Constructive texturing based on hypervolume modeling

B. Schmitt
LaBRI, University of Bordeaux I,
Talence, France.
schmitt@labri.fr

A. Pasko
Faculty of Computer and Information Sciences,
Hosei University, Japan.
pasko@k.hosei.ac.jp

V. Adzhiev
National Centre for Computer Animation,
Bournemouth University, UK.
vadzhiev@bournemouth.ac.hk

C. Schlick
LaBRI, University of Bordeaux I,
Talence, France.
schlick@labri.fr

Abstract

The concept of solid texturing is extended in two directions: constructive modeling of space partitions for texturing and modeling of multidimensional textured objects called hypervolumes. A hypervolume is considered as a point set with attributes of both physical (density, temperature, etc.) and photometric (color, transparency, diffuse and specular reflections, etc.) nature. The point set geometry and attributes are modeled independently using real-valued scalar functions of several variables. Each real-valued function defining geometry or an attribute is evaluated at the given point by a procedure traversing a constructive tree structure with primitives in the leaves and operations in the nodes of the tree. This approach provides a framework for modeling, texturing and visualization of 3D solids, time- dependent and multidimensional objects in a completely uniform manner. We introduced a special modeling language and implemented software tools supporting the proposed approach. The concept of constructive hypervolume textures is independent of the geometry representation. We provide examples of textured Frep and BRep objects as illustrations.

1. Introduction

A lot of research efforts in computer graphics have been devoted to the improvement of the realism of synthetic images. It appeared that sometimes to apply only colors to visible surfaces is not enough. Thus, other shading parameters were introduced, such as transparency, diffuse, specular, or reflectance properties. Several existing methods of texturing are dependent on the representation of the object.Some methods, described in Section 2, may be applied to polygonized objects, others to implicit surfaces. One of them, called solid texturing [20, 21], allows for texturing objects, regardless of their nature. Furthermore, this method provides realistic textures patterns, such as wood, marble or water, and very convincing effects, such as fog or fire. The concept is to define a color space partition where some shading properties are defined in each subset. Nevertheless, the method suffers from a lack of flexibility in its application. For instance, if an object is deformed, it may be very difficult to apply the same modification to the shading parameters space.

In this paper, we introduce a new texturing technique extending the concept of solid texturing, where the space partition appears naturally as a part of a constructive multidimensional model of a point set with attributes called hypervolume. The general model discussed in Section 3 supports modeling 3D, time-dependent and multidimensional point sets with arbitrary attributes presented as scalar functions. The models of geometry and attributes are similarly constructed using primitives and operations and represented by individual construction trees. The space partition for texturing is then defined by a procedure traversing corresponding construction trees and assigning necessary attributes to the given point in space.

In Section 4, we apply the discussed constructive hypervolume model to tackle the long-standing problem of texturing static three-dimensional, time-dependent, and multidimensional implicit surfaces and more general FRep solids [18]. However, there are no obstacles in applying the proposed approach of constructing textures to parametric surfaces, BRep and other solids, because representations of point sets and their attributes can be completely independent. Section 5 describes the extension of the HyperFun language [27] for supporting constructive hypervolume modeling, corresponding software tools and obtained results.

2. Existing Texturing Techniques

In computer graphics, several texturing techniques exist to embellish geometric objects. In the following, we will use the texture definition given in [9], where the term texture is defined as "anything that is evaluated at a point using only information local to that point". Texturing is usually decomposed into two steps, known as the texture pattern definition and the texture application, or mapping.

To create a pattern, one can either scan a picture (or even hand-paint it) or use a function to generate it automatically. Several methods exist to create such functions; some of them are based on Fourier synthesis [3], or on a fractal subdivision [11]. In Eberts book [8], a complete description of different methods of composing such functions can be found. Very realistic textures can be obtained, because they take into account the fact that many natural materials are non-homogenous, and may have complex internal structures. This set of methods is known as Procedural Pattern Generation.

Once the pattern is defined, one has to think about how to apply it to a surface. The texture mapping introduced in [5] was the first solution proposed allowing us to apply some 2D color patterns (digital images or procedural function) to a surface. An alternative concept is that of solid texturing initially introduced in [20, 21]. This method procedurally defines a texture pattern in the 3D object space, using solid texture functions. Given a point (x, y, z) on the surface of an object, the shading values are defined as $T_i(u, v, w)$, where the coordinate space is mapped to the shading space using a simple affine mapping. The main advantage of this method is that it does not require any additional complex steps for defining the mapping from the 3D space to a 2D space, and solves many problems with the previously mentioned methods, such as, for instance, continuity of the texture between patches. Another valuable property of this method is that it can be applied to any kind of surface. Indeed, solid texturing can be thought as the functional definition in space for a procedural texture to be defined for any given point. But the way we define the space partition is arbitrary, and does not rely on a robust framework. There is an alternative method for defining a space partition with a more robust structure, called A BlobTree but it is only applicable to implicit surfaces. A BlobTree [31] is a hierarchical tree structure with skeletal implicit surface primitives as leaves and operations (blending, warping, and Booleans) as nodes. A special attribute node can be placed anywhere in any non-terminal position, and the values specified by this node will be the default attributes for nodes lower down the hierarchy. Another attribute node deeper down this tree can override a shallower attribute node. Such a scheme supposes a fixed discipline of assigning attributes to the entire implicit surface rather to than particular space points.

In the following section, we introduce a texturing process applicable to surfaces, 3D solids, animated and other multidimensional objects. In some sense, the proposed method extends the solid texturing method, but in our case, we use a special constructive tree for each shading attribute to define the space partition.

3. Constructive Hypervolume Model

The simplest volume object can be thought as a 3D point set with a single scalar attribute given at any of its points. Here, the 3D point set can represent geometry of a real world object. The scalar value can represent density, temperature or any other physical characteristic. A more general hypervolume object is a multidimensional point set with multiple attributes given at any of its points.

In general, the hypervolume model can be expressed as :

$$(G, a_1, a_2, \ldots, a_k), \tag{1}$$

where G is a multidimensional point set and a_i is an attribute. In 3D case, a point set G can be defined using any existing representational schemes for solids: BRep (polygonal or curvilinear boundary surface), CSG (Constructive Solid Geometry), spatial partitioning (voxels, octrees, etc.), generative models (parametric function representation), ray representation, FRep (real-valued function representation), and others [22, 25, 23, 15, 18]. While some of the traditional representations such as CSG or BRep have multidimensional extensions [30, 10], the generative model [25] and FRep [18] have been initially formulated as multidimensional models. Attributes a_i can be represented by scalar, vector, or tensor fields defined on the multidimensional point set G. For example, one needs to introduce a 4D point set and at least three scalar values to model a textured time-dependent object.

