N° d'ordre : **2610**

THÈSE

présentée à

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET INFORMATIQUE

par **SCHMITT Benjamin**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : **Informatique**

****

Modélisation d'hypervolumes constructifs
Constructive Hypervolume Modelling

****

Soutenue le : 13 Décembre 2002

Après avis de :

Mme	M.P. Cani, Professeur, (INPG Grenoble, France)	**Rapporteurs**
M.	B. Juettler, Professeur (Université J. Kepler, Linz, Autriche)	

Devant la commission d'examen formée de :

MM.	**Pascal Guitton**	Professeur	**Président**
M.	**Christophe Schlick**	Professeur	**Rapporteur**
MMe	**Marie-Paule Cani**	Professeur	**Examinateurs**
MM.	**Laurent Grisoni**	Maître de Conférences	
	Bert Juettler	Professeur	
	Alexander Pasko	Professeur	

- 2002 -

Contents

Introduction

This document deals with modelling point sets with attributes. We consider point sets in geometric spaces (affine, Euclidean, etc.) of arbitrary dimension. A point set is a geometric model of a real or abstract object under consideration. An attribute can be defined as a mathematical model of an object property of arbitrary nature defined at any point of the point set. For example, to model a mechanical part with varying internal material distribution one can introduce a three-dimensional solid as a point set and a real-valued scalar function to represent material density as an attribute. Application areas of such models include:

- Fabrication of objects with multiple materials and varying material distribution [KD97, KBDH99];
- Physics based simulations for the analysis of physical fields distribution over the given geometric areas [Nie00];
- Analysis of geological structures [Hou94];
- Medical examination and surgery simulation using computer tomography and other scanning devices [HFP90];
- Analysis of molecules configuration [BPRS98];
- Computer graphics and visualisation of objects with varying colours and other optical characteristics, amorphous and gaseous phenomena [KCY93].

In general, multidimensional point sets with an arbitrary number of attributes of different mathematical nature (scalar, vector, tensor, etc.) can be introduced in various ways depending on the application. Modelling techniques related to point sets with attributes span such areas as solid modelling, heterogeneous objects modelling, scalar fields or implicit surface modelling, and volume graphics. A brief survey of different modelling techniques related to point sets with attributes is provided in the first chapter. Then, on the basis of this survey we formulate requirements to a general model of hypervolumes (multidimensional point sets with multiple attributes).

In the second chapter, a new model for modelling a point set with attributes, called *constructive hypervolume*, is proposed. A Point set and its attributes are represented independently by real-valued functions model using vector functions, on the base of the *function representation* model. Each function can be associated with a tree structure and is evaluated by a tree

traversing procedure. This reflects the constructive nature of the symmetric approach to modelling geometry and the associated attributes. This model provides a rich system of primitives, operations and relations for modelling both geometry and attributes. Application examples are given, related to various areas such as heterogeneous object modelling, geological analysis and physically based simulation. Special software tools have been developed, and we introduce the HyperFun [Pro] language and its extension to the constructive hypervolume model at the end of this second chapter.

Special attention is paid in the third chapter to the problem of texturing objects. An object is considered as the point set and attributes are its photometric properties. The concept of solid texturing [Pea85, Per85] is extended in two directions: constructive modelling of space partition for texturing and modelling of multidimensional textured objects. This approach provides a framework for modelling, texturing and visualisation of 3D solids, time-dependent and multidimensional objects in a uniform manner. Furthermore, the proposed approach for texturing is independent of the geometry representation. We provide examples of textured FRep, BRep and voxel objects.

In the fourth chapter, we propose to define a volume sculpting scheme to model either an object geometry or a space partition for an attribute. A trivariate B-spline function is used to define a 3D object. While the first three coordinates are used to represent the spatial component of the object to be sculpted, the fourth coordinate is used as a scalar, which corresponds to a function value or a volume density. Thus, the shape can be manipulated by changing the scalar control coefficients of the B-spline function. To control the behaviour of the B-spline function outside its domain of definition and to provide the distance property, we apply a so-called functional clipping. The obtained distance property, combined with the properties of the B-spline object, allows us to use the resulting 3D solid as a leaf of a constructive modelling tree and to apply to it operations defined in the FRep model. To facilitate the modelling process, a multiresolution capability based on the wavelet transformation is also proposed. An interactive modeller is then described. It combines both sculpting and constructive approaches. At any time of the modelling process, one can either sculpt a part of an object while using a B-spline primitive or add another part while combining its current model to some other primitive with the use of set-theoretic. Visualisation of the polygonized object surface is done in real time.

The last chapter considers deformations in the proposed constructive hypervolume model. Given an existing shape, a large variety of techniques exist to deform an object. Most of them lead to the separation of the construction step from the deformation step while modelling. A valuable approach would be to be able to model an object regardless of the technique being used, i.e., to use either the constructive scheme or a deformation scheme in any order. To offer this possibility, we propose to define point and shape driven deformations as a special node in the constructive hypervolume model. The mathematical background for such definition is the inverse mapping. Additional deformations can be achieved by moving arbitrary points in the coordinate space and applying space mapping at any level of the constructive tree. The final constructive object is defined by a single real-valued function evaluated by the tree traversing procedure. This point-to-point based deformation is extended, and a framework for a general technique is proposed to deform an object. One important feature is that this technique allows one to change the topology of the initial object easily. From the pure geometrical point of view, when an object is deformed using the proposed framework, similar visual results can be obtained with the only

use of the constructive approach. However, in the constructive hypervolume model, not only geometry is considered, but also attributes. In this chapter, we chose photometric attributes to illustrate new features brought by the proposed approach.

Chapter 1

Modelling point sets and their attributes

1.1 Introduction

Research in modelling point sets with attributes has been developing in several interrelated directions. From the general point of view, a point set can be thought as the geometric model being used to define a given object. Without loss of generality, we use the term "attributes" to point out different properties of an object, usually defined as a mathematical model.

In this chapter, we present a brief survey of some of the existing techniques. In the literature, plenty of models cover the definition of a point set with attributes. To be convinced, one can consider a simple textured surface and notice that it is a point set with attributes. In several cases, attributes are mapped on the geometry of the point set. In the case of photometric attributes, this is called *texture mapping*. In this survey, we will rather consider point sets with attributes as a whole, i.e., attributes are defined for the point set without the use of an external mapping.

To present the different models, we choose to follow a classification proposed by Shapiro in [Sha01], where two basic representation schemes are used as main guidelines, i.e., the enumerative and combinatorial approaches, and the functional and constructive approaches. Other representations either fall in one of these classes or are combinations of them. In the latter case, such representation is called *hybrid*.

The enumerative and combinatorial approaches are widely used for representing an object. An object is defined as a set of simple entities, called *cells*. Most popular cells are points, curves, segments, triangular surface patches, cubic and tetrahedral elements. The first section presents different ways to combine cells together, by enumeration using a mapping rule, by grouping them together and by building cell complexes.

The functional approach uses real-valued functions to define a point set. Usually, a predicate is used to define the point set, and can be expressed, for a given function F, and for every given point p as $F(X) \geq 0$. This approach covers the whole area of *implicit modelling*. In this chapter, special attention is paid to this approach, and the FRep model, standing for *function representation*, is presented in details. Indeed, for some reasons that will be explained in the

next chapters, the FRep model has very powerful features that naturally lead us to use it as framework for our proposition of a new hypervolume model, i.e., a multidimensional point set with multiple attributes.

The last section is dedicated to hybrid representations, which are a combination of different representations. Several models fall in this category, in particular the *boundary representation*. Other interesting models that will be discussed are the Constructive Volume Geometry representation [CT98], object model and the general object model [KBDH99].

At the end of every section or subsection, we propose to discuss the following aspects of the different approaches: model of a point set, point set dimensionality, operations on point sets, types of attributes, attribute model, and operations on attributes.

1.2 Enumerative and combinatorial representations

1.2.1 Enumeration using a mapping rule

An enumerative and combinatorial representation specifies the rules for generating points in the set and no other points. A popular rule to define a point set is to use a parametric definition. Points of a given space, called parametric space, are mapped to another using some mapping rules or functions. Enumeration of the points is a matter of marching along a given interval and applying the mapping rule. An interval can be defined as a unit segment, square, cube, or hyper-rectangle depending on the dimension of the object. In this case, the object is defined as a collection of 0D cells, i.e., points.

