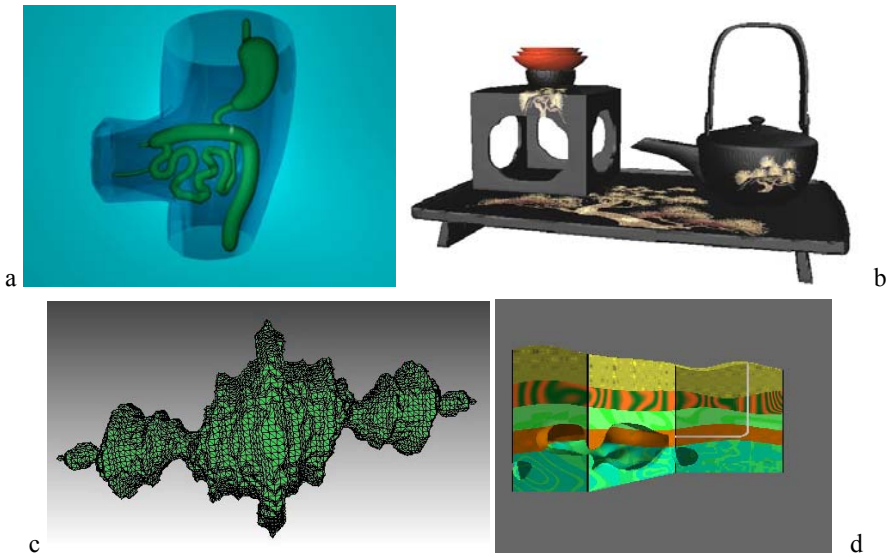


Figure 3: HyperFun models: a) human embryo digestive system (modeled by R. Durikovic and S. Czanner [5]); b) Japanese lacquer ware (modeled in collaboration with G. Pasko [12]); c) 3D quaternion Julia fractals (modeled by F. Delhoume); d) multi-layer geological structure with internal material attribute distribution, an oil well, and cavities (heterogeneous object modeled in collaboration with B. Schmitt).

WEB-BASED MODELING COMPONENTS

In this section, we describe such basic components as HyperFun to Java Bytecode compiler and polygonizer in Java, and HyperFun applet in Java as a demonstration of their usage.

HyperFun to Java Bytecode compiler

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 that is platform independent and appropriate for easy distribution via the Internet. The second (discussed in the next section) 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.

As a first step, a HyperFun to Java source code translator was developed, which required running the standard Java compiler to create the Java bytecode from the source as an external process. In this case, an applet version of a modeler would be impossible to achieve, because Java security models prevent an applet rather than an application from reading and writing local files, and from triggering external processes. To overcome these problems, it was necessary to investigate the

possibility of leaving out the standard Java compiler completely and going directly from HyperFun to Java Bytecode. This process can be achieved entirely in memory without the need to read and write files. After some investigations into the Java bytecode syntax, it became obvious that even though HyperFun is a straightforward language, generating bytecode for a class required careful construction of lookup tables and links to other classes. Fortunately, Open Source Software came to the rescue in the form of Markus Dahm's Byte Code Engineering Library (BCEL) (www.inf.fu-berlin.de/~ dahm/JavaClass), a package of classes that takes care of all the hard parts of writing a compiler that has Java bytecode as its target.

The actions of the HyperFun to Java source code compiler have been replaced by calls to the BCEL API with the result that executable bytecode is generated. Best of all, the time taken to compile a HyperFun script to bytecode is very small using this method and it is possible to load the class into the Java virtual machine from memory. Our measurements of the time difference between using the HyperFun to Java source code translator and HyperFun to Java Bytecode conversion show two orders of magnitude increase of speed for the new technique.