This model is general enough to cover objects considered in such different research areas as solid modeling, heterogeneous objects modeling, and volume graphics. In traditional solid modeling, an object is supposed to be homogeneous inside, for example, with the fixed density. The proposed model covers such solids, if a constant scalar field is applied. To model heterogeneous solids and other 3D objects with attributes addressed in the general "object model" of [13], one has to define different properties as scalar fields on the level of primitives and introduce the rules of combining these scalar fields when doing set-theoretic, blending and other operations on such primitives. Usually, quite simple point sets such as rectangular blocks are processed in volume graphics. Intensity and other visual characteristics are defined by scalar values in the nodes of the discrete grid or in scattered points inside the point set. A point subset with

the scalar value equal or greater a given threshold is visualized.

In Constructive Volume Geometry (CVG) [6], a spatial object is defined as a tuple of scalar fields defined in 3D space. Special attention is paid to the first field in the tuple, which is an opacity field specifying the visibility of every point in space. In a conceptual distinction from it, the important feature of our hypervolume model is that object's geometry and its visual and physical characteristics are represented independently. This can easily be found in reality where, for example, the shape of the object does not necessarily predefine its color and vice versa. Such purely optical characteristics as transparency or opacity are usually neither dependent on the geometric shape nor define it as was suggested in [6]. There are scalar fields directly connected to the object's geometry. The density field and other scalar fields proportional to it determine the object's boundary and its internal structure. The other possible source of confusing the shape model with the model of some property is the fact that implicit surface models [4] generate scalar fields. Such a scalar field defines the object's boundary (e.g., as a zero value isosurface) but in general does not have optical or other physical meaning and cannot be used to represent properties. Moreover, this scalar field can be scaled or undergo some non-linear transformations without changing the object's geometry, which is not acceptable in the case of the physical property model.

A survey of modeling techniques for point sets with attributes and details of the hypervolume model with the outline of the formal framework can be found in [17]. Here, we use a specific implementation of the presented hypervolume model:

$$(F, s_1, s_2, \ldots, s_k), \tag{2}$$

where $F(X)$ and $s_i(X)$ are real-valued scalar functions of point coordinates X in n-dimensional space. We use here FRep [18] to represent point sets. Therefore, F is at least C^0 continuous function positive inside the point set, negative outside, and having zero value on its boundary. Functions $s_i(X)$ represent attributes and are not necessarily continuous.

The main distinctive feature of FRep is that the real-valued function F defining the point set 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. This tree is similar to one used in CSG and is created during the construction process of the object. In contrast to CSG, the sets of primitives and operations are not fixed and can easily be extended without redesigning the modeling system. Solids bounded by algebraic surfaces, skeleton-based implicit surfaces, and convolution surfaces, as well as procedural objects (such as solid noise), swept objects, and volumetric (voxel) objects can be used as primitives (leaves of the construction tree). Many operations such as set-theoretic, blending, non-linear deformations, metamorphosis, sweeping and others have been formulated for this representation in such a manner that they again yield continuous real-valued functions as their output [18, 24].

The attribute functions s_i can be defined in a similar way. Each attribute has an associated tree with primitives and operations that can be borrowed from FRep and extended by attribute specific ones. The tree structure reflects the construction logic of the attribute definition. The function s_i is evaluated at the given point by a tree traversing procedure. Thus, symmetry in treating the point set and its attributes can be achieved in a sense of the constructive nature of the definition and internal representation.

In this paper, we apply the discussed constructive hypervolume model to tackle the long-standing problem of texturing static and time-dependent implicit surfaces and FRep solids. Here, $F(X)$ represents the object geometry and $s_i(X)$ may represent any attribute suitable for texturing such as color, opacity, diffuse and specular reflections, and others. However, there are no obstacles in applying the proposed approach of constructing textures to parametric surfaces, BRep and other solids, because representations of point sets and its attributes can be completely independent.

4. Constructive Hypervolume Texturing

In this section, we introduce the constructive hypervolume texturing on the basis of the described general model. Using the solid texturing defined in [20, 21], we propose to build a constructive tree in order to define a partition of the space where the object to be textured is placed. Most of the following examples are concerned with colors and opacity, but the extension to other shading parameters such as ambient reflection, diffuse and specular reflectance is straightforward. Figures illustrate application of this concept to all of these shading parameters.

4.1. Constructive Solid Texturing

In this subsection, we deal only with 3D objects, particularly implicit surfaces and FRep solids.

When applying solid texturing to an object, one has to create a space partition of the object space, where each subset contains a different material property. Then, the shading parameters are computed for each point in the space, where point coordinates determine which subset it belongs to, and eventually the corresponding attributes are determined. The main difficulty is how to define such subsets. Actually, a subset can be thought as a solid object that for each point in the space has to provide an answer the following question: "Is this point inside or outside the subset?" In the affirmative case, the corresponding solid texture function is applied. It

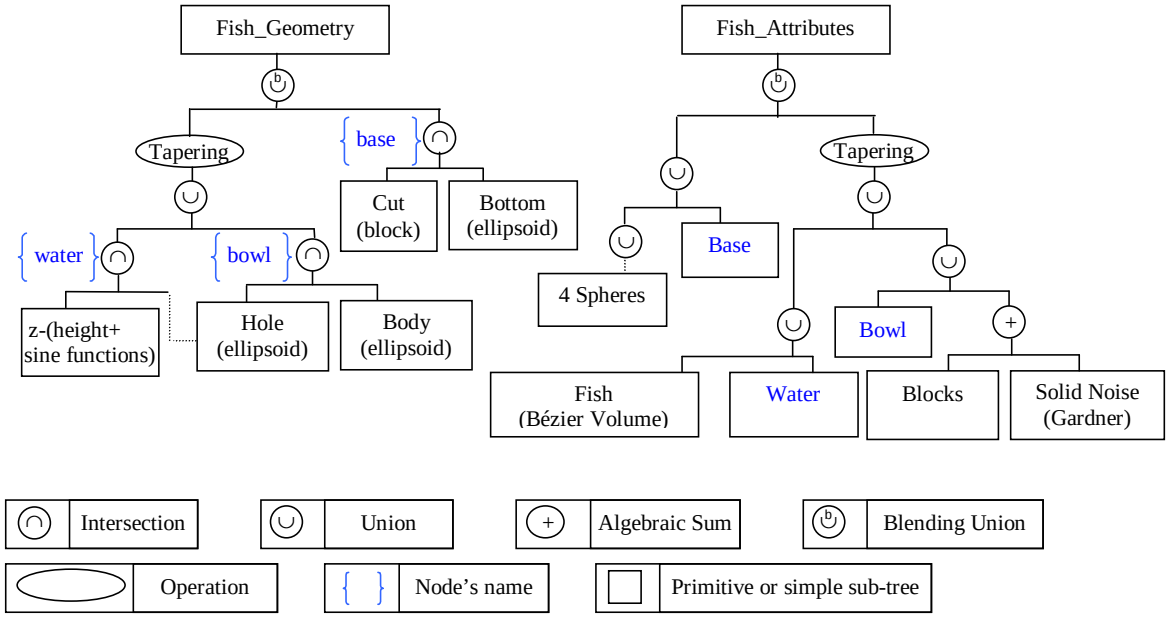


Figure 1. Constructive trees for a model in Fig. 2. (Left) Geometric tree. Dots line between the "hole" and the node "water" indicates that the defining function of the "hole" is used for the intersection with the "water" and with the "body". (Right) Texturing tree, composed of internal nodes of the geometrical tree, and of some other primitives. Different attributes are set to each node.