Mapping rules can be defined by classical parametric functions, such as Bézier, spline, and other [Far90, FDFH95]. Those functions are based on an interpolation or approximation of values given over a net of points, called control points. This approach, also known under the general term *free-form surfaces*, is very popular and widely used in several CAD systems. Examples of the concept of mapping a parameter space to the object space are given in Figs. 1.1*(top)* and *(middle)*, where respectively a curve and a solid are defined.

This representation based on parametric functions has severe limitations. Point membership classification requires complex numerical procedures as one has to define explicitly the inverse mapping. Furthermore, once an object is defined using some approximation functions, it can be hardly combined with others using set-theoretic operations for instance.

Another representation using also parametric functions is the *sweeping representation*. An initial shape, the *generator*, is moved along an arbitrary trajectory. Similarly, point coordinates of the object are obtained while marching along a given set of intervals. Several kinds of sweeps exist, such as rotational and translational (extrusion). An example is given in Fig. 1.1*(bottom)*, where a 3D solid is defined by a rotational swept. A general model is proposed in [Sny92], called *generative model*, where objects are defined by a sweeping in arbitrary dimension. Several operators are proposed, such as summation, Cartesian product, branching, integration, differentiation and other. This model can combine objects of different dimensions. For instance, an animated teddy bear with fur was shown in [Sny92], where the fur was modelled as curves, the bear as a surface, and the resulting object was time-dependent.

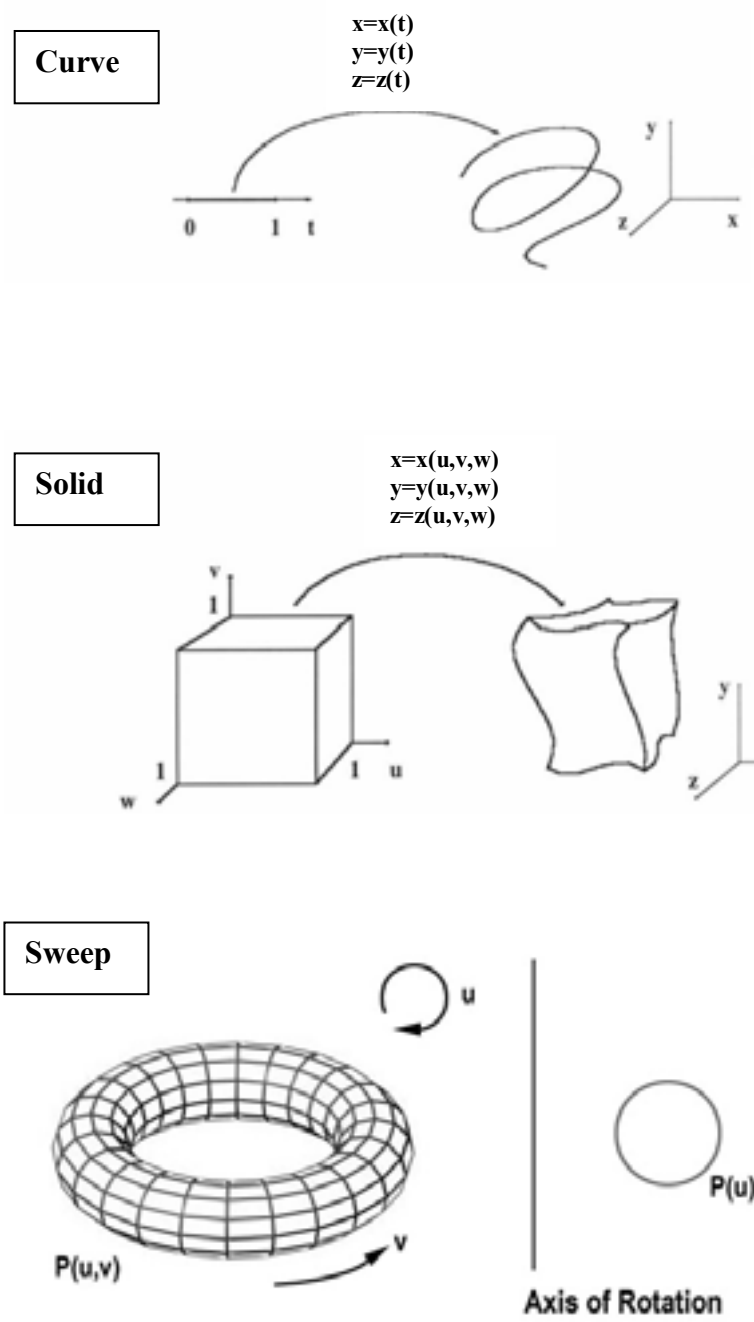


Figure 1.1: Enumeration representation using parametric rules. From top to bottom, a 3D curve, a 3D solid defined using an interpolation methods, and another 3D solid obtained by sweeping. Objects are defined while marching along the parameter spaces.

1.2.2 Grouping

Grouping is one of the simplest representation schemes, and the enumeration consists of a collection of homogeneous cells. A collection of cubic cells is an example of grouping, and the corresponding representation is called *voxels*. Another examples of representation using this scheme are polygon soup, union of overlapping spherical balls, ray representation and other [MMZ94, EKL+91].

Considering the object space as a discrete space, each cell is defined as one of its small subsets. If the entire space is discretised in a uniform manner, i.e., all the cells are strictly identical, it results in a uniform grid. In this case, enumeration is called exhaustive. If cells have different size, the enumeration is called *adaptive*. Finally, cells can be different, and correspond to a non-uniform grid. Examples of exhaustive and adaptive enumeration are given in Fig. 1.2, where the decomposition of a 2D solid is given. Figure 1.2*a* shows the resulting object, Fig. 1.2*b* shows a binary matrix, where "0" indicates that the cell is empty, and "1" indicates the presence of some material of the object. This is the exhaustive enumeration. Figures 1.2*c* and *d* show respectively an adaptive representation and its corresponding hierarchical data structure. Each square is recursively subdivided into four smaller squares when it intersects the initial object. This adaptive decomposition is called a quadtree in 3D, and an octree in 3D. Another adaptive representation that offers an efficient hierarchical structure is the *binary space partition*, where the object space is divided into regions. With such structures, point membership classification can be done in an efficient way.

Grouping cells are often used to define solids and volumes. A volume can be thought as a subset of 3D space with an additional scalar value given to each of its point. If this scalar value is interpreted as an additional point coordinate, the volume becomes a 4D "height field" or a hypersurface in 4D space. Scalar values can be given to the cells of a regular space grid (voxel data) or to a non-regular grid. In the simplest case, only binary values "0" or "1" can be assigned (see Fig. 1.2*b*). This binary voxel model is equivalent to the spatial occupancy enumeration, which is a well-known representation of solids [Req80]. This allows to model only homogeneous solids. This representation is well suited for mass properties calculations and other solid modelling applications.

In volume graphics, this model is associated with voxelisation procedures [WK93] supporting conversions of other models (presented below) to binary voxel data. As an extension of the *spatial occupancy enumeration*, integer or real scalar values can be given for each cell of a regular or a non-regular space grid of a heterogeneous volume model. Processing of scalar values given at discrete scattered points requires some approximation procedure [Nie93]. The scalar values can represent either the geometry of a point set (density field [UO91], distance field [PT92, JC94, Jon96]) or its physical attributes [Nie93].

Operations on voxel models include Boolean operations and linear transformations [UO91], transformations from one voxel data structure to another by manual 3D painting and carving [GH91, NF91, AS96], volume sculpting [WK95, Bae98, ATTY99, FCG00], metamorphosis [Hug92, LGL95], and morphological operations [OF00]. Most of the operations change the scalar values defining the object geometry. Operations on non-geometric attributes include approximation of scattered data [Nie93] and other specific operations.

