Constructive Hypervolume Textures

B. Schmitt[α], A. Pasko[β], V. Adzhiev[γ] and C. Schlick[α]

[α] LaBRI , University of Bordeaux, France.
[β] Faculty of Computer and Information Sciences, Hosei University, Japan.
[γ] National Centre for Computer Animation, Bournemouth University, UK.

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 in 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. This allows the graphics system user to obtain more realistic images. Several existing methods of texturing are dependent on the representation of the object one wants to render. Some methods may be applied to polygonized objects, others to implicit surfaces. Such methods are briefly described in the second section of this paper. A method, called solid texturing [19, 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 its attributes can be completely independent. Section 5 describes briefly the extension of the HyperFun language [14] for supporting constructive hypervolume modeling, corresponding software tools and obtained results.

2. Existing Texturing Techniques

Several texturing techniques exist in computer graphics to embellish geometric objects. In the following, we will use the texture definition given in [11], where the term texture is defined as "anything that is evaluated at a point using only information local to that point is a texture". 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 [13]. In the book [9], a complete description of different methods of making 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 called 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 [6] was the first solution proposed allowing to apply some 2D color patterns (digital images or procedural function) to a parametric surface. Many other works in this area followed and extended the application of a color to some other shading parameters [3], or to the modification of a surface properties such as the normal vector perturbations, i.e., bump mapping [4]. Various texture mapping techniques exist, like spherical, cylindrical, or gaussian mapping, and can usually be described as mapping $\Re^3 \to \Re^2$. Surfaces to be textured was first restricted to parametric one, and then extended to implicit surfaces [20, 27], to skeleton-based models [28], both with a special parameterization process, or even a completely different approach, such as the one in [22], for instance, where the idea is to cover an arbitrary surface using overlapping copies of a texture patch, and to define some local parameterizations ("lapped texture").

The previously exposed texture mapping is widely used, but this solution encounters some difficulties due to the requirement of parameterization of the considered surface. Other texturing methods exist, like cellular texturing [10], which is applicable to any kind of surfaces, but is restricted to a certain kind of texture.

An alternative is the concept of solid texturing initially introduced in [19, 21]. The method defines procedurally a texture pattern in the object 3D-space, with the help of functions called solid texture functions. Given a point (x,y,z) on the surface of the 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 identity mapping. The main advantage of this method is that it does not require anymore extra complex steps for parameterization, and solves many problems existing with the previously exposed methods, such as continuity of the texture between patches, for instance. Another valuable property of this method is that it can be applied to any kind of surfaces, and, in particular, to more arbitrarily complex ones. Indeed, solid texturing can be thought as the definition of space where a procedural texture is defined everywhere. But the way to define the space partition is arbitrary, and does not rely on a solid framework. There is another method to define a space partition with a more robust structure, but only applicable to implicit surfaces. BlobTree [31] is a hierarchical tree structure with 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 values specified by this node will be the default attributes for nodes lower in the hierarchy. Another attribute node deeper in the down this tree will override the more shallow one. Such a scheme supposes a fixed discipline of assigning attributes to the entire implicit surface rather than particular space points.

In this paper, 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 in 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 in any of its points.

In general, the hypervolume model can be expressed as :

$$(G, a_1, a_2, \ldots, a_k) \qquad (1)$$

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 [23, 25, 24, 16, 18]. While some of the traditional representations such as CSG or BRep have multidimensional extensions [30, 12], 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 model of [15], 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) [8], 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 [8]. 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 [2] generate scalar fields. Such 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:

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

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 has 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 in 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, 26].