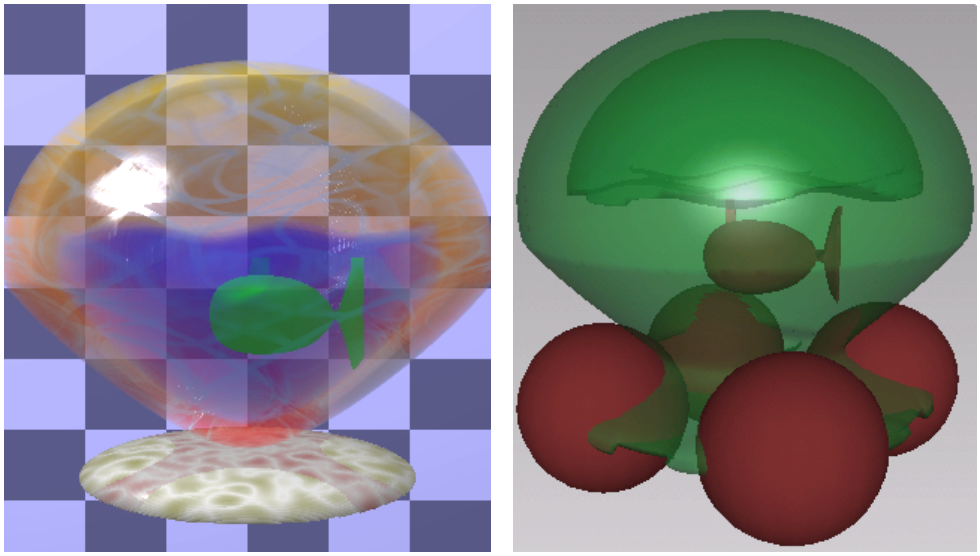


Figure 2. Example of space partitioning. (Left) The geometric model is defined as a blending union of an ellipsoid and a sphere. The water, the fish and the different mosaic patterns are defined using a tree composed of spheres, Bézier volume, blocks and solid noise. Details of the tree can be found in Fig. 1. (Right) Visual reprensation of the constructive texturing tree shown in Fig. 1. Common parts of both trees are in green, parts dedicated to the attribute one are in red. One part of the attributes tree is defined as a union of four spheres and of the ellispoid used for the geometry. A set of attributes is defined for points of the ellispoid inside the spheres, and another set for outside.

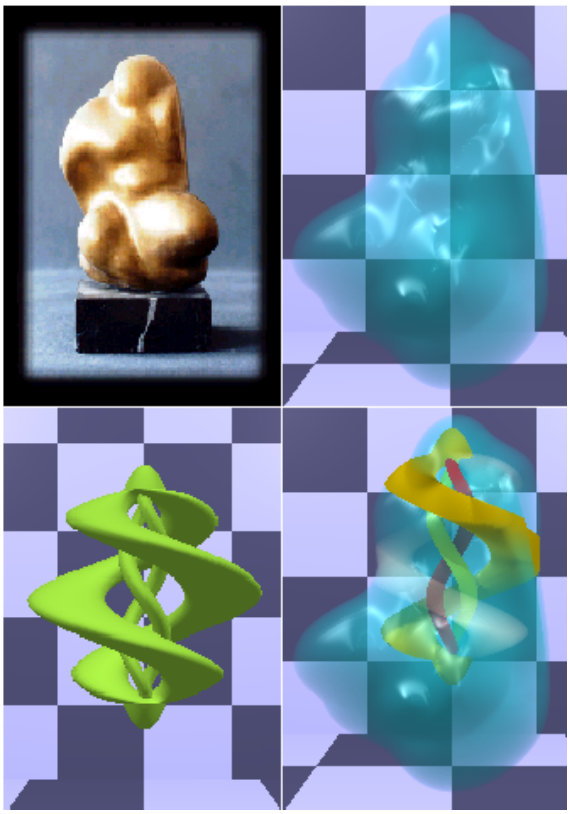


Figure 3. Constructive solid texturing. (Top-Left): Original shape (photo). (Top-Right): Original static 3D shape. (modelled) (Bottom-Left): Constructive 3D solid representing the space partitioning for texturing. (Bottom-Right): Textured shape.

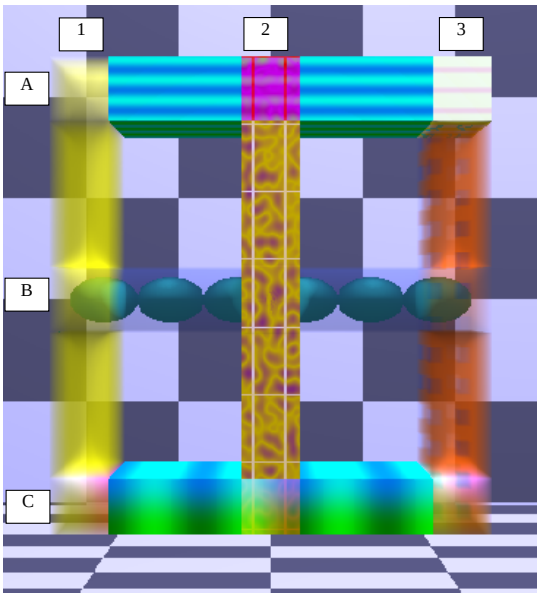


Figure 4. Examples of different union operations. Six different blocks, and nine different unions. For instance, the union A1 gives priority to the attributes of the block 1; the attributes of the union A2 are defined as the difference of the attributes of blocks A and 2, and as the sum of attributes for the union A3.

is well known that the best way for point membership classification against a 3D solid is to apply Constructive Solid Geometry (CSG).

Nevertheless, classical CSG suffers from a lack of variety due to the restricted set of available primitives and operations, especially those applicable on the level of complex solids. This limitation means that, if one wants to use classical CSG to decompose the coordinate space into complex subsets, it may be too difficult or too time consuming, or even impossible to do so. A much more powerful solution is to use FRep as shown in the following examples.