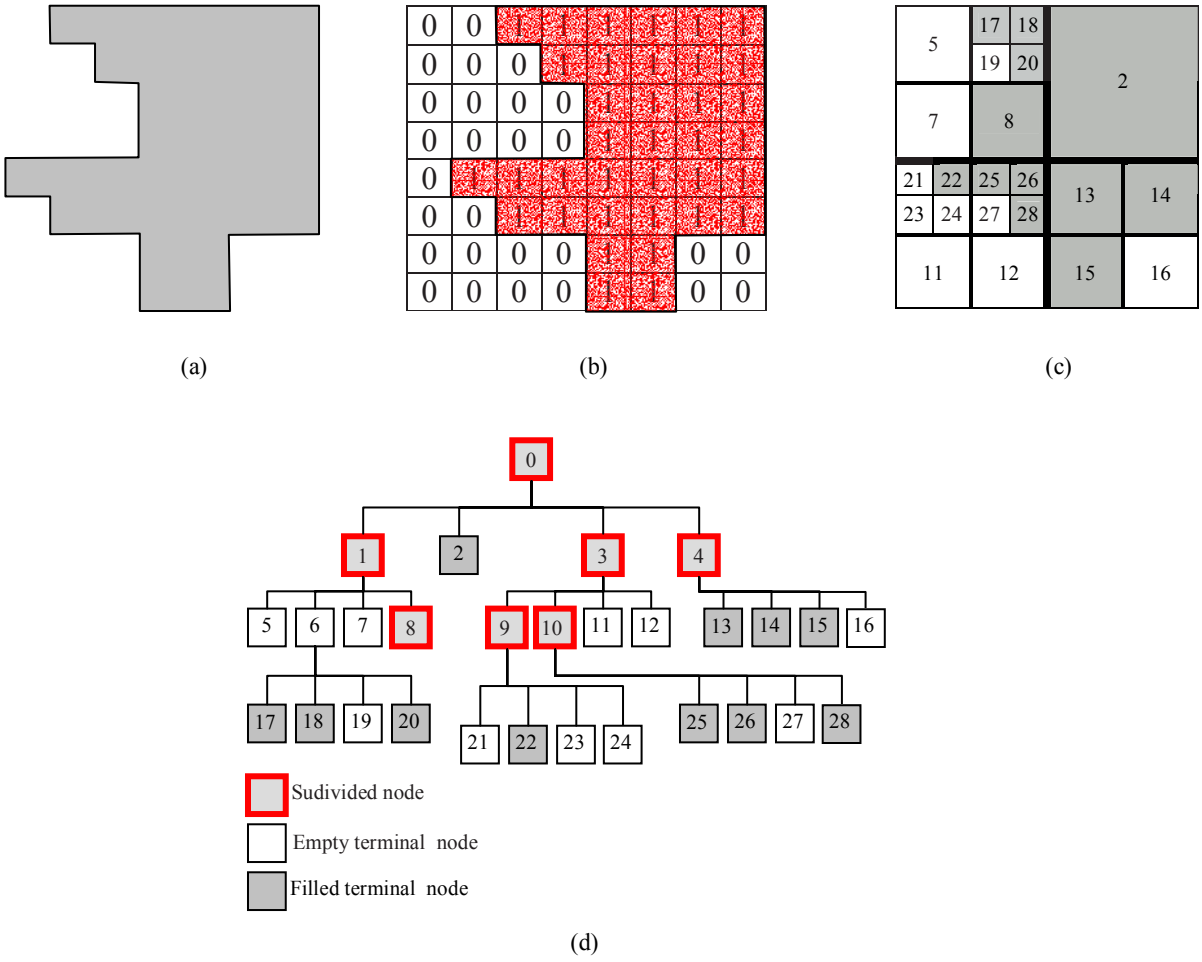


Figure 1.2: Spatial occupation enumeration. (a) Original 2D solid. (b) Binary representation, defined as an exhaustive enumeration. (c) Quadtree. Adaptive enumeration of the object. (d) Hierarchical data structure corresponding to the quadtree.

1.2.3 Cellular complexes

Cellular representation is also a combination of cells. The major difference grouping is that this model includes relationship between cells and its neighbourhood. Without formalism, the first polygonal models, such as wires, polygon quads and strips that were used in computer graphics were simplicial complexes.

The main relations defined between elements of a cellular complex are incidence and adjacency. We can consider the relation "to be properly joined" between two cellular complexes (see [FK97] for details on the conditions as they would require a heavy mathematic description, and are out of the scope of this survey). Point membership, inclusion and intersection relations are valid for the cellular representation, CRep for short. All operations defined for CRep can be subdivided into three groups: analysis, synthesis and conversion. The first group includes operations for topological analysis such as determination of connectivity and orientability, evaluation of the homology and homotopy groups, and others [FK97, Kar99]. The synthesis operations include cell complexes' generation procedures, geometric transformations, set-theoretic operations for properly joined complexes, various reconstructions of complexes (e.g., cell subdivision or collapse, replacement of a subcomplex by another one with the same boundary, optimization of complexes, transformation of a simplicial complex into a cellular one with the same carrier [Kar99], etc.), selection of subcomplexes according to various restrictions (for example, boundary or co-boundary of a cell, boundary of a complex, a common subcomplex for two properly joined complexes, etc.) and topology transformation (e.g., slicing of h-genus surfaces or solids [Kar99]). The third group involves procedures of conversions between other models and CRep such as evaluation of functional representation for cellular complexes (see the discussion below), polygonalisation of functionally defined objects [PPP88], and tetrahedrisation of 3D solids [Loh97].

Cellular complexes representation is based on a reliable mathematical framework. From the modelling point of view, to construct an object using a CRep scheme directly is far too difficult and applications of such model is more dedicated to physical simulation, such as heat transfer, material resistance calculation and other areas of finite element analysis. Several attributes can be defined in the CRep model, but inside a given cell, they are constant. Another important feature is that cellular complexes can be composed of cells of different dimension.

1.3 Function representation

Another approach than the enumeration scheme is to use a predicate A to define a set of points S that can be evaluated for every point X:

$$S = \{X | A(X) = true\} \tag{1.1}$$

Any real-valued function F, and the inequality $F(X) \geq 0$ can be used as a valid predicate. In the case of equality, and in a 3D Euclidean space, the surface of a solid is described, and is known in the literature as an *implicit surface*. In the following, we first recall the main results and ideas of this area and then present a more general representation, called *function representation*, FRep for short, that spawns this concept.

1.3.1 Implicit surfaces

Since the past two decades, a new modelling technique has appeared, based on the use of real-valued functions. For 3D solids, a real-valued function is a function F, defined in the Euclidean space E^3, that maps every given points of E^3 to a real value. Depending on the resulting value, a classification can be made, as three closed subsets are defined:

- if $F(X) < T$, the point X is outside the object,
- if $F(X) = T$, the point X lies on the boundary of the object,
- if $F(X) > T$, the point X is inside the object.

The real value T is called *threshold value*. The function F can also be called a *field function*. The equality $F(X) = T$ defines a surface, called *iso-surface*. If $T = 0$, this surface can be also called *zero-set* or *zero-surface*. For instance, one can define a sphere of radius R and centred at the origin as follows:

$$F(X) = R^2 - x^2 - y^2 - z^2 \tag{1.2}$$

where $X = (x, y, z)$ is a point of the Euclidean space E^3.

In the literature, the iso-surfaces are also often called *implicit surfaces*[1]. Depending on the nature of F, several terms are used to describe this representation. Historically, one of the first formulation of the function F was proposed by Blinn [Bli82], based on a composition of an exponential function with a distance function. Resulting surfaces are called equi-potential implicit functions, and objects are often denoted as *soft objects* or *blobby objects* [WMW86]. Nishimura *et al.* [NHK+85] proposed later a similar technique, which is called *metaballs*. Several other works followed to propose different definitions of the field function F, based on Blinn's initial work [MI87, Gas93, BS95]. Those techniques are generally point-based, i.e., to create an object, different points, called *sources*, are specified in space, and the combination of their respective field functions defines an object. Several works propose the definition of sources of other natures, such as line-segments, arcs, triangles, and other [BS91, MS98]. Those extensions lead to the techniques called *skeleton-based* or *convolution based* modelling.

Another technique is the algebraic method, where the function F is defined as a polynomial. The example of the sphere given above falls into this category. When the polynomial is of degree two, the set of generated objects is called *quadric*. It includes several shapes such as the sphere, the ellipsoid, the cylinder, the torus, the hyperboloid, the cone, and other. The superquadratics proposed in [Bar81] is an extension of the quadrics, where a rational polynomial is used. Another formulation is proposed in [BS96], and the resulting shapes are called *ratioquadrics*.