Polygonizer in Java

Polygonization is a process used for generating polygons from a functionally defined surface. The implicit surface polygonizer implemented in Java is based on the algorithm described in [10], which falls within a class called exhaustive enumeration in [3]. The polygonization algorithm is free of ambiguities and heuristics essential to other algorithms of this kind, for example, to the marching cubes (MC) algorithm [8]. To improve the polygonization speed on the phase of the polygons extraction, a look-up table similar to that of the MC algorithm is used. The polygonizer implemented in Java is used in the HyperFun applet and in the EmpiricalHyperFun system described below.

HyperFun applet

To demonstrate the work with the compiler and the polygonizer, a simple Web-based interactive modeler has been implemented as a Java applet (Fig. 4). It support editing of the HyperFun program, polygonization, and viewing on the client computer. The applet consists of three components: user interface, HyperFun to Java byte code compiler, and polygonizer. The user interface has two text areas, one is for editing a HyperFun program and another serves for showing errors found during parsing the program, and the 3D view for the generated polygonal model from a given HyperFun program. To visualize and manipulate with a three-dimensional model, the Java 3D API is used.

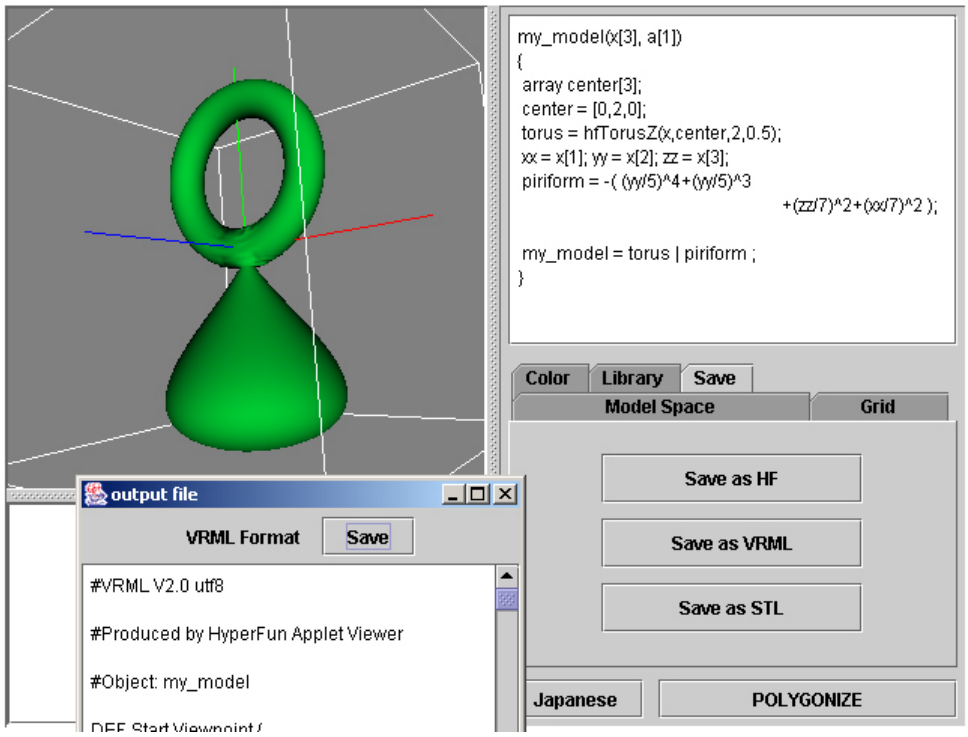


Figure 4: HyperFun applet

The visualization process can be described as follows. First, when the user initializes the polygonization, the HyperFun to Java byte code compiler acts and then a Java class is synthesized from the byte code in memory. The class contains the method representing an F-rep object described in the given HyperFun program. The synthesized class is loaded into the Java virtual machine using Java's dynamic class loading mechanism. The Java virtual machine can translate it into the native code for a certain platform using just-in-time compiler technology. Finally, the polygonizer generates triangle meshes based on

function values obtained from the method representing the F-rep solid. The mesh is then visualized using the Java 3D API. The HyperFun applet is available at http://cis.k.hosei.ac.jp/~F-rep/HF_applet.html