Complex space partitioning can be obtained using a constructive FRep tree, as it is shown in Fig. 2 and in Fig. 3. Fig. 1 shows the simplified constructive geometric and texturing trees for the example of Fig. 2. As one can see, the geometry is rather simple; the main used primitive is an ellipsoid and a tapering operation is applied at the top level of the tree (Fig. 1(Left)). The attributes tree is slightly more complex. Nodes of the geometrical tree (in brackets) are combined with some other primitives. For instance, a union operation is applied to the bottom ellipsoid and to four spheres, as Fig. 2(left) shows. The point set defined by the ellipsoid is divided into two sub-point sets; points that belong only to the set defined by the ellipsoid, and points that belong to both sets defined by the ellispoid and the union of spheres. Then, to each sub-point set, different shading parameters are applied.

Colors of objects in Fig 2(right) are mainly based on Perlin's solid noise. A mosaic pattern is applied to the bowl, using a set of block primitives. Each block is combined with solid noise, based on the Gardner's function with using an algebraic sum. Then, some attributes are assigned to each noisy block. Finally, an additional partition is defined as a union of the water and a fish (defined as Bézier volume), each with its own attributes. Because the space partition for texturing is defined by a constructive tree, we call this method constructive solid texturing.

The initial object in Fig. 3(top-right) is an FRep object (a model of a real sculpture "Naked" by Russian artist I. Seleznev Fig. 3(top-left)) mainly composed of unions of convolution surface primitives. The space partition is defined with union of four swept spirals (Fig. 3(bottom-left)). Dif-

ferent shading parameters are assigned to each spiral. Then, when the tree traversing procedures are applied to the construction trees of the initial object and space partition, we get both the defining function value for the sculpture and the shading values and can then use them for texturing (Fig. 3(bottom-right)).

4.2. Constructive Texturing Tree

This sub-section provides a description of some particularities of the constructive texturing tree. After a general overview, we will make some remarks on the meaning of set-theoretic and other operations illustrated by the union of two textured objects.

The constructive texturing tree has somewhat different meaning if one compares it to an FRep constructive tree. In both cases, they are used to evaluate a real-valued function by means of a tree traversing procedure. In the case of FRep solid modelling, this function defines point membership and has to be continuous. The constructive texturing tree is used in a different way. If, at a given point, the defining function of the space partitioning solid is positive (i.e., the point belongs to this solid) then, one can evaluate an attribute function by applying the tree traversing procedure to the texturing attributes in the tree. Thus, we add the operator "if" as a node in the constructive texturing tree. Furthermore, the continuity requirement for the attribute functions $S_i(X)$ is not necessary in the case of texturing, and "jumps" between colors are allowed, as one can naturally see in a checkerboard pattern, for instance.

When one builds a constructive tree to model a geometric solid, the use of set-theoretic and other operations such as blending is required, but the visual result of such operations should be clearly defined. Let us consider a simple solid, defined as the union of three horizontal blocks and three vertical blocks as shown in Fig. 4. A single definition of the union is used for the geometric FRep model [18], but several different definitions are possible for the constructive texturing tree. In Fig. 4, different textures are applied to each block (simple color for the block 1; fully opaque checker-board pattern for the block 2, with noisy colors for the blocks, and grey color for the space between them; semi-transparent checker-board pattern for the block 3; textures based on trigonometric functions for the blocks A and C; texture for the block B is defined using a union of the block and different ellipsoids, where the block's initial texture is semi-transparent, and the ellipsoids are completely opaque). Then, nine different union operations are defined for the attributes. Namely, the union A1 gives priority to the texture of the block 1, 2C is the sum of two RGBA vectors, and 3C is composed of the green and blue components from the block C texture and the opacity and the red component from the block 3 texture. Other unions are defined in a similar way, which illustrates the variety of available operations. In this example, the color attribute was presented as an RGBA vector. It is clear that the variety of possible definitions for the operation becomes larger if one considers another color space, such as HLS, XYZ, La^*b^* and others. Change from one color space to another can be meaningful if one considers, for instance, a blending union, which requires attributes interpolation for the added material. If one can deduce a formula to interpolate the color, it appears that the RGB color space does not suit this purpose. If one mixes two bright colors with equal brightness, the most natural result would be also a bright color. This result is very difficult to achieve using the classical RGB model, and the solution is to change the color model to HLS, for example. The interpolation function can then be applied only to the luminosity component.

These simple examples lead to the conclusion that a tool, which uses constructive solid texturing, has to allow users enough freedom in order to make them enable to define all the attribute functions easily. The HyperFun language satisfies this need, as it is described in the next section.

4.3. Constructive Time-dependent Texturing

There is a long-standing interest in time-dependent texturing concerned, in particular, with a concept of the "shade tree" model [7], and later used in a scene-graph based rendering environment [26]. The extension of our method to a time-dependent texture is straightforward. For example, when one creates a 4D model for animation the hypervolume model functions $(F, S_1, S_2, \ldots, S_k)$ can be expressed as $F(X)$ and $S_i(X)$ with $X = (x, y, z, t)$. The geometric and attribute constructive trees are defined using the FRep approach. The similarity with the way these trees are created is significant for our framework. It implies that each transformation that occurs in the geometry tree can also occur and is supported in the attribute tree.

For instance, consider the object in Fig. 5(a), and apply a twist and a tapering operations to it. The top and bottom parts of the model are textured using a pattern based on Perlin's noise. The problem with the standard definition of solid textures is that the space partitioning is done during a separate step rather than the modeling one, and may not support such transformations. The proposed method allows us to support these transformations by including corresponding nodes in both geometric and attribute trees. The straightforward application of solid textures leads to the result shown in Fig. 5(b), where one can see the transformed object, and a "transformation independent" pattern. The areas of constant colors are localized at the same positions, and are not stretched as they should be. In the real world, if one applies such operations, the result similar to one shown in Fig. 5(c) is expected, where the pattern follows the twist

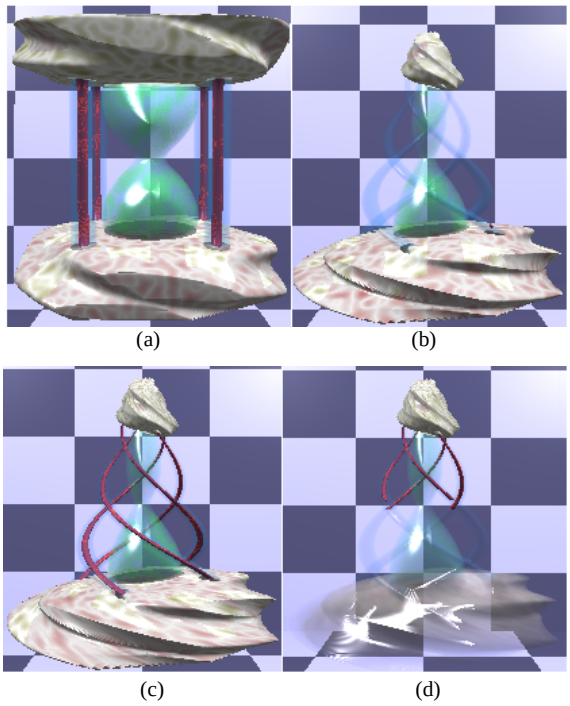


Figure 5. Time-dependent texturing. The initial model (a). Twisting and tapering applied to the geometrical object with standard solid texture (b). The same transformations applied to the object and to the constructive solid texture (c). Time-dependent space partition (d).