[1] To avoid ambiguities, a short discussion is needed about the word "implicit". Under no circumstances the function F is an implicit function. Indeed, an implicit function cannot even define a sphere. The function $f(x, y) = z$ is called an explicit function of two variables, where z is a function of x and y. Then, the function $f(x, y, z) = 0$ is called an implicit function of two variables. Here, z is also a function of x and y, but is defined implicitly. This is not enough to define a sphere. The zero-set of an explicit function defined above is written as $F(x, y, z) = 0$. It looks exactly the same, but has completely different meaning

1.3.2 Function representation

As one can see, a lot of researches are made in the field of shape modelling using real-valued functions. One can also notice that there is no unifying framework, and each separate research leads to a different terminology. Pasko et al. introduced in [PSAS93, PASS95] a general framework and a unifying model for solid modelling using real-valued function, the so-called *function representation*, FRep for short. 3D objects, but also time-dependent and objects of higher dimension are covered by the FRep model. Furthermore, the geometric domain of this model in 3D space also includes solids with non-manifold boundaries and lower dimensional entities (surfaces, curves, points).

FRep is formulated as an algebraic system including sets of objects M, operations Φ and relations W on them, expressed as the triple (M, Φ, W).

Objects

The FRep of a geometric object M can be expressed as:

$$M : G = X | X = (x_1, \ldots, x_n) \in E^n, F(X) \geq 0 \tag{1.3}$$

where G is the point set that defines the object, X is a point of the Euclidean space E^n. G is defined by the inequality $F(X) \geq 0$, where F is a real-valued function (defined in E^n). The function F is called *defining function*, and the above inequality the function representation of the geometric object. The only restriction on F is to be at least C^0 continuous.

If one wants to describe a straight line segment in a 2D space, the following equation can be used:

$$F(X) = ax + by + c \tag{1.4}$$

where $X = (x, y)$. The inequality $F \geq 0$ defines a halfplane. Then, $-F^2(X) \geq 0$ defines the line itself, where in fact the function $-F^2$ is never positive and only becomes zero on the line. The line can be trimmed using some 2D solid to produce one or several segments.

For 3D objects, solids bounded by algebraic surfaces, skeleton-based implicit surfaces, and convolution surfaces, as well as procedural objects (such as solid noise), and voxel objects can be used as defining functions. In the case of a voxel object (discrete field), it should be converted to a continuous real function, for example, by applying a trilinear or higher-order interpolation.

In general, the function F can be defined by an equation or by a "black box" procedure converting point coordinates into the function value. This possibility allows one to divide the set of objects M in two categories. The first one contains objects described by a specific instance of an equation, and are usually called *primitives*. The second contains other complex objects, obtained while combining primitives under the use of operations defined in Φ.

Operations

The set of geometric operations Φ defined in the FRep model can be expressed as:

$$\Phi_i : M^1 + M^2 + \ldots + M^n \rightarrow M \tag{1.5}$$

where Φ_i is a n-ary operation. Many operations such as set-theoretic ones, blending, non-linear deformations, metamorphosis, sweeping, hypertexturing, and others, have been formulated for this representation in such a manner that they yield continuous real-valued functions as output [PASS95, SP98], thus guaranteeing the closure property of the representation.

Let us consider for instance the basic set-theoretic operations. To insure a C^k continuity and to provide an exact definition, R-functions originally introduced in [Rva63], and later in [Sha88, Pas88, PSAS93, Sha94], are used. Nevertheless, in practical, the following definitions are used (but still, are defined on the base of R-functions):

$$\begin{cases} f_1|f_2 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2} \\ f_1\&f_2 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2} \\ f_1\backslash f_2 = f_1\&(-f_2) \end{cases} \tag{1.6}$$

This definition of set-theoretic operations yields a C^1 discontinuity, when both functions are equal to zero. But still this definition is preferable than the one proposed in [Ric73a] :

$$\begin{cases} f_1|f_2 = max(f_1, f_2) \\ f_1\&f_2 = min(f_1, f_2) \end{cases} \tag{1.7}$$

In this case, there is also a C^1 discontinuity that occurs when both functions are equal.

Figure 1.3 is given as an illustration. An object is defined as an intersection between two spheres, shown in green in Fig. 1.3*a*. A third sphere, yellow, is added to the object while using a blending union operation. In Figs. 1.3*b* and *c*, different union operations are used for two initial spheres, respectively a min function and a R-function. As one can see, in the first case, an edge appears along the blend, corresponding to the set of points where the defining functions of the two intersecting spheres are equal. In case of the R-function, no edge appeared, and the result is smoother.

The formulations of other operations can be found in [PASS95]. Let us mention that because of the closure property, the proposed operations allow to define functions of complex objects using a constructive approach. Starting from simple ones (primitive) and applying a sequence of constructive operations to them, a tree is built as a special data structure with primitives in its leaves and operations in its nodes. This leads to one of the main distinctive characteristics of the FRep model, which is the real-valued function defining the object is evaluated at the given point by a procedure traversing a tree structure. An example of tree is given in Fig. 1.4. Several primitives are used, corresponding to the functions f_1 to f_6, and combined using the set-theoretic operations, defined in eq. 1.6. The resulting function F is also a real-valued function, and is defined as:

$$F = (((f_1\backslash f_2)|(f_3\backslash f_4))|f_5)\backslash f_6 \tag{1.8}$$

1.3.3 Other constructive approaches

Historically, FRep was not the first representation that uses real-valued functions to define primitives for a constructive tree. Indeed, the first constructive approach was formulated in the CSG model, standing for *Constructive Solid Geometry*. Widely used in computer graphics, the

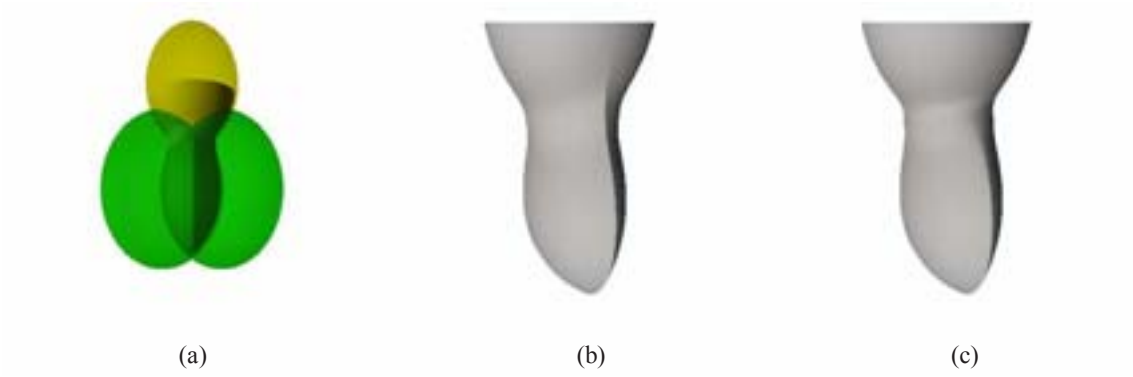


Figure 1.3: Union operation. (a) The object is composed of three spheres, combined using a blending union and a set-theoretic intersection operation. (b) The intersection is defined using a min/max function. A discontinuity along the material added by the blend appears. (c) Set-theoretic intersection operation defined using an R-function; the edge due to the discontinuity disappeared.