EMPIRICALHYPERFUN

In this section, we provide a description of EmpiricalHyperFun (EHF). There are the following motivations for linking HyperFun to *Empirical Modeling* (EM) in EHF design. EM principles (see side bar) concern the representation of realistic causal dependencies in computer-based models in tandem with the analysis of the agents that participate in observing and changing the model. A HyperFun program on its own can be considered as representing a single shape model. For collaborative Web-based modeling, the creation and instantiation of shapes needs to be managed such that users can share their shape models in a controlled way. Also, a user may wish to build a simulation or animation from a library of shapes that already exist, and define new operations that modify these shapes, without starting completely from scratch. In the construction of virtual or synthetic worlds from these realistic shape models, behavioral models will need to be linked to the shape models that situate the shapes within some form of physically-based or fictional environment.

Empirical Modeling

The *Empirical Modeling* (EM) Project [1] was established at the University of Warwick in 1985. EM is founded on a set of specific principles for modeling on computer systems in such a way that models remain open to change at all times. EM also concerns the analysis of models with respect to *agents* who can observe them, take part in collaborative interaction with and have protocols to change the models. Key components of the computer-based representation are *observables* (values) and the causal *dependency* relationships that exist between them. For example, the observed position of a cup depends on the position of the table on which it rests and the force of gravity. These values and relationships are expressed by *definitions* in a *definitive script* and the use of definitive scripts is known as *definitive programming* [2, 3].

Typically, a computer-based model has a real-world *referent* that is modeled as observed. The real world is full of dependency relationships and if a realistic model of a referent is to be constructed as a virtual computer-based artefact, realistic behaviors, agency, shape, geometry and materials need to be combined in the description of an *artefact* and the environment in which it is situated [4]. EM can provide the underlying behaviors and support the sharing of shape models and worlds by many agents, both human and computer-based. To complement EMs realistic behavioral models, HyperFun for implicit shape modeling can provide an open-ended and realistic shape representations with rich potential for visualization.

To support the principles of EM, several tools (implemented on different platforms and available for downloading from [1]) have been developed to allow the exploration of computer-based models constructed exploiting the modeling concepts. The primary general tool *tkeden* is based on a general C-like definitive notation *EDEN* and also implements:

- *DoNaLD* (a Definitive Notation for Line Drawing) and *DoNaLD3D*,
- *SCOUT* (a SCreen layOUT definitive notation),
- *Sasami* (definitive 3D graphics for constructing OPENGL based models notation),

There is also *dtkeden* tool, which extends tkeden with distributed communication features.

As far as geometry is concerned, such tools as *CADNO* (a combinatorial representation of 3D shape in terms of points, lines and faces with limited wireframe visualization) and *HyperJazz* (an interactive object-oriented tool, which established definitive scripts for describing Frep based shape models [5]) have been developed. In all these tools, compound shapes could be constructed and made to depend on one another, often acting as a visualization of an underlying model [6].

REFERENCES

1. *The Empirical Modeling Project Web site*: http://www.dcs.warwick.ac.uk/modelling

2. W. M. Beynon, “Empirical modelling for educational technology”, *Proc. Cognitive Technology 1997*, University of Aizu, Japan, IEEE, 1998, pp. 54-68.

3. W. M. Beynon, “Empirical modelling and the foundations of artificial intelligence”, *Proc. International Workshop on Computation for Metaphors, Analogy and Agents*, *Lecture Notes in Artificial Intelligence 1562*, Springer, 1999, pp. 322-364.

4. W. M. Beynon, R. I. Cartwright, P-H. Sun and A. Ward, “Interactive Situation Models for Information Systems Development”, *Proc. SCI'99 and ISAS'99*, vol.2, Orlando, USA, July 1999, pp. 9-16.