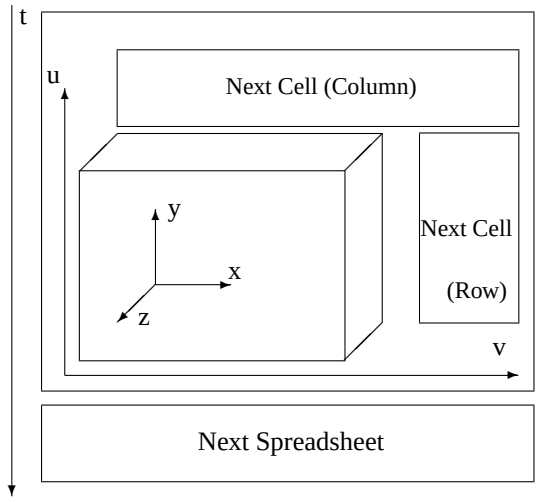


Figure 6. Animated spreadsheet concept: a spreadsheet has (u,v) coordinates. Each (u,v) pair corresponds to a cell containing a 3D object with (x,y,z) coordinates. Spreadsheet changes in time with the t coordinate.

and the tapering during the time-dependent transformation. The main advantage of constructive solid texturing is that the space partition can also change in time. As it can be seen in Fig. 5(d), the space partition for the opacity has been added, corresponding to a block with its time-dependent size along y-axis. The visual result for this model is the entire object becoming semi-transparent from bottom to top.

Depending on the desired result, transformations of the FRep object geometry can also occur in the constructive texturing tree. If they occur in both trees, the visual result is the transformation of all texture patterns together with the object geometry.

4.4. Constructive texturing in multiple dimensions

Multidimensional models are conventional in mathematics, natural sciences and data mining. Here, we illustrate an application of our approach to scientific visualization. As an example, we propose to construct a visual representation of a function of six variables $f = f(x_1, x_2, x_3, x_4, x_5, x_6)$. This function was first introduced to illustrate unstable states in plasma physics. Assigning a zero value to the function defines a star-shaped isosurface as an elementary object in the cells of the animated spreadsheet as illustrated in Fig. 7(see Fig. 6 for the concept explanation). 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 $x[4]$. Formally, the following types are assigned to the geometric coordinates:

$x[1] \rightarrow x$	$x[2] \rightarrow y$	$x[3] \rightarrow z$
$x[4] \rightarrow t$	$x[5] \rightarrow u$	$x[6] \rightarrow v$

(3)

where t is a dynamic variable corresponding to physical time, u and v are spreadsheet coordinates. Three time steps of the animated spreadsheet are shown in Fig. 7. We used this function to show that the way the constructive solid texturing method was defined was independent of the model's dimensionality. Fig. 7 shows coloring of the shapes for three different time steps. The red component is a function of $x[1]$, $x[4]$ and $x[5]$, the green one depends on $x[6]$. The space partition has been done with the use of a union of the star shape and a torus. The radius of the torus is time-dependent, and it grows in time. The transparency value has been assigned to it. The result shows that the use of a multidimensional object for constructive texturing tree becomes meaningful, and visualization of the dependence between variables becomes easier.

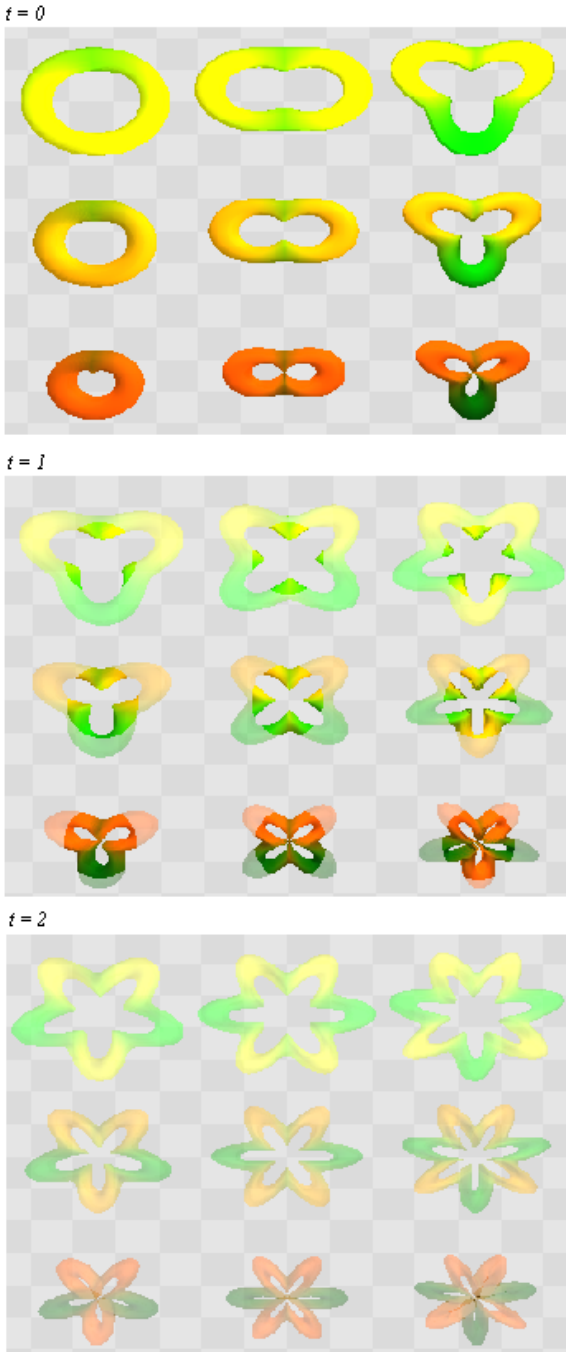


Figure 7. Animated spreadsheet of a constructive hypervolume 6D point set with 13 photometric attributes : three frames of animation for three time steps $t = 0, 1, 2$.

5. Implementation

5.1. Language for Hypervolume Modeling

HyperFun [1] has been developed as a high-level specialized language for the parameterized description of functionally based multidimensional geometric models. While being minimalist and suitable for easy mastering (our extensive experience shows that half of hour is enough for an ordinary undergraduate to work further without guidance), it supports all main notions of FRep. The current version of the language that is publicly available [27] only allows for the description of geometry. Here, we present a new version that allows us to deal with the constructive hypervolume model of any degree of generality.