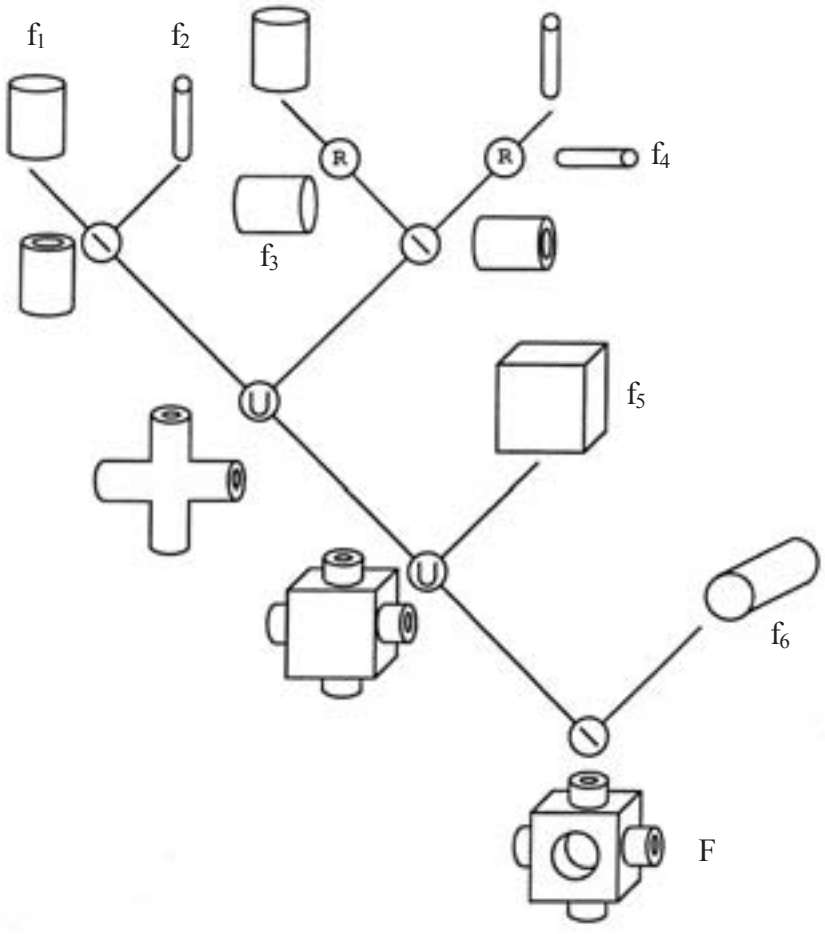


Figure 1.4: Example of constructive tree in the FRep model. Leaves of the tree contains primitives, defined as real-valued functions, f_1 to f_6. The resulting object is also a real valued function F. Its expression is given in eq. 1.8.

CSG representation uses predicates and Boolean set operations in its tree structure. Primitives can be defined as a halfspace, box, cone and other simple shapes. The major requirements are:

- every available primitive has to be closed regular
- operations have to be regularised

Two pseudo procedures to evaluate a primitive, respectively a sphere and a box, can be defined according to the following algorithms (in C-style):

```
Sphere
Boolean isInside(float x, float y, float z) {
      return (r*r < x*x+y*y+z*z);
}
Cube
Boolean isInside(float x, float y, float z) {
      return ((x<left) & (x>right) & (y<top) & (y>bottom) &
              (z<front) & (z>back));
}
```

where r and *left,right, top, bottom, front* and *back* are parameters defining the location of the primitive in space. Regular set-theoretic operations, union $\cap$, intersection $\cup$, difference $\setminus$ and inverse $-$ are the main available nodes in a CSG tree. The Bool algebra is used while applying a tree traversing procedure to determine whether or not a point belongs to the set.

The set of available primitives can be extended to other objects, for instance to implicit surfaces as in [Bow95], but the set of available nodes can hardly be extended, due to the requirement imposed by the regularity condition. CSG representation has a multidimensional extension, proposed in [WB96]. Attributes can also be defined in the CSG model. An advanced modelling system, Svlis [Bow95], is based on an extension of CSG where one can assign a fixed index of material to each primitive of a constructive tree. While traversing the constructive tree to determine the geometry, a material index is also determined, while using regularised Boolean set-theoretic operations for geometry and special corresponding operations for attribute indices.

Another constructive approach is proposed in [WGG99], and is called *BlobTree*. All the primitives are skeleton-based primitives, and are defined as a composition of a distance function with a potential function [WMW86]. Classical set-theoretic operations are defined using min/max functions, but can be extended easily to R-functions. The FRep model includes the BlobTree as a subset. An advanced set of operations on photometric attributes for the BlobTree is proposed in [TW99]. While building the constructive tree of an object, textures can be assigned to each primitive. Union and blending operations that are applied to the geometric primitives are also applied to the attributes, resulting in smooth transitions. Attributes and geometry are deeply connected. As it will be discussed at the end of this chapter, this connection is somewhat controversial.

1.4 Hybrid models

1.4.1 Boundary representation and extensions

A solid described using the boundary representation (BRep) is based on the description of the surface, subdivided into cells of different dimensions. Vertices are defined as 0D cells, edges as 1D cells, and faces as 2D cells. Result of such combination is then embedded in a 3D space. Relationship between each cell of same dimension but also of different dimension is explicitly defined, and corresponding graphs can be retrieved from this.

Object

BRep can be presented as a combination of different approaches, enumerative, combinatorial and implicit. Indeed, vertices are enumerated, as well as the connections between edges and faces. The solid is defined implicitly by its boundary. The term "implicit" may be misleading. In this case, it refers to the fact that the point set X is implicitly defined by all the points that satisfied the following predicate:

$$X = \{p | p \in \text{set bounded by } \delta X\} \tag{1.9}$$

δX has to guarantee that it separates the Euclidean space E^3 in exactly two subsets, one is the bounded interior of X, the other the unbounded exterior space. The boundary δX can be defined using any representation, implicit, constructive, parametric or combinatorial. The validity conditions on the boundary can be expressed as follows:

- to be a valid cell complex (disjoint cells satisfying the frontier condition)
- to be homogeneously 2D
- every edge shared incident on an even number of faces
- to be orientable

This representation scheme is the most widely used in CAD systems. The major advantage of such representation is the information it provides on the topology of the object. The planar graphs defining vertices/edges or edges/faces relationship can be deduced from the representation. Figure 1.5 is given as illustration. A cube is described using BRep. The original object is shown in Fig. 1.5*a*, its corresponding set of faces in Fig. 1.5*b*, its set of edges in Fig. 1.5*c*, and its set of vertices in Fig. 1.5*d*. An example of a planar graph is shown in Fig. 1.5*e*, and illustrates the relationship between faces and vertices. Exhibition of other relations can be also made using other graphs, as for instance, a faces/edges or edges/vertices relationship.

Operations

To construct a solid using a boundary representation, different operators are available. For every modelling step, the validity of the object is verified using the Euler formula, defined as follows:

$$V - E + F = 2(S - H) + R \tag{1.10}$$

where V is the number of vertices, E the number of edges, F the number of faces, S the number of disjoint components, H the number of holes in the solid, and R the number of cavity in a face.

This equation 1.10 has to be verified for every time step in the modelling process. A finite set of operations are proposed, known as the Euler operators, where one can add/remove vertices and edges under some conditions. A dozen of operators are available. A more detailed description and can be found in [Man88]. Basic operators are shown in the following table:

M	Make
K	Kill
S	Shell
H	Hole
R	Ring
F	Face
V	Vertex
E	Edge

The operators M and K and respectively the construction and the destruction operators. By combining them together, advanced operators can be obtained. For instance, when one applies the operator MEV, it creates an edge and a vertex, or the operator KEF destroys an edge and a face.

Extensions

In most of the CAD systems, BRep and CSG representations are combined together, taking advantages of each model. A tree structure is still used, where primitives are defined either as CSG solids, or solids defined by their boundary. While building a constructive tree (in a CSG style), boundary elements are added to insure the validity of the model.

This hybrid representation has several advantages, such as the possibility to build a model easily using a constructive approach, the simplification of the evaluation of the volume for further numerical simulations, the possibility to add constraints for mechanical engineering, and other. Nevertheless, this representation has also several drawbacks. To store such representation is expensive, to maintain the validity of both models is a difficult task, and operations that were fast for one model or the other become more complex due to the evaluation mechanism needed to retrieve the desired information.

Another hybrid model, combining the BRep and CSG models are proposed in [KD97], where a solution for defining heterogeneous solids, composed of multiple materials, is proposed. A 3D solid is subdivided into components made of unique materials. A non-manifold BRep scheme is used to model such objects. Each component is homogeneous inside and has an assigned index of material. Regularised set-theoretic operations are applied to the solid components. Corresponding operations on material indices are introduced on the basis of the resulting material selection for each pair of materials and for each set-theoretic operation.