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 in 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, on the basis of the introduced general model we set the framework for the constructive hypervolume texturing. Using the solid texturing defined in [19, 21], we propose first 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, 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

As we mentioned earlier, we consider hypervolumes, defined as $(F, s_1, s_2, \dots, s_k)$. In this subsection, we will deal only with 3D objects, particularly implicit surfaces and FRep solids. In 3D space, our approach can be considered as extension of the solid textures concept. When one applies solid texturing to an object, he/she has to create a space partition

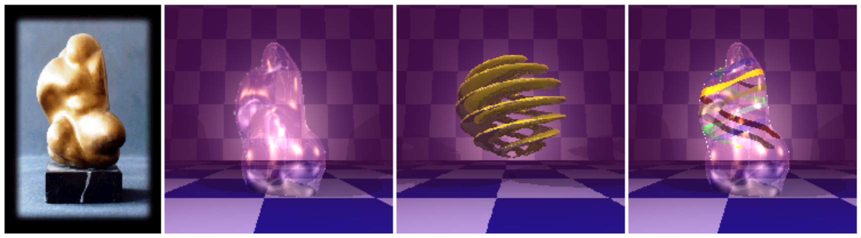

Figure 1: *Constructive solid texturing: From left to right : Original model; original static 3D shape; constructive 3D solid representing the space partitioning for texturing; textured shape.*

of the object space, where each subset contains different material property. Then, the shading parameters are computed for each point in space, where its position determines which subset it belongs to, and eventually the corresponding attributes. One difficulty is how to define such subsets. Actually, a subset can be thought as a solid object and has to provide for each point an answer to the question: "Is it inside or outside the subset?" In the affirmative case, the corresponding solid texture function is applied. It is well known that the best way for point membership classification against a 3D solid is to apply a Constructive Solid Geometry (CSG) tree. Nevertheless, CSG suffers from a lack of variety due to the restricted set of available primitives and operations. It means, if one wants to decompose the coordinate space into complex subsets, it may be too difficult or too time consuming, or even impossible. A much more powerful solution is to use FRep as it will be shown in the following.

Complex space partitioning can be obtained using a constructive FRep tree, as it is shown in Fig. 1 and in Fig. 2. The initial object in Fig. 1(left) is an FRep object (a model of a real sculpture "Naked" by Russian artist I. Seleznev) mainly composed of unions between convolution surfaces. The space partition has been defined with union of three swept spirals (Fig. 1(second picture)). Different 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 obtain both defining function value for the sculpture and shading values. An another non-trivial space partition is shown in Fig. 2. A simple block is considered as the object to be textured. The space partition has been defined with three ellipsoids and default RGB color values outside of them, corresponding to a gradient of the red and green components along the x and y axes. The ellipsoids have been deformed using a non-linear space mapping. The bottom one contains a noise red pattern, the middle one - a checker pattern, and the top one - a sinusoidal blue pattern. Note that for the last one, the blue lines follow the deformation of the ellipsoid. To obtain such space partition and texturing with traditional methods can be quite difficult and really time consuming. Since the space partition has been defined with a constructive tree, we call this method constructive solid texturing.

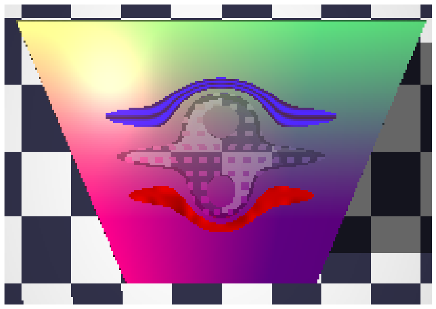

Figure 2: *Non trivial space partitioning. Three ellipsoids deformed by non-linear space mapping. Notice the blue pattern, a simple sinusoid, follows the deformation.*

4.2. Constructive Texturing Tree

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

The constructive solid texturing tree has somewhat different meaning if one compares it to a construction FRep tree. In both cases, they are equivalent to a real- valued function obtained by a tree traversing procedure. In the case of Frep solid modeling, this function defines point membership and has to be continuous. The constructive solid texturing tree is used in a different way. If, for the 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 texture 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

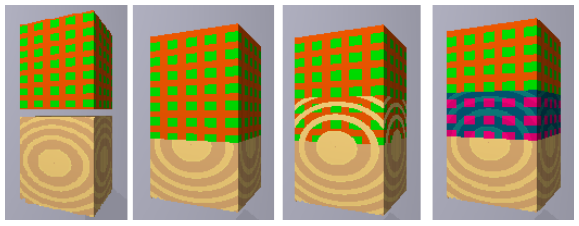

Figure 3: *Examples of different union operations. Top left shows disconnected blocks. The first union operation (top-right) gives the priority to one pattern, the second (bottom-left) shows min/max operations on the color components and (bottom-right) some user defined operation.*

consider a very simple solid, defined as the union between two blocks. The corresponding tree will be composed of a union operation between two block primitives. The resulting object is easy to imagine. But some problems arise if one assigns different colors or texture patterns to each primitive. For instance, if one of the blocks is textured with a red and green brick pattern, and the other pattern is made of concentric circles with two different browns, the color of the union can be set in many different ways.