5. V. Adzhiev, A. Pasko, A. Sarkisov, “HyperJazz project: development of geometric modeling systems with inherent symbolic interactivity”, *Proc. CSG96*, Information Geometers, UK, 1996, pp. 183-198.

6. R. I. Cartwright, *Geometric aspects of empirical modelling: issues in design and implementation*, Ph.D. Thesis, University of Warwick, UK, September 1998.

Java Maintainer Machine API

The Java Maintainer Machine API (JaM) is one of the first object-oriented tools supporting EM. JaM provides a mechanism whereby a new underlying algebra for a definitive notation can be constructed by a specialist developer. This is achieved through the implementation of classes representing new data types and new operators. The current implementation called JaM2 is a programming API that is not bound to a notation, although a default parser and notation is provided as part of the broader package. New data types and operators for an underlying algebra can be registered on-the-fly with an instance of a JaM2 script. Once this has taken place, JaM2 provides services that allow many users to share interaction with the same instance of a script and processing of the models they construct takes advantage of multi-processor architectures through the use of threads.

As an API, the JaM2 consists of a number of Java packages. These include a parser supporting a dedicated scripting language, implementations of common basic types and library of useful utility operations. The main benefits of JaM2 are:

- Application-domain objects are represented as an entities rather than through a contrived and sometimes complex naming convention;
- Operator code is compiled rather than interpreted and hence it executes more efficiently;
- Explicit representation of dependencies between objects of any well-defined underlying algebra;
- Registration of data types and operators describing underlying algebras on-the-fly;
- Description of data values by parameter sets, allowing the notation and API calls of JaM2 to be separated;
- Organization of definitions describing dependencies into directories;
- User, group and other permissions similar to a multi-user operating system;
- Automatic update of dependencies between objects when state is changed, considering a set of definitions simultaneously and executed efficiently in parallel where facilities are available.

EmpiricalHyperFun scripts

EHF scripts are written in a combination of two notations:

-*HyperFun* for the description of shape primitives, operators for shape and bounding box calculations.

-*JaM2* as a script notation for building classes of shape, compiling them, instantiating them and controlling their realization.

An *EHF* script has a *JaM2* data type called *HFShape* that represents a classification of a type of shape in terms of a HyperFun script and its parameterization. This includes its bounding box calculation and the name of a data type that a user would like to be available from the current notation. The *Sphere* data type, for example, can be represented by an instance of the *HFShape* data type in the following way:

```
sphere_shape is HFShape {
name "Sphere"
hfparameters [
{"centre", type "DOUBLE_LIST"},
{"radius", type "DOUBLE"}
]
hyperfun "sphere(x[3], a[0]) {
x[1] = x[1] - centre[1];
x[2] = x[2] - centre[2];
x[3] = x[3] - centre[3];
sphere = radius * radius - x[1] * x[1] -
x[2] * x[2] - x[3] * x[3];
}"
minx "minx = centre[1] - radius;"
maxx "maxx = centre[1] + radius;"
miny "miny = centre[2] - radius;"
maxy "maxy = centre[2] + radius;"
minz "minz = centre[3] - radius;"
maxz "maxx = centre[3] + radius;"
};
```

In this example, the parameters describing everything required to create an instance of a new class of shape are given as follows:

name: the name of the new shape data type to be created.

hfparameters: a list of the parameters used to describe instances of the shapes, defined in terms of their name and their type. In the example, this is a list of double values called centre and a single double value called radius. These parameters are available within the HyperFun script that describes the shape and in the bounding box calculations. Note that the standard HyperFun object's parameter *a* is automatically created (although in this particular example the object is not parameterized.)

hyperfun: the HyperFun script describing the recipe for making instances of the shape model.

minx … maxz: short HyperFun scripts describing how to find the three dimensional bounds of the shape, where these define planes orthogonal to the x, y and z axis of the space. A space searching/shrink wrapping function will be available here as default.