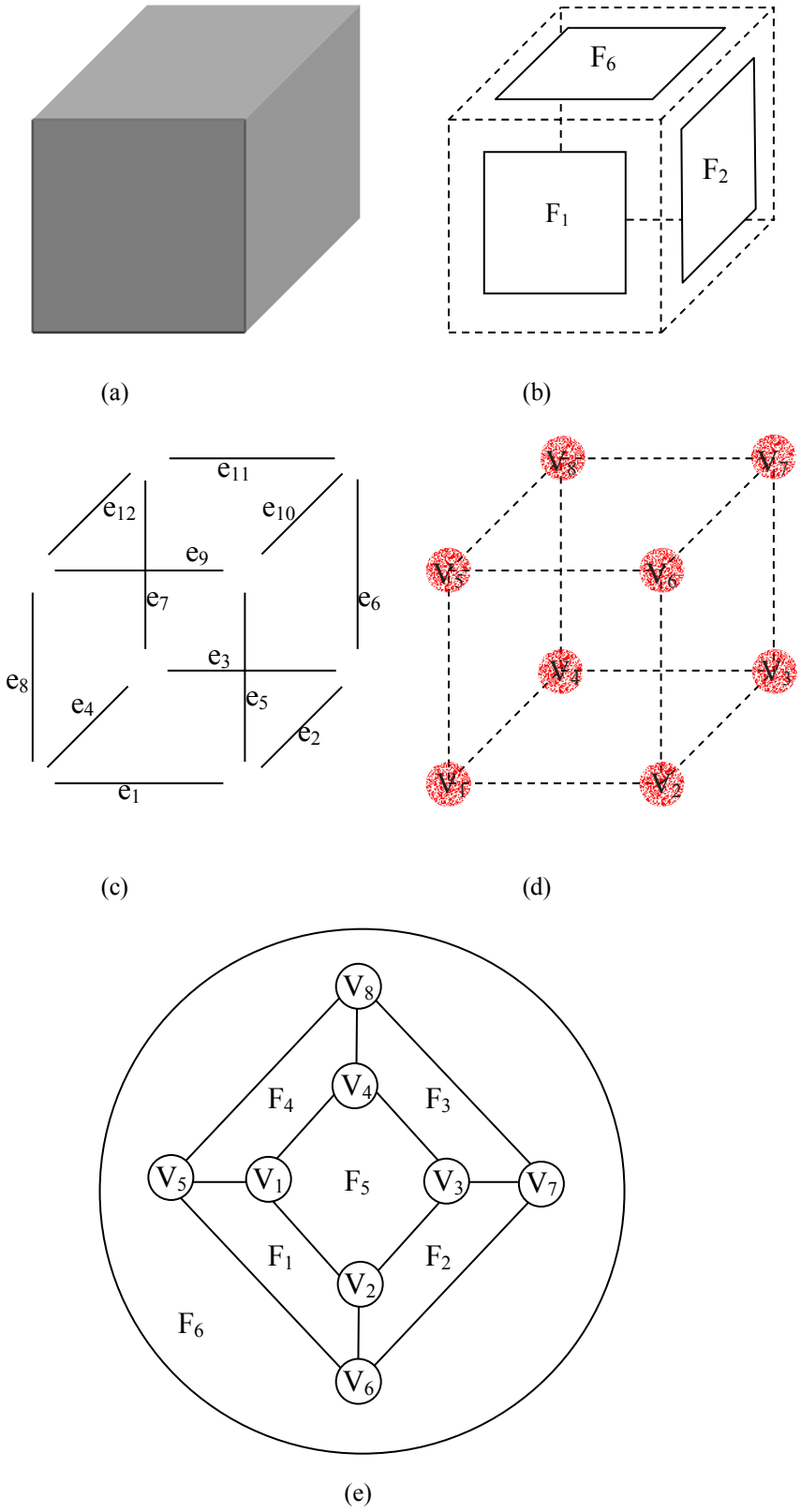


Figure 1.5: Boundary representation. (a) The initial object, a cube. (b) List of faces. (c) List of edges. (d) List of vertices. (e) Example of a planar graph describing the relationship between vertices and faces.

1.4.2 Hybrid volume

Hybrid volumes are modelled using a combination of FRep and discrete field volumes [AKPS00]. The object is represented by a tree structure with functional and voxel primitives. The tree can have nodes representing functional, voxel, or hybrid operations. The basic feature of a hybrid operation is the absence of any explicit conversion between two representations on the level of the object specification and in the tree structure. Depending on the types of the arguments and the result, such function can invoke the conversion procedures with subsequent call to the voxel or FRep libraries' functions. The hybrid model can represent the geometry or a non-geometric attribute.

1.4.3 Object model

A general object model [KBDH99] includes all the characteristics and attributes of an object. Geometry is considered as the most fundamental attribute of an object. All other attributes are described as a function of geometry. A 3D point set is represented by its decomposition (atlas) into a finite set of closed 3-cells. Each 3-cell is a compact connected 3-manifold. The authors proposed to use BRep scheme to model individual cells and the entire point set. Each point of the point set is mapped to its corresponding attribute, which can be a vector or a tensor. The model of attributes is a collection of functions mapping the object geometry to several attributes. The general object model combining the point set and attributes models is represented by a trivial fibber bundle, where the attributes are strictly attached to geometry as sections of the bundle. This is a generalisation of the multi-material solid model [KD97] discussed above (in the subsection "Extensions" of the BRep presentation). Affine transformations and regularised set-theoretic operations can be applied to point sets. Basic operations on attributes include vector sum and product with scalar, union, intersection and complement specialised for specific attributes (abstraction of the material combining operation [KD97]).

1.4.4 Constructive volume geometry

In Constructive Volume Geometry (CVG) [CT98, CT00] a spatial object is defined as a tuple of scalar fields defined in 3D space :

$$o = (O, A_1, \ldots, A_n) \tag{1.11}$$

where O defines the opacity of the object, and the A_i different photometric attributes, such as colour, ambient, diffuse, and specular reflection parameters.

A special attention is paid to the opacity field, $O : E^3 \to [0, 1]$, that determines the visibility of every given points in E^3. In other words, the opacity defines implicitly the "visible geometry" of an object. While substituting the interval $[0, 1]$ to the Boolean domain $B = \{0, 1\}$, one can easily use the set-theoretic operations of the CSG representation. Other operations on attributes, O and A_i, are also proposed, such as blend, arbitrary selection and other. In the general definition (e.q. 1.11), scalar fields are used, but discrete fields can also be included in the model using some interpolation procedure. The model is presented as an algebra of 3D static spatial objects.

1.5 Discussion

Historically, separate treatment of geometry and attributes was introduced in computer graphics for rendering textured surfaces. Voxel arrays in volume graphics can be considered as attribute models with the default geometry represented by a bounding box. A subset of the bounding box geometry can be defined by introducing some constraints on the attribute values. For, example, an iso-surface is defined inside the bounding box as a surface of a constant value of some attributes. The next step of models development was to combine geometric and attribute representations in a single model. In solid modelling, this was done for multi-material solids [KD97] with the material indices assigned to different geometric regions. Then, this approach was generalised in the object model [KBDH99] covering arbitrary geometry and multiple attributes of different mathematical types (scalars, vectors, tensors) defined at each point. Only 3D geometry is considered in the object model with the boundary representation being the primary geometric model. The object model does not include voxel arrays or scalar fields for modelling geometry.

In volume modelling, CVG [CT98, CT00] was one of the first model, with the general object model [KBDH99], that combines geometry and attributes in a systematic manner. The CVG model is presented as an algebra of spatial objects with operations available for both geometry and attributes. The model allows for utilising voxel arrays and continuous scalar fields. The purpose of the opacity field in the model is to "implicitly define the visible geometry of an object". This idea is somewhat controversial and needs additional discussion. There are scalar fields directly connected to the object's geometry, for example, density and distance fields. In reality, the shape of an object does not necessarily predefine its photometric characteristics and vice versa. We believe it is important that a point set and its visual and physical characteristics are represented independently. Using opacity to define the geometry limits application of this approach to graphics and visualisation. Probably, taking this into account, a possibility was reserved in the CVG model to include an additional scalar field to explicitly specify geometry.

A quite limited set of geometric operations was developed for implicit surfaces and later adapted in CVG. For example, it is well known that using min/max functions for set-theoretic operations causes problems in further transformations of the model due to C^1 discontinuity of the resulting function (see Fig. 1.3). Further blending, offsetting, or metamorphosis can result in unnecessary creases, edges, and tearing of such an object.

Only 3D static objects are considered in the object model and CVG. No time-dependent or multidimensional objects are included in these models. However, multidimensional models do have potential to be exploited. Constructive modelling of higher dimensional objects using scalar fields and an extended set of geometric operations is discussed in the next chapter.

Note also that the object model introduced in the area of solid modelling is oriented towards the mechanical design and rapid prototyping applications. On the other hand, CVG has its origin in volume graphics and is mainly aimed to providing more flexible object and scene definitions in volume rendering. These two areas look like two separate worlds now, and the motivation of our work is to introduce a universal model that being supported by appropriate techniques and tools can be suitable for above mentioned and other applications.