Fig. 3 shows some results of such union. A single definition of the union is used for the geometric FRep model [18], but several definitions are possible for the constructive texture tree. In Fig. 3, different operations were applied as examples to illustrate the variety of available operations. In this example, the color attribute was composed of an RGB vector. It clearly appears that the multitude of different possibilities becomes infinite if one considers another color space, such as HLS, XYZ, La*b* and others. To change the color space is meaningful if one considers, for instance, a blending union, which requires attributes interpolation along the added material. Even if one can deduce a formula to interpolate the color, it clearly appears that the RGB color space does not suit this purpose. If one mixes two bright colors with an 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 enough freedom in order to be able to define all the attribute functions easily. The HyperFun language answers this need, as it is exposed 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 "shade tree" model [7], and later used in some scene- graph based rendering environment [29]. The extension to a time dependent texture of our method is straightforward, and includes this feature. When one creates a 4D model, i.e., an animation for instance, the functions of a hypervolume model $(F, s_1, s_2, \ldots, s_k)$ can be expressed as $F(X)$ and $s_i(X)$ with $X = (x, y, z, t)$. The geometric and attribute trees are defined using the FRep approach. The similitude in the way these trees are created is the framework for many possible extensions. It implies that each transformation that occurs in the first tree can also occur and is supported in the attributes tree. For instance, consider the cube with a 3D color grid pattern of Fig. 4(top-left), and apply a twist to it. The problem with the definition of solid textures is that the space partitioning is done during a separate step than the modeling one, and does not support such transformation. The straight application of solid textures leads to the result shown in Fig. 4(top-right), where one can see the twisted cube, and a "transformation independent" grid pattern. In the real world, if one twists the grid patterned cube, he/she expects a result similar shown in Fig. 4(bottom-left), where the pattern follows the twist during the time-dependent transformation. Fig. 4(bottom-right) shows another frame of the animation, where another transformation, namely tapering, is applied. The angle of the twist is increasing in time until the given limit is reached, then tapering is applied, where the scale at the base of the block is time dependent. While this operation is applied, the grid texture and the space partition change. As it can be seen in Fig. 4(bottom-right), some of the grid stripes become yellow according to a sinusoid function with a time dependant pe-

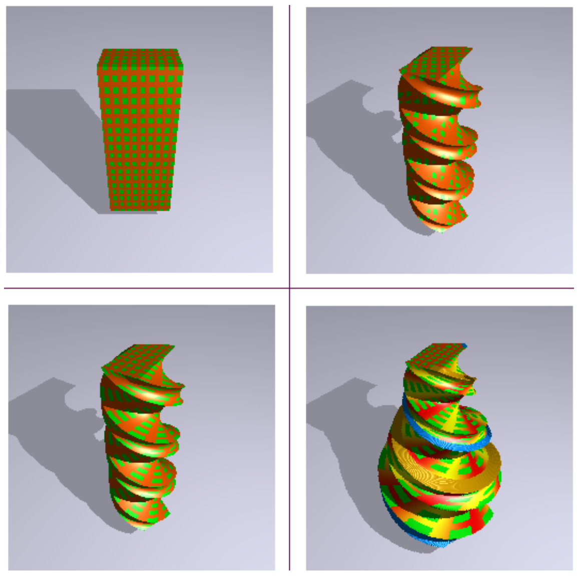

Figure 4: *Time dependant texture. The original patterned cube (top-left). A twist applied onto the geometrical object, and a solid texture (top-right). (bottom-left) The same twist with a constructive solid texture. (bottom-right) Time-dependent twisting, tapering and texture pattern.*

riodicity. The main advantage of constructive solid texturing is that the space partition also changes in time. As it can be seen in Fig. 4(bottom-right), two blocks were added only to the constructive solid texture tree at the corners of the original block, one with a blue color pattern, and the other with dark yellow. Transformations of the FRep object geometry also occur in the constructive solid texture tree, and the visual result is the transformation of all texture patterns.