In essence, the instance of the HFShape data type can be considered as a representation of the source code for a class of HyperFun shape. It cannot be interpreted or visualized in its current form. To do this, it must be compiled using the HyperFun to Java compiler described above. This compiler generates from the HyperFun script and embeds it in the implementation of a new JaM2 data type, called *Sphere* for the example. The class generated then has: fields *centre*, *radius* and *a*; the HyperFun script is compiled to an executable method; the bounding box calculations compiled to executable methods.

The HyperFun to Java bytecode compiler is embedded in the *compileHF* operator available in EHF scripts. To create the compiled version, it is only necessary to issue the following definition:

```
Sphere_exec is compileHF(sphere_shape);
```

As this is a definition in EM terms, if the HyperFun source for the parameter ever changes, the executable version is updated. As a side effect of the use of this operator, EHF ensures that the data type *Sphere* is registered with the underlying JaM2 engine and that it is immediately available for instantiation.

To realize the sphere through a visualization engine, an instance of it must be constructed and this instance added to a scene of viewable objects. The instantiation can be done in one of two ways.

1. Create an instance of the *Sphere* data type, where an update of the HyperFun describing the shape data type does not cause an automatic re-evaluation of the data type:

```
s1 is Sphere { centre 0.0 0.0 0.0
radius 3.2 };
```

2. Use the special *buildHF* operator that ensures that if the executable code for the shape is altered, any shape built from the underlying HyperFun shape will be recalculated:

```
s2 is buildHF(sphere_exec,
"centre", 0.0 0.0 0.0,
"radius", 3.2);
```

Operations and Libraries

It is possible to define more than just shape primitives using instances of EHF's *HFShape* data type. New operations on shapes can be introduced in exactly the same way as the classification of shape data types. This is made possible by the fact that with FRep, an operation on a shape is a shape in its own right, not simply a data structure to be traversed during rendering. The following example illustrates how a simple affine transformation, a translation, can be constructed using the *HFShape* data type:

```
translate_op is HFShape {
name "Translate"
hfparameters [
{"original", type "SHAPE"},
{"shift", type "DOUBLE_LIST"}
]
hyperfun "translate(x[3], a[0]) {
x[1] = x[1] - shift[1];
x[2] = x[2] - shift[2];
x[3] = x[3] - shift[3];
translate = original(x, a);
}"
```

```
minx "minx = original.minx + shift[1];"
miny "miny = original.miny + shift[2];"
minz "minz = original.minz + shift[3];"
maxx "maxx = original.maxx + shift[1];"
maxy "maxy = original.maxy + shift[2];"
maxz "maxz = original.maxz + shift[3];"
};
```

The type of the parameter original for shape type *Translate* is not a number or an array, but instead it is another shape that can be sampled by executing its defining HyperFun form within the new shape's HyperFun script. In this way, a new classification of shape data type is constructed that must have an existing shape as one of its instantiating parameters. The bounding box of the existing shape can be used to calculate the bounding box of the new shape. To make use of the translation data type as defined above, the following EHF script can be used:

```
a is Point3D {-1.0 -1.0 -1.0};
b is Point3D {1.0 1.0 1.0};
c is midpoint(a, b);
s1 is buildHF(sphere_exec,
"center" c
"radius" distance(c,a) );
traslate_exec is compileHF(translate_op);
t1 is buildHF(translate_exec,
"original", s1,
"shift", [1.0, 1.0, 1.0]);
```

This segment of script firstly introduces values `a` and `b` which are constructed with absolute explicit values of type `Point3D`. Another value `c` is also of type `Point3D` but it is defined by a dependency on the values of *a* and *b*. Whatever subsequently happens to the values of `a` and `b` following the definition of `c` and throughout the lifetime of this definition of `c`, the value of `c` will always remain a point located exactly half way between `a` and `b` – this is the feature point of *definitive programming*. The surface of sphere *s1* will always pass through the points `a` and `b`. Then we create a copy of sphere *s1* called *t1* with its centre located at *(1, 1, 1)* rather than the original *(0, 0, 0)*. If `s1` is redefined to have a new centre point `c` (for instance, by redefinion of `a` or `b`), `t1` will always have a relative centre that is shifted by vector (1; 1; 1). Because of the dependency maintenance in use in the example, it is possible to modify the code that defines the translation and shape *t1* will be updated to correspond to this change. Such a modification can be done by any user who has a right for such an action at any time, and other users in the network sharing the model will immediately get a modified code too.