Chapter 2

Constructive hypervolume framework

2.1 Introduction

Based on the different existing models presented in the previous chapter, we can specify a general model of point sets with attributes.

Three important requirements can be expressed as follows:

- independent representation of the point set and its attributes;
- uniform treatment of point set geometry, photometric, physical and other attributes of arbitrary nature;
- coverage of time-dependent and other multidimensional point sets.

First, the independence of a point set and its attributes is a natural requirement. As a trivial example, let us consider a red apple. The point set is the apple, and the red colour is an attribute. It is obviously clear that the colour attribute does not define the geometry (other objects than an apple are red), and the geometry does not define the attribute (look at a green apple to be convinced of). Another example, less trivial, is the *Invisible Man* [Wel97]. The whole space, and in particular the body of the Invisible Man, are fully transparent. The opacity does not define here the geometry of the point set. Then, one can state that all attributes have to be independent of the point set geometry.

Second, the point set and its attributes should be treated uniformly. Indeed, when one applies some operations to a point set, the same operation has to be also applied to attributes if they are considered assigned to specific points of the object. For instance, if a twist is performed on a simple box with a checker-board texture pattern, it clearly appears that the whole object is deformed, i.e., the texture follows the geometry deformation.

In this chapter, we introduce and discuss a general model of constructive hypervolumes satisfying the requirements listed above. This model is an "instance" or a particular implementation of the general object model [KBDH99] extended to the multidimensional case and it is based on combining the advantages of FRep [PASS95] and hybrid volumes [AKPS00]. This model

also conforms with, and further develops the volume model [Nie00] and the constructive volume geometry model [CT00].

The ability to model both geometry and attributes using real-valued functions is a promising solution. The use of scalar fields allows for defining everywhere in space both point set and attributes exactly, and is a good alternative for instance to the problem of constant attributes inside of a cell in the BRep or CRep models. The FRep model also provides an extendable set of primitives, operations and relations that allow, through a constructive approach, further treatment of the point set with attributes.

Finally, let us emphasise the dual nature of FRep: an FRep function can be considered not only as defining a solid (or an implicit surface), but also as a scalar field defining an attribute, and can be used to model amorphous or gaseous phenomena as well as their physical properties. Note that in the latter case the constructive approach can be applied to modelling attributes rather than to geometry. In the case of using FRep to model solids, attributes can be assigned only to the entire point set and no operations on attributes are provided. The dimension independent formulation of FRep allows the modeller to treat animated and other multidimensional objects in the uniform manner.

In the next section, we give a formal definition of a general definition of constructive hypervolumes[1]. We propose then to use this new model in different case studies related to different areas, such as modelling heterogeneous objects, a visualisation of geological layers and to a physical simulation. A short description of the language used to define FRep object is presented, i.e., the HyperFun language [Pro], and its extension to the proposed model is discussed.

2.2 Constructive hypervolume modelling

Extending the FRep formal model introduced in [PASS95], let us describe a general hypervolume model as a triple (O, Φ, W), where O is a set of hypervolume objects, Φ is a set of hypervolume operations, and W is a set of relations for the set of objects. Mathematically, the triple can be treated as an algebraic system.

2.2.1 Objects

A hypervolume object can be expressed as a tuple, $o = (G, A_1, \ldots, A_k)$, where G is a multidimensional point set and A_i is an attribute. In 3D, a point set G can be defined using any existing representational schemes for solids: BRep, CSG, spatial partitioning, generative models, ray implementation, and others (see previous chapter). In the multidimensional case, one can apply multidimensional extensions of CSG or BRep [WB96, GMR99], or originally multidimensional models such as the generative model [Sny92] and the FRep [PASS95]. Here we introduce a specific "FRep" representation of the hypervolume object that can be expressed as:

$$o = (G, A_1, \ldots, A_k) : (F(X), S_1(X), \ldots, S_k(X)) \tag{2.1}$$

where :

- $X = (x_1, \ldots, x_n)$ is a point in n-dimensional Euclidian space E^n,

[1] This work has been published in [PASS02]

- $F : X \to \Re$ is a real-valued defining function of point coordinates to represent point sets G. Therefore, F is at least a C^0 continuous function, which is positive inside the point set, negative outside, and has a zero value on its boundary.

- $S_i : X \to \Re$ is a real-valued scalar function corresponding to an attribute A_i that is not necessarily continuous.

2.2.2 Operations

The set of operations $\{\Phi_i\}$ includes operations of the type:

$$\Phi_i : O^1 + O^2 + \ldots + O^m \to O \tag{2.2}$$

where m is a number of operands of an operation. The result of an operation of this sort is also a hypervolume object of the set O, which insures the closure property. Accordingly, to create a complex hypervolume object, a sequence of operations can be applied over initially defined objects. An algebraic expression ("term") representing the composition of such a sequence of operations can be recursively defined in BNF as:

$$t ::= o | \Phi(t, \ldots, t), \tag{2.3}$$

where t is a term, o - an object, and Φ is an m-ary operation.

Let the object o_i be defined as:

$$o_i = (G_i, A_{i_1}, \ldots, A_{i_k}) : (F(X), S_{i_1}(X), \ldots, S_{i_k}(X)) \tag{2.4}$$

Then, for a general unary operation, the object o_2 can be derived from the initial object o_1 (note, that the objects can in principle have a different number of attributes):

$$o_2 = (G_2, A_{2_1}, \ldots, A_{2_k}) = \Phi^1(o_1) = \Phi^1(G_1, A_{1_1}, \ldots, A_{1_m}) \tag{2.5}$$

which, at a functional level, corresponds to a composition of the functions on scalar fields:

$$\begin{aligned} (F_2, S_{2_1}, \ldots, S_{2_k}) = \quad & (\Psi(X, F_1, S_{1_1}, \ldots, S_{1_m}), \\ & \Omega_1(X, F_1, S_{1_1}, \ldots, S_{1_m}), \\ & \qquad \vdots \\ & \Omega_k(X, F_1, S_{1_1}, \ldots, S_{1_m})) \end{aligned} \tag{2.6}$$

where Ψ is a function giving a new geometric point set and Ω_j is a function giving a new attribute. Note, that these functions Ψ and Ω_j have the following features:

- they are pointwise in the context of space E^n;

- they can explicitly be dependable on X, thus allowing for defining both the objects' geometry and attributes through transformations of a coordinate space;

- they are $(m+2)$-ary in a general case which allows us to deal with models where the objects' geometry can influence the objects' attributes and vice versa, thus providing a powerful and flexible framework. Of course, in a particularly frequent case the new object's geometry cannot depend on the initial object's attributes; the same is true of the regarding functions Ω_j;
- Ψ must be at least C^0 continuous and thus complies with FRep defining function;
- Ω_j can be discontinuous.

Similarly, one can build a framework for binary hypervolume operations. If o_1 and o_2 are initial objects, and o_3 is a new object, we have (note that these three objects can in principle have a different number of attributes!):

$$o_3 = (G_3, A_{3_1}, \ldots, A_{3_k}) = \Phi^2(o_1, o_2) = \Phi^2(G_1, G_2, A_{1_1}, \ldots A_{1_l}, A_{2_1}, \ldots, A_{2_m}) \tag{2.7}$$

which at a functional level corresponds to a composition of the functions on scalar fields:

$$\begin{aligned}(F_3, S_{3_1}, \ldots, S_{3_k}) = \quad & (\Psi(X, F_1, S_{1_1}, \ldots, S_{1_l}, F_2, S_{2_1}, \ldots, S_{2_m}), \\ & \Omega_1(X, F_1, S_{1_1}, \ldots, S_{1_l}, F_2, S_{2_1}, \ldots, S_{2_m}), \\ & \vdots \\ & \Omega_k(X, F_1, S_{1_1}, \ldots, S_{1_l}, F_2, S_{2_l}, \ldots, S_{2_m})).\end{aligned} \tag{2.8}$$

The general n-ary operation $\Omega_n(o_1, o_2, \ldots, o_n)$ can similarly be built and functions Ψ and $\{\Omega_j\}$ are supposed to have the features described above for unary operations.

2.2.3 Relations

Relations are defined over hypervolume objects $\{o_j\}$ and, possibly, some objects of another nature (numbers, points, etc.) $\{q_k\}$. They are generally represented in the form of n-valued predicates over functions representing hypervolume objects and other objects by using relations over real numbers and logical expressions.