A model in the HyperFun language can contain the specification of several hypervolume objects parameterized by input arrays of point coordinate $\{x_i\}$ 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 (the function's name coincides with the object's name) accompanied, if necessary, by a set of scalar functions $\{s_i\}$ representing its attributes. Note the following feature that enables us to increase flexibility while dealing with hypervolume objects with texturing-like attributes: in distinction from the "geometric" function, values of the corresponding scalar functions can either be calculated within the HyperFun program from scratch in accordance with the formulae defined in the program or can be passed from the outside to be modified (if necessary) within the object's program.

The functions defined in HyperFun are actually a symbolic embodiment of the corresponding trees whose structure reflects a constructive logic of building both the object's geometry and its attributes. Note that along with primitives (that can be library functions and local variables defined by algebraic expressions with an appropriate semantics), other objects can be tree leaves too. At the language level, this means that references to objects that have already been specified can be present in functional expressions.

There are actually no limitations on the functions' complexity that can be built in a step by step manner using assignment statements (by introducing local variables and arrays, if necessary). Conditional selection ('if-then-else') and iterative ('while-loop') structures are also available. Functional expressions are built using conventional arithmetic and relational operators by utilizing standard mathematical functions ('exp', 'log', 'sqrt', 'sin', 'cos', etc.). The distinctive feature of HyperFun is the support of fundamental set-theoretic operations by special built-in operators with the reserved symbols ('|' - union, '&' - intersection, '\ ' - subtraction, '∼' - negation, '@' -

Cartesian product).

In principle, the language is self-contained and allows users to build objects from scratch, without using any pre-defined primitives. However, its expressive power is greatly increased by the availability of the system "FRep library" that is easily extendable and can be adapted to a particular application domain and can even be customized for needs of a particular user. The current FRep library version in general use contains the most common primitives and transformations of a quite broad spectrum.

Thus, there are functions implementing conventional CSG primitives (block, sphere, cylinder, cone, torus) as well as their more general counterparts (ellipsoid, superellipsoid, elliptic cylinder, elliptic cone). Another group of the library primitives implements popular implicits (blobby object [2], soft object [32], metaballs [16]) including recently added convolution objects [14] with skeletons of different types (points, line segments, arcs, triangles, curve, and mesh). Primitives derived from parametric functions (cubic spline and Bézier objects) have also been included into the library. As to transformations, one can mention 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.

5.2. The HyperFun library functions for texturing

Taking into account that texturing often requires non-trivial mathematical skills and specialist knowledge (e.g., in color theory), we have been developing the library functions that can facilitate creating constructive hypervolume texturing models. There are two main groups of useful functions.

The first one includes functions that are applicable to different attributes irrespective to their specifics. For instance, the functions 'hfA_Gradient', 'hfA_Wave' and 'hfA_NoiseG' return a value respectively linearly interpolated within an interval, interpolated using a sine function, and a pseudo-random value from the interval [0,1] calculated using Gardner's solid noise.

The second group includes more specialized functions to generate automatically some patterns. Let us mention here the function 'hfA_Wall' which was used to produce the "noisy" checker-board of the different figures, and that is relevant in this paper context.

Then, once attribute arrays are defined, we propose to use functions defining basic operations such as set-theoretic and others. In general, these operations transfer input arrays of attribute values to an output array according to the function values.

Some other service functions are also provided, such as "hfA_HLStoRGB", which allows for the conversion from one color space to another.

Fig. 8 shows an example of a HyperFun model. The first part of the model is a geometrical definition of the object. The point set is defined as a union of a torus, a superellipsoid and a soft object. Then, when the defining function of the model is positive, the attributes tree is defined. In this example, it is very similar to the geometric one. . A number of various attributes are set to the soft object (not shown in the HyperFun example for length reason), to the superellipsoid and, finally, to the torus. For the ellipsoid, a gradient along the y-axis is set, and an additionnal smaller superellipsoid with a different color is added to the texturing tree, with a different color. For the torus, a pattern of different blue gradations is defined using Perlin's noise function. Inside the main torus, a smaller red torus is also inserted. As the bigger torus is completely opaque, two additional spheres are added to the tree with a similar color, but with a different opacity coefficient. . Thus, one can see the red torus inside. Details of each library function can be found in [17].

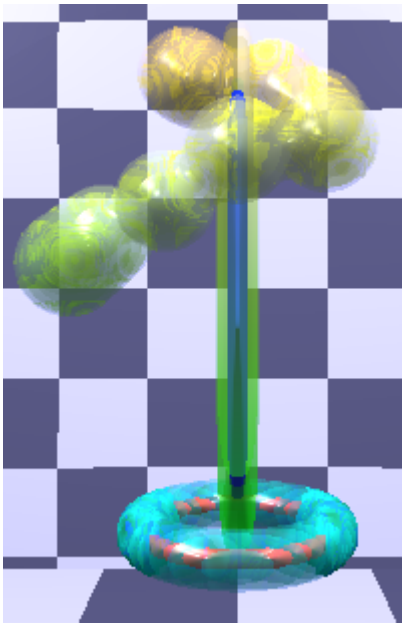
5.3. Software Tools

Application software deals with HyperFun models with help of a built-in interpreter or by using HyperFun-to-C or HyperFun-to-Java compilers and utilities of the HyperFun API. The latter way concerned with intermediate generation of C/Java code ensures more efficient function evaluation but is much more demanding for developers of application software in a multi-platform environment. In this work the former way has been used.

The HyperFun interpreter has been implemented as a small set of functions in ANCI C. It is quite easy to integrate them into the application software since the developer needs to deal with only two functions. '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 that is actually a collection of the tree structures optimized for subsequent efficient evaluation. If there are any errors in the program, the function outputs a list containing the location and details of each error found.

Another interpreter function ('Calc') called every time there is a need to evaluate the functions at a given point in the modelling space and for the given external numerical parameters. Externally defined values for scalar functions can be passed too. The object's internal representation serves as an input parameter for the function that returns both the value of the "geometric" function and a set of values for scalar functions - all evaluated at the given point.