The ability to create new parameterized shape primitives and operators allows for the possibility that libraries of shapes and operators can be defined. These libraries can be placed onto the Internet and distributed between interested groups simply by cutting and pasting relevant sections of EHF scripts. It may even be appropriate to consider replacing the existing HyperFun library of shapes and operations with versions described by EHF scripts. A user could then bootstrap the shape modeling environment so that it contains the operators they are interested in. A prototype implementation of EmpiricalHyperFun as a Java applet is shown in Fig. 5. Besides the image, the applet contains two text windows: the upper one for the system output, and the lower one for input of new script lines by the user on the fly. In Fig. 5 the model of a 'helicopter' toy is firstly composed using HyperFun language and then interactively modified on the fly by just inputting one line of code redefining the line with set-theoretic subtraction thus removing the toy's body (Fig. 6b).

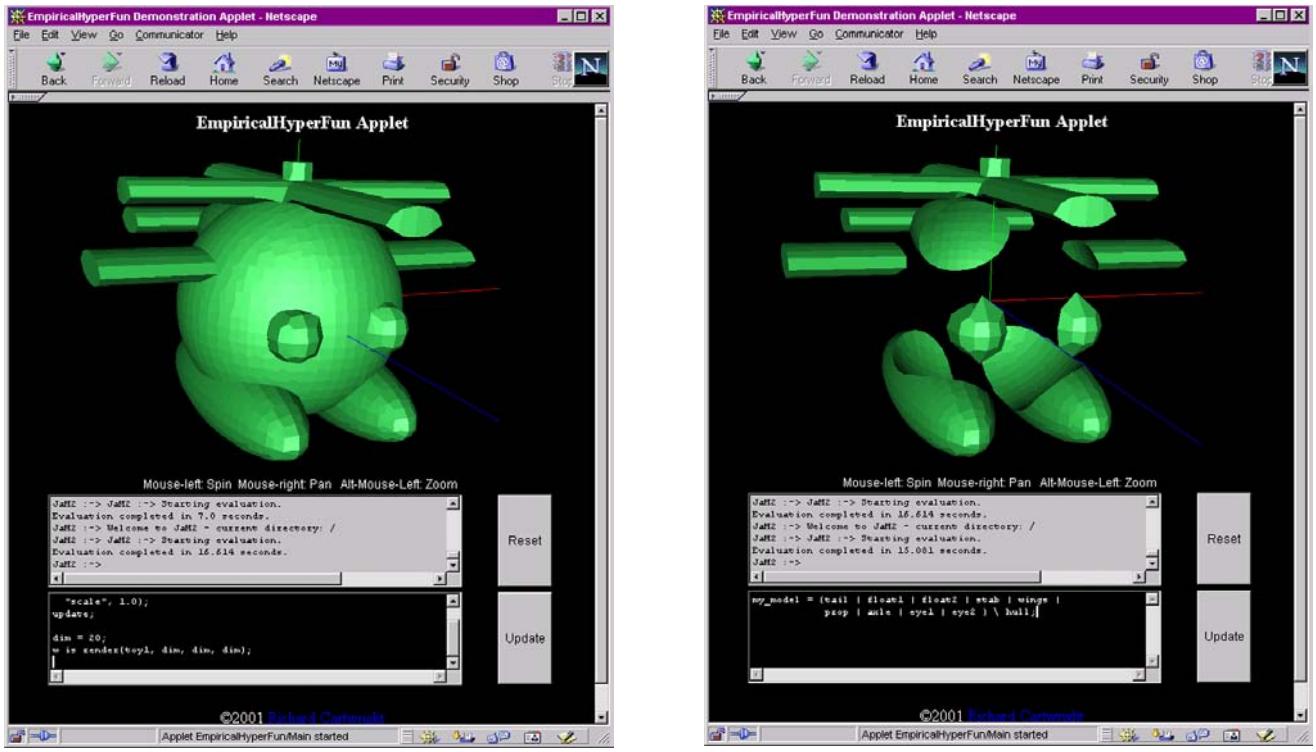
Architectures for collaborative shape modeling