4.4. Constructive texturing in multiple dimensions

Multidimensional models are conventional in mathematics, natural sciences and data mining. Here, we illustrate 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 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. 5. 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:

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

(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. 6. We used this function as a basis to show that the way the constructive solid texturing method was defined is independent of the dimension of the model. Fig. 6 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 depends on x[6]. The space partition has been done with the use of a union between the star shape and a torus. The radius of the torus is time-dependent, and it grows in time. A 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.

5. Implementation

5.1. Language for Hypervolume Modeling

HyperFun [1] has been developed as a high-level specialized language for parameterized description of functionally based multidimensional geometric models. While being minimalist and suitable for easy mastering , it supports all main notions of FRep. The version of the language that is open now [14] allows for only defining a geometric shape. Here, we briefly present a new version that makes it possible to deal with the constructive hypervolume model of any degree of generality.

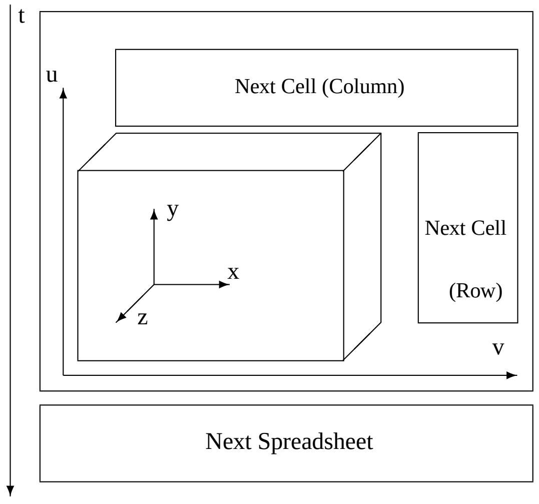

Figure 5: *Animated spreadsheet concept: a spreadsheet has (u, v) coordinates. Each (u, v) pair corresponds to a cell containing a 3D object with (x, y, z) coordinates. Spreadsheet changes in time with the t coordinate.*

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, where the function's name coincides with the object's name.

The extension we propose, (and which has been implemented), is the adjunction of a set of scalar functions $\{s_i\}$ representing attributes. The use of a such array is not a requirement, and may be used only if necessary. In this paper, we used this array to define the shading parameters of the considered object. The corresponding scalar functions were defined inside the HyperFun model, at the same time as the "geometric function". An another possible way is the s_i attributes can be passed from the outside within the object's program to be (if necessary) modified. The separation between the "geometric" function and the "scalar" functions increase the flexibility while dealing with hypervolume objects.

Within a HyperFun program, the object's attributes s_i are considered as abstract real-valued functions. 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