The formal specification of the internal representation and the functions' evaluation procedure was given in [19]. Note, that the function 'Parse' is invoked just once while



```
my_model(x[3], a[1],s[4]){
--Declaration part skipped...
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];
soft  = hfBlobby(x,x0,y0,z0,d,0.2);
-- final model as set-theoretic union
model = superEll | torus | soft;
if(model>0.0) then
--Definition of the constructive texturing tree.
--SuperEllispoid
      --A smaller ellipsoid is defined inside the
      --superEllisoid
         elB = 1-(x1/0.4)^4-(x2/8)^4-(x3/0.4)^4;
         clA=[0.45,0.87,0.1,1.0]; clB=[0.1,0.2,0.8,1.0];
      --Gradient along the y-axis for the opacity
         tr =(x2+10)/20;
         clA[4] = 0.5+0.5*(1.0-tr);
       --Union operation. "cell" is an array
       --containing the
       --resulting attributes
         tmp = hfA_Union(superEll,elB,cell,clA,clB,1);
--Torus
       --Definition of the mosaic
       --checker-board patterns plus Gardner's noise ; size of the blocks and space
       --inbetween
          space = [0.15,0.15,0.15]; bricks = [4.0,2.0,2.0];
          noise = hfNoiseG(x,0.75,1,1);
          bricks[1] = 1+noise; bricks[2] = 1+noise; bricks[3] = 1+noise;
       --color of the space and of the block (Perlin's noise).
          noise = hfA_NoiseP(x,1.0);
          c_br[1] = 0.8-0.5*noise;c_br[2] = 0.8-0.5*noise;c_br[3] = 0.8-0.8*noise;
          c_sp[1]  =0.7-0.2*noise ; c_sp[2]  =0.7-0.5*noise ; c_sp[3]  =0.7+0.5*noise ;
       --Definition of the pattern. Resulting color in the ct array
          tmp = hfA_Wall(x,bricks,space,ct,c_bricks,c_space);
       --Additionnal space partition
           center = [0,-9,-3.5];
           sp1 = hfSphere(x,center,3.25);
           center = [0,-9,3.5];
           sp2 = hfSphere(x,center,3.25);
           sp = sp1 | sp2;
       --Color of the sphere is the same as for the torus,
       --only opacity has changed.
           c_sp[1]  = ct[1] ; c_sp[2]  = ct[2] ; c_sp[3]  = ct[3] ; c_sp[4]  = 0.8 ;
       --Union of the spheres and the torus.
           tmp = hfA_Union(torus,sp,ct,ct,c_sp,1);
       --Creation of a new space partition
       --A smaller red torus inside.
          c_midt=[1.0,0.2,0.2,1];
          center = [0,-9,0];
          mid_t = hfTorusY(x,center,3.5,0.35);
       --Final color for the torus.
           tmp = hfA_Union(torus,mid_t,ct,ct,c_midt,1);
--(….skip the soft object…). The result is copied to the « s » array
--Red attribute : Gradient along the y axis for all the tree
--except for the smaller superellipsoid and the smaller torus
        if( (elb | mid_t ) <0.0) then  s[1] = (x2+10)/20; endif;
endif;
my_model = model;
}
```

Figure 8. Example of a HyperFun model. The image has been produced using vlib and this HyperFun model.

processing the HyperFun program; in a way, the internal representation can be treated as "byte-code" and can serve as a protocol for data exchange between system components. In fact, these two procedures constitute an application programming interface (API) that is extremely simple for utilizing.

Software tools for HyperFun creation and processing are being developed in an open source project manner by the international team of developers. Some of them are currently available for free download at the Web site [27]: HyperFun Polygonizer for the surface mesh generation with VRML output and HyperFun plug-in to POVRay [28], which makes it possible to generate high quality photorealistic images on an ordinary PC. As the previously mentioned tools are dedicated to surface rendering, we also provide a plug-in to the volume rendering engine Vlib [29] in order to fully exploit the capability of the proposed concept. The majority of the figures produced for this paper were rendered using this engine.

A special Interactive Construction Tool has been under development within HyperFun Project that makes it possible for the user to design the constructive trees through interactive graphical interface with automatic generation of programs in HyperFun language [12]. A combination of an interactive GUI having inevitably limited design vocabulary with perhaps less intuitive but yet very rich features of a symbolic functional description (using built-in editor and interpreter) promises a new quality for creating such non-trivial models as concerned with hypervolume texturing.

Let us emphasize here the following non-trivial feature of our framework that has found its implementation in the tools. Conceptually, we strive to separate the modelling in multidimensional space with abstract coordinate variables $x_1, \ldots, x_n$ from subsequent interpretation of the model in "real" terms (that can be, in particular, a visualization). The concept of multimedia types [1] is exploited here. A special mapping with giving each coordinate an interpretation has been proposed (for instance, 'x', 'y', 'z' types can correspond to Cartesian coordinates; 't' - to "dynamic" coordinate representing continuous values that can be linearly or no-linearly mapped onto physical time; 'u' and 'v' - to 2D "spreadsheet" coordinates etc. - the example has already been given in Subsection 4.4). It is important that HyperFun tools have special features allowing users to fulfil this mapping procedure.

With introducing a set of scalar functions for representing object attributes, one can propose a similar methodology. This means that within a HyperFun program, the object's attributes are considered as abstract real-valued functions; as to their actual meaning, it can be determined later - by an appropriate application program. There can obviously be models that could greatly benefit from such a technology that actually allows us to introduce "generic" objects with subsequent generation of their different instances. For example, the same attribute can be treated (without change of the program in HyperFun!) as color, or as transparency, or as density, or as temperature, depending on circumstances and available application software features. Moreover, it is possible to assign simultaneously a few multimedia types to the same attribute. However, if the user considers it appropriate, it is possible to fix the attribute's meaning as early as on the modelling stage. If this is the case (as with the examples in this paper), the library of specialized functions introduced in the previous Subsection can be especially useful.

5.4. Examples

As we mentioned before, the proposed approach is applicable to different geometric models. Fig. 9 illustrates the application of the constructive solid texturing to an Frep object (Fig. 9(a)), a polygonized Frep object (Fig. 9(b)), and a standard BRep object (the Stanford Bunny, Fig. 9(c)). The constructive solid texturing tree for the object of Fig. 9(a) was built together with the constructive FRep tree and used the same subtrees in some parts. The original constructive FRep tree helped to define the space partition for the mesh coloring in Fig. 9(b). The most difficult task was to build a constructive solid texturing tree for the BRep model in Fig. 9(c). Some special interactive tools are needed to support visualization of a BRep model overlapping with the visual representation of the constructive tree.