Let us briefly discuss architectures for using EHF in a distributed way. The distribution supposes both collaborative shape modeling and clustering for generating realizations of shape models. All systems are based on JaM2, which is designed to support the collaborative modeling process. Rendering can take two forms: polygonization with further using the Java3D API and ray tracing, for example, on the base of the open source Raja project (http://raja.sourceforget.net/).

The simplest model for distribution is a single user client. A user can download EHF models, render them locally and send them back to the server. The advantage of this system is that EHF models are small and so can be downloaded much faster than an equivalent VRML model. Ray tracing to any reasonable quality is likely to be a slow process and so local polygonization is the best approach to realization. Involving a local cluster of computers allows a user to scale their interactive response with a modeler to the performance they desire and even perform rendering of animation frames.

In more sophisticated architectures, the models are shared by the users through a server that acts as a broker for their interaction with other users and as a rendering farm. Users can share shape primitives (instances of HFShape), instanciated

shapes, scenes, and animation parameters. The JaM2 engines are running on a rendering farm controlled directly by the broker and receive the same definitions, with the exception of the configuration of the spatial sub-division for rendering. Another possible architecture is to provide a more intelligent broker that can deal with modeling clients and rendering clients joining and leaving a shape modeling network. The broker would need to optimize the performance of the rendering to suit the systems and network infrastructure. The architecture supports the full range of collaborative modeling described above and could supply a means for people to trade rendering time with other users. The system could work by distributing work units in the same manner as SETI@home project (http://setiathome.ssl.berkeley.edu), which is a scientific experiment that uses Internet-connected computers worldwide in the Search for Extraterrestrial Intelligence (SETI).



a b

Figure 5: Prototype implementation of EmpiricalHyperFun as a Java applet:
interactive modeling of a toy with its body present (a) and removed (b) through the script redefinition.

CONCLUSION AND FUTURE WORK

In this paper, technology to enable a new software culture for shape modeling has been proposed. We described the state of the art of the HyperFun project devoted to developing an open system architecture for functionally based shape modeling. The software described will allow users to treat Internet-based shape modeling as a resource, collaborate in the development of shapes, and extend their models in an open way meaning that the software is never finished.

We presented the *HyperFun* language as a basis of a lightweight protocol for exchanging function-based models of surfaces, solids, volumes, and hypervolumes. Web-based modeling components such as HyperFun to Java Bytecode compiler and the polygonizer in Java were combined for demonstration into a simple HyperFun applet available at Web.

The last part of the paper presented an advanced Web-based modeling environment called *EmpiricalHyperFun* (EHF), which combines HyperFun with Empirical Modeling principles. It is envisaged that the EHF concept will be both of educational benefit in teaching people the concepts of shape modeling, and valuable in some form with existing applications for shape modeling.

For conformity to the current standard, our tools provide for the conversion of HyperFun models to the VRML format. Note that VRML besides the description of the shapes contains additional information about the graphical scene such as light sources, background, texture, and other data, which are not covered in HyperFun. The next logical step in the geometric protocol development is to include HyperFun objects as special nodes in VRML. There are two independent proposals [17, 7] of such a node. The node proposed in [7] has extensible structure and is supported by a prototype plug-in to a commercial

VRML browser. One of the X3D standard [16] profiles fully supports the VRML97 specification. In the context of this paper, an open question is whether HyperFun should be implemented as a separate X3D profile or as a special VRML node within the corresponding X3D profile.