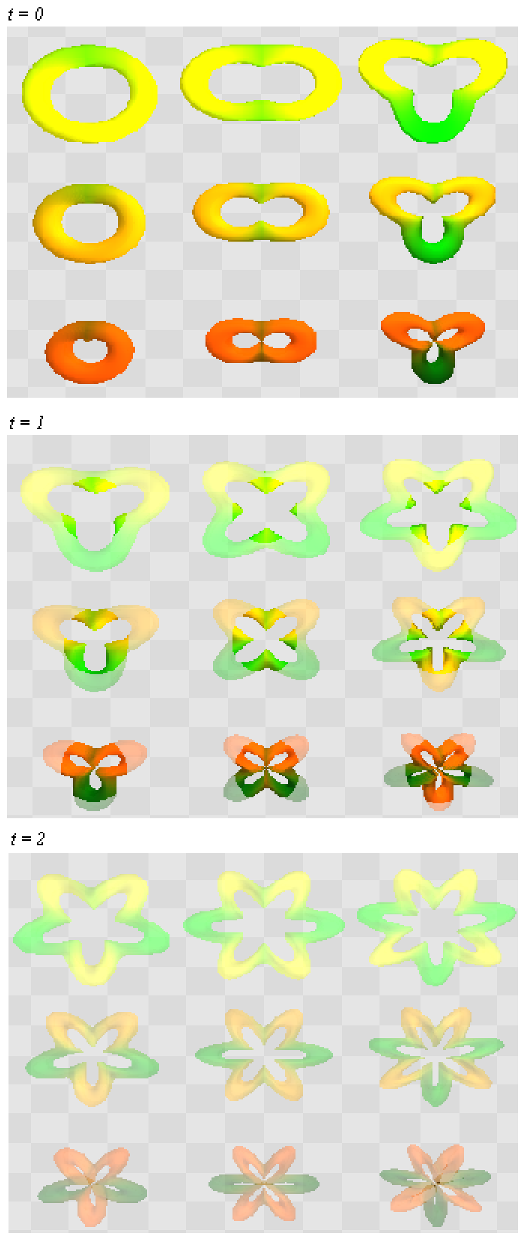

Figure 6: *Animated spreadsheet of a constructive hypervolume 6D point set with 13 photometric attributes : three frames of animation for three time steps* $t = 0,1,2$.

and available application software features. However, if the user considers it appropriate, it is possible to fix the attribute's meaning as early as on the modeling stage. If it is the case (as in this paper's examples), the library of specialised functions can be useful.

In principle, the language is self-contained and allows users to build objects from the scratch, without using any pre-defined primitives. However, its expressive power is greatly increased by 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.

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. For examples, a function which is able to return a pseudo-random value, or functions who generate automatically a space partition (similar to a checker-board pattern for instance), and also functions defining operations such as set-theoretic, blending and others.

5.2. Examples

As we mentioned before, the proposed approach is applicable to different geometric models. Fig. 7 illustrates the application, respectively from left to right,of the constructive solid texturing to an Frep object, a polygonized Frep object, and a standard BRep object (the Stanford Bunny). The constructive solid texturing tree for the object of Fig. 11a 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. 7(middle). The most difficult task was to build a constructive solid texturing tree for a BRep model in Fig. 7(right). 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 using real-valued scalar functions of several variables. Each real-valued function defining geometry or an attribute is evaluated in 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 special modeling language called HyperFun and implemented software

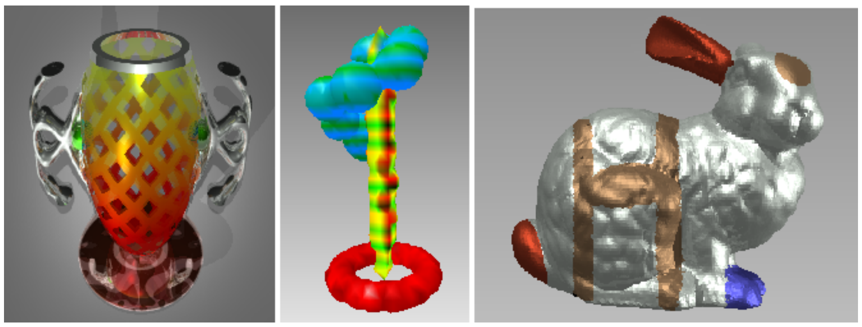

Figure 7: *Application of the constructive solid texturing to (from left to right) an FRep object, a polygonized Frep object, and a BRep object (the Stanford Bunny).*

tools supporting the proposed approach. The tools will soon be available for downloading from HyperFun Project Home Page [14].

The concept of constructive hypervolume textures is independent of the geometry representation. We provide 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 and experiments with voxel arrays, applications of volume rendering as well as multiple-material rapid prototyping of modelled objects will be the subjects of our future research.