6. Conclusion

In this paper, we used a general hypervolume model as a framework. A hypervolume is considered as a multidimensional point set with multiple photometric attributes (color, transparency, diffuse and specular reflections, etc.) and other physical attributes (density, temperature, etc.). The point set geometry and attributes are modeled independently with using real-valued scalar functions of several variables. Each real-valued function defining geometry or an attribute is evaluated at the given point by a procedure traversing a constructive tree structure with primitives in the leaves and operations in the nodes of the tree. By applying this general model to texturing, we extended the well-known concept of solid texturing in two directions: constructive modeling of space partitions for texturing and modeling of multidimensional textured objects. We discussed some operations specific for constructive solid texturing. The proposed approach allows for modeling, texturing and visualization of 3D solids, time-dependent and multidimensional objects in a completely uniform manner. We introduced an extension of a special modeling language

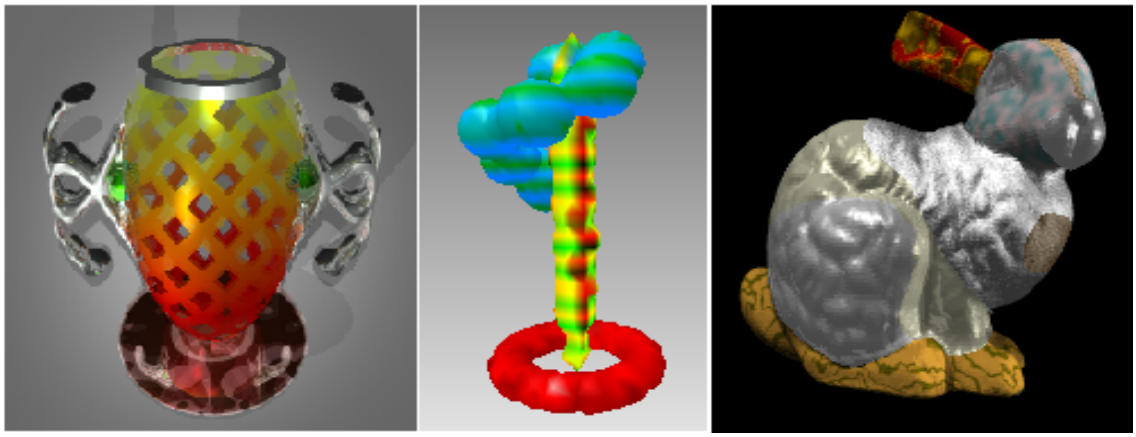


Figure 9. Application of the constructive solid texturing to (from left to right) an FRep object, a polygonized Frep object, and a BRep object (the Stanford Bunny).

called HyperFun and implemented software tools supporting the proposed approach. The tools will soon be available for downloading from HyperFun Project Home Page [27].

The concept of constructive hypervolume textures is independent of the geometry representation. We provided examples of textured Frep and BRep objects as illustrations. The hypervolume model can also accommodate 3D and higher dimensional voxel arrays to represent geometry or attributes of different (not only photometric!) nature using appropriate interpolation procedures. Incorporating voxel arrays and experiments with them, applications of volume rendering as well as multiple-material rapid prototyping of modelled objects will be the subjects of our future research.

7 Acknowledgments

The authors sincerely thank Professors Peter Comninos and Jian J. Zhang (NCCA, Bournemouth University, UK) for their support of this work. Peter Comninos has also greatly helped us with stylistic problems of English text. Our special thanks to Moscow artist Igor Seleznev for his kind permission to use his sculpture image and to Alexander Ogarko (student, Moscow Engineering Physics Institute, Russia) for his work on sculpture modeling. We also thank anonymous referees for their useful remarks.

References

[1] V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, and V. Savchenko. HyperFun project: a framework for collaborative multidimensional FRep modeling. *Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop*, pages 59–69, September 1999.

[2] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):135–256, 1982.

[3] J. Blinn and M. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, October 1976.

[4] J. Bloomenthal and al. *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

[5] E. Catmull. *A subdivision algorithm for Computer Display of Curved Surfaces*. PhD thesis, Department of Computer Science University of Utah, December 1974.

[6] M. Chen and J. Tucker. Constructive volume geometry. *Computer Graphics Forum*, 19(4):281–293, 2000.

[7] R. Cook. Shade trees. *Computer Graphics*, 18(3):223–231, 1984.

[8] D. Ebert and al. *Texturing and Modeling : A practical approach*. A.P. Professional USA, ISBN 0-12-228760-6, 1996.

[9] A. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann USA ISBN 1-55860-276-3, 1995.

[10] A. Gomes, A. Middleditch, and C. Reade. A mathematical model for boundary representations of n-dimensional geometric objects. *Fifth Symposium on Solid Modeling and Applications*, pages 270–277, September 1999.

[11] S. Haruyama and B. Barsky. Using stochastic modeling for texture generation. *IEEE Computer Graphics and Applications*, 4(3):7–19, March 1984.

[12] T. Hibi and A. Pasko. Graphical interface for design of geometric data structures. *Lecture Notes in Computer Science*, 1966:134–147, 2000.

[13] V. Kumar, D. Burns, D. Dutta, and C. Hoffmann. A framework for object modeling. *Computer-Aided Design*, 31(9):541–556, 1999.

[14] J. McCormack and A. Sherstyuk. Creating and rendering convolution surfaces. *Computer Graphics Forum*, 17(2):113–120, 1998.

[15] J. Menon, R. Marisa, and J. Zagajac. More powerful solid modeling through ray representations. *IEEE Computer Graphics and Applications*, 14(3):22–35, 1994.

[16] H. Nishimura., M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modeling by distributed function and a method of image generation (in Japanese). *Transactions of IECE of Japan*, J68-D(4):718–725, 1985.

[17] A. Pasko, V. Adzhiev, and B. Schmitt. Constructive hypervolume modelling. Technical Report TR-NCCA-2001-01, ISBN 1-85899-123-4, National Centre for Computer Animation, Bournemouth University, UK, 2001.

[18] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995.

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

[20] D. Peachey. Solid texturing of complex surfaces. *SIGGRAPH Proc.'85, USA*, 19(3):279–286, 1985.

[21] K. Perlin. An image synthesizer. *Computer Graphics, SIGGRAPH Proc.'85*, 19(3):287–296, 1985.

[22] A. Requicha. Representations for rigid solids: theory, methods, and systems. *ACM Computing Surveys*, 12(4):437–464, 1980.

[23] J. Rossignac. Through the cracks of the solid modeling milestone, From Geometric Modeling to Advanced Visual Communications. *S.Coquillart, W. Strasser, P. Stucki, Springer-Verlag*, pages 1–75, 1994.

[24] V. Savchenko and A. Pasko. Transformation of functionally defined shapes by extended space mappings. *The Visual Computer*, 14(5/6):257–270, 1998.

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

[26] S. Uptill. *The Renderman Companion*. Addison-Wesley, 1990.

[27] URL. *The HyperFun Project : Language and Software for FRep Modeling*. http://www.hyperfun.org.

[28] URL. *The Persistence of Vision Raytracer Homepage*. http://www.povray.org.

[29] A. Winter and M. Chen. vlib : A volume graphic api. *Proceedings of the 2^{nd} international workshop on Volume Graphics*, pages 359–376, June 2001.

[30] K. Wise and A. Bowyer. Using CSG models in many dimensions to map where things can and cannot go. *CSG 96 Set-theoretic Solid Modelling*, pages 359–376, 1996.

[31] B. Wyvill, E. Galin, and A. Guy. Extending the CSG tree. warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18(2):149–158, 1999.

[32] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.