The proposed approach and software tools do not constrain creative exploration of shape models to a particular representation, rendering technique or to a limited set of primitives and operators. This will allow users to treat Web-based shape modeling as a resource that enables collaboration in the development of shapes and configurable extension of their modeling environment in an open way.

ACKNOWLEDGEMENTS

We would like to thank Kazuhiro Mochizuki and Mio Hiraga who took part in the HyperFun applet development. The HyperFun models used in the illustrations have been made by or in cooperation with Hidekazu Yoshida, Silvester Czanner, Roman Durikovic, Galina Pasko, Frederic Delhoume, and Benjamin Schmitt. Our sincere thanks go to Jody Vilbrandt for proofreading the manuscript. The authors are thankful to the anonymous referees for their helpful comments.

REFERENCES

1. V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, V. Savchenko, “HyperFun project: a framework for collaborative multidimensional F-Rep modeling”, *Proc. Implicit Surfaces ’99, Eurographics/ACM SIGGRAPH Workshop*, ed. by J. Hughes, K. Schlick, 1999, pp. 59-69.

2. V. Adzhiev, A. Ossipov, A. Pasko, "Multidimensional shape modeling in multimedia applications”, *Multimedia Modeling '99*, ed. by A. Karmouch, World Scientific, Singapore, pp. 39-60.

3. J. Bloomenthal et al., *Introduction to Implicit Surfaces*, Morgan-Kaufman, 1997.

4. A. Bowyer, *SvLis -- Introduction and User Manual*, Information Geometers Ltd and University of Bath, 1999.

5. R. Durikovic and S. Czanner, “Implicit surfaces for dynamic growth of digestive system”, *Proc. Shape Modeling International*, IEEE CS, 2002, pp. 111-117.

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

7. F.M. Lai, A. Sourin, Function-defined shape node for VRML, *Proc. EUROGRAPHICS 2002, Short Presentations*, 2002, pp.71-79, http://www.ntu.edu.sg/home/assourin/FVRML.htm

8. W. E. Lorensen, H. E. Cline, “Marching cubes: a high resolution 3D surface construction algorithm”, *Computer Graphics (Proceedings of SIGGRAPH ’87)*, vol. 21, no. 4, 1987, pp. 163-169.

9. A. Pasko, V. Adzhiev, B. Schmitt, C. Schlick, “Constructive hypervolume modelling”, *Graphical Models*, a special issue on volume modeling, vol. 63, no. 6, 2001, pp. 413-442.

10. A. Pasko, V. Pilyugin, V. Pokrovskiy, “Geometric modeling in the analysis of trivariate functions”, *Computers and Graphics*, vol.12, nos.3/4, 1988, pp.457-465.

11. A. Sherstyuk, "Interactive shape design with convolution surfaces", *Proc. Shape Modeling International '99*, University of Aizu, Japan, IEEE Computer Society, 1999, pp. 56-65.

12. C. Vilbrandt, G. Pasko, A. Pasko, P.-A. Fayolle, T. Vilbrandt, J. R. Goodwin, J. M. Goodwin, T. L. Kunii, "Cultural heritage preservation using constructive shape modeling", *Computer Graphics Forum*, vol. 23, No.1, 2004, pp. 25-41.

13. *ISO/IEC. Information technology | computer graphics and image processing | the virtual reality modeling language (VRML)* - part 1: Functional specification and UTF{8 encoding. ISO/IEC 14772-1:1997, 1997.

14. B. Wyvill, A. Guy, "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.

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

16. *X3D – Extensible 3D, New-Generation Open Web3D Standard, 2001*, http://www.web3d.org/x3d/

17. J. Zara, *HyperFun project and VRML Language*, 2001, http://www.cgg.cvut.cz/~zara/HyperFun/