References

1. V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, V. Savchenko, HyperFun project: a framework for collaborative multidimensional FRep modeling , *Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop*, Université de Bordeaux 1, France, September 13-15 1999), J. Hughes and C. Schlick (Eds.), pp. 59-69.
2. J. Bloomenthal et al., Introduction to Implicit Surfaces, *Morgan Kaufmann*, 1997.
3. J.F. Blinn, M.E. Newell. Texture and reflection in computer generated images. *Commun. ACM* **19**:(10)542-547, October 1976.
4. J.F. Blinn. Simulation of wrinkled surfaces. *Computer Graphics. SIGGRAPH'82 Proc.*, **16**:21-29, 1982.
5. J. Blinn, A generalization of algebraic surface drawing, *ACM Transactions on Graphics*, **1**:(3)135-256, 1982.
6. E.E. Catmull. A subdivision algorithm for Computer Display of Curved Surfaces. *PhD thesis, Department of Computer Science, University of Utah*, December 1974.
7. R.L. Cook. Shade trees. *Computer Graphics*. **18**:(3)223-231, 1984.
8. M. Chen, J. Tucker, Constructive Volume Geometry, *Computer Graphics Forum*, **19**:(4):281-293.
9. D.E. Ebert et al. Texturing and Modeling : A practical approach. ISBN 0-12-228760-6. *A.P. Professional*, USA, 1996.
10. K. W. Fleischer, D.H. Laidlaw, B.L. Currin, A.H. Barr. Cellular Texture Generation. *Computer Graphics Proc., Annual Conference Series, ACM SIGGRAPH. ACM Press*, pp. 239-248, 1995.
11. A.S. Glassner. Principles of Digital Image Synthesis. ISBN 1-55860-276-3 *Morgan Kauffmann Publishers*, USA, 1995.
12. A. Gomes, A. Middleditch, C. Reade, A mathematical model for boundary representations of n-dimensional geometric objects, *Fifth Symposium on Solid Modeling and Applications, W. Bronsvoort and D. Anderson (Eds.), ACM Press*, pp. 270-277, 1999.
13. S. Haruyama, B.A. Barsky. Using stochastic modeling for texture generation. *IEEE Computer Graphics and Applications*, **4**:(3)7-19, March 1984.
14. HyperFun Project: Language and Software for FRep Modeling, URL:http://www.hyperfun.org
15. V. Kumar, D. Burns, D. Dutta, C. Hoffmann, A framework for object modeling, *Computer-Aided Design*, **31**(9):541-556, 1999.
16. J.P. Menon, R. Marisa, J. Zagajac, More powerful solid modeling through ray representations, *IEEE Computer Graphics and Applications*, **14**(3):22-35, 1994.
17. Pasko A., Adzhiev V., Schmitt B., Constructive Hypervolume Modelling, Technical Report TR-NCCA-2001-01, *National Centre for Computer Animation*, Bournemouth University, UK, 2001, , ISBN 1-85899-123-4..
18. A. Pasko, V. Adzhiev, A. Sourin, V. Savchenko, Function representation in geometric modeling: concepts,

implementation and applications, *The Visual Computer*, **11**(8):429-446, 1995.

19. D.R. Peachey. Solid Texturing of Complex Surfaces. *SIGGRAPH Proc.'85*, **19**(3):279-286, USA, 1985.

20. H. Pedersen. Decorating Implicit Surfaces, *Proc. of SIGGRAPH 95*, Computer Graphics Proc., Annual Conference Series, pp. 291-300.

21. K. Perlin. An Image Synthesizer. *Computer Graphics, SIGGRAPH Proc.'85*, **19**(3):287-296, USA, 1985.

22. E. Praun, A. Finkelstein, H. Hoppe. Lapped Textures. *SIGGRAPH 2000*, pp465-470, USA.

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

24. J. Rossignac, Through the cracks of the solid modeling milestone, From Geometric Modeling to Advanced Visual Communications, *S.Coquillart, W. Strasser, P. Stucki (Eds.)*, Springer-Verlag, 1994, pp. 1-75.

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

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

27. J.-P. Smets-Solanes. Vector Field Based Texture Mapping of animated Implicit Objects, *Computer Graphics Forum*, **15**(3):289-300, 19960.

28. M. Tigges, B. Wyvill, A Field Interpolated Texture Mapping Algorithm for Skeletal Implicit Surfaces, *Computer Graphics International '99, IEEE Computer Society*, 1999, pp. 25-33.

29. Steve Uptill, "The Renderman Companion", *Addison-Wesley*, 1990.

30. K. Wise, A. Bowyer, Using CSG models in many dimensions to map where things can and cannot go, CSG 96 Set-theoretic Solid Modelling: *Techniques and Applications*, Information Geometers, UK, 1996, pp. 359-376.

31. B. Wyvill and 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.