The set of relations W includes such relations as :

$$w_i(O_1, \ldots, O_m, Q_1, \ldots, Q_k) : \Gamma(F_1, S_{1_1}, \ldots, S_{1_{l1}}, F_m, S_{m_1}, \ldots, S_{m_{lm}}, Q_1^{'}, \ldots, Q_k^{'}), \tag{2.9}$$

where $Q_i^{'}$ is a function representing a non-hypervolume object Q_i, Γ is a predicate.

The examples of binary relations over point sets G, such as inclusion, point membership and intersection can be found in [PASS95].

The basic point membership relation can be formulated for a point with attributes:

$$P_a = (X, a_1, \ldots, a_2) \tag{2.10}$$

and the hypervolume object:

$$o = (G, A_1, \ldots, A_k) : (F(X), S_1(X), \ldots, S_k(X)), \tag{2.11}$$

using a three-valued predicate:

$$\Gamma_3(P_a, o) = \begin{cases} 0, F(X) < 0 \; or \; \exists j : S_j(X) \neq a_j \\ 1, F(X) = 0 \; and \; S_j(X) = a_j, j = l, k \\ 2, F(X) > 0 \; and \; \forall j = l, k : S_j(X) = a_j \end{cases} \tag{2.12}$$

This means that the point with attributes belongs to the hypervolume object, if it is an internal or boundary point of the point set, and its attributes are equal to the object attributes. This can help to answer such queries as "Is this point of the object red? ". The answer will be positive, if the point belongs to the object and its red colour attribute is equal to the same attribute of the object. This example shows that the attributes equality condition is not always necessary. For example, the positive answer to the above query could be generated in the case of any non-zero value of the red colour attribute. More sophisticated versions of the predicate can be introduced using specific conditions for the attributes.

2.2.4 Constructive hypervolume model and its underlying representation

We call the introduced representation *a constructive hypervolume model* to emphasise the underlying constructive process while modelling functionally based multidimensional point sets with attributes. As it was described in [PASS95], formally specified in [PSAS93], and recalled in the previous chapter, the main distinctive feature of FRep is that the real-valued function F defining the point set is associated with a tree structure that serves as its underlying representation. The function F is evaluated at the given point by a procedure traversing the tree structure with primitives in the leaves and operations in the nodes of the tree.

As to the constructive hypervolume model, its underlying representation can be defined in a similar way by introducing a set of tree structures. Along with the tree corresponding to a function F defining the point set, there are constructive trees associated with functions $\{S_j\}$ defining attributes and reflecting the construction logic of the attribute definition. Two main types of elements of the set O are considered: basic hypervolume objects (primitives) and complex hypervolume objects. A hypervolume primitive is a specific instance of a function chosen from a finite set of possible types. Primitives can be either predefined (and stored in the library) or introduced on the fly. A complex hypervolume object is the result of operations on primitives. The tree structure with hypervolume primitives in the leaves and hypervolume operations in the nodes of the tree provides the computational scheme for complex hypervolume objects. Some nodes, including root nodes corresponding to the whole complex object, can refer to the hypervolume relations.

The function S_j 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 accordance with the constructive nature of the definition and the underlying representation. A formal description for the traversing procedure for the FRep constructive tree [PSAS93] is easily adapted to hypervolume constructive trees.

The constructive tree is similar to one used in CSG, and is created during the object construction process. In contrast to classical CSG, the sets of primitives and operations are not fixed and can easily be extended without redesigning the modelling system, and all operations

are applicable on any level of the tree. As to the geometric constituent, solids bounded by algebraic surfaces, skeleton-based implicit surfaces and convolution surfaces, as well as procedural objects (such as solid noise), swept, and discrete field objects can be used as primitives. Let us mention in particular that the framework is general enough to embrace multidimensional discrete field (voxel) objects represented as "hybrid volumes" [AKPS00] that can also be treated as primitives.

Many operations that have been formulated for FRep in such a manner that they in turn yield continuous real-valued functions as their output [PASS95, SP98] can be generalised to produce more specific hypervolume operations. Of course, there can be introduced a much more application-specific operations over attributes that can hardly be sensibly applied to the geometry.

2.3 Case studies

In the previous section, we defined a new mathematical model for point sets with attributes. Herein, we consider different applications of the proposed model. First, our interest will be turned towards the definition of simple heterogeneous objects, then to a geological example, and finally show how it can be applied to a physical simulation and to an adaptive mesh generation [2].

In order to visualise a constructive hypervolume object, we choose to map the attribute to different colours. Taking into account that this mapping, and thus the texturing process, requires often non-trivial mathematical skills and specialist knowledge, the next chapter is dedicated to its study.

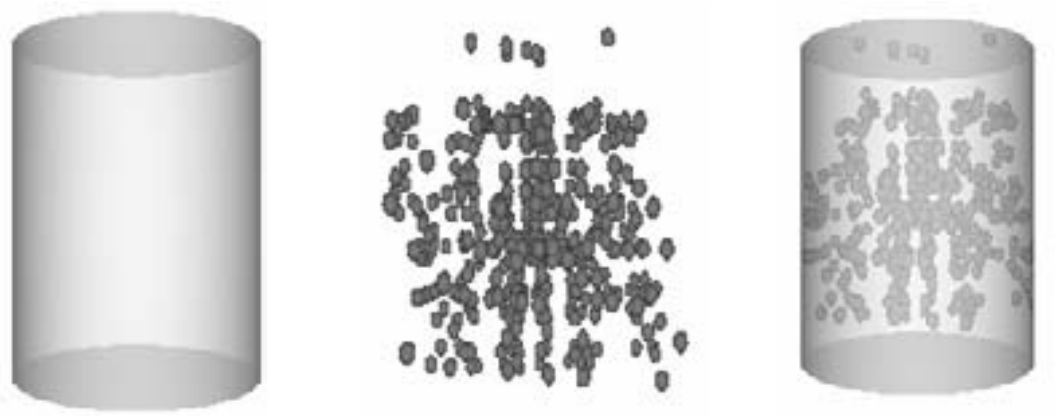
2.3.1 Heterogeneous material

Heterogeneous objects are omnipresent around us. We consider two simple examples here that are a direct application of the proposed constructive hypervolume model.

Composite materials are widely used in the industry. They are composed of several elementary elements, whose association provides properties that each single element does not have. Usually, such materials can be decomposed in two parts, the matrix and the reinforcement material. The reinforcement material confers the skeleton to the composite material, and the matrix the envelope. In Fig. 2.1, we propose an example of composite material, and focus on a single attribute A, the material density. There are two steps in making this model: description of the geometry and description of the attribute. The geometry, i.e., the matrix, is defined as a cylinder $F(X)$ shown in Fig. 2.1*a*. The reinforcement material is defined as microspheres, corresponding to a function F_s that defines their location in space. It is defined as a FRep tree with several spheres in the leaves and set-theoretic unions in the nodes. A constant density corresponds to each material.

To visualise the resulting object, the density value is mapped to a greyscale colour. Then, for every given point X, a first tree traversing procedure is processed on the geometrical tree

[2] This work has been published in [AKK+02], and realised in collaboration with the different authors. The example is made by E. Kartasheva.



(a) (b) (c)

Figure 2.1: Density of a composite material using greyscale. (a) The cylinder corresponds to the matrix, with a constant density. This is the geometrical tree. (b) Reinforcement material composed of microspheres with constant density. (c) Visualisation of the density of the composite material.

$F(X)$. When $F(x) \geq 0$, another tree traversing procedure is processed on $F_s(X)$ to determine the density value. In the case where $F_s(X)$ is positive, the resulting density is the density of the microspheres, otherwise, the density of the cylinder is returned. The resulting composite material is shown in Fig 2.1*c*.

The second example shows another heterogeneous object, where a sheathed electric cable is considered. The sheath is made of plastic and three different cables are embedded inside. One of them has the same orientation as the sheath, and the two other round it up. Each cable is composed of a gainer made by different plastics too, and inside of copper. The three cables are then surrounded with a twisted pair, made by another material. We propose to model this object with the proposed model. The geometry is defined as a single cylinder, using a function F_{geom}, and the attributes a represented by a material index vector. The constructive hypervolume object is defined as:

$$o = (F_{geom}, A) \tag{2.13}$$

One needs then to define the spatial occupation of each material. Different constructive trees are built for this purpose, i.e., three for the different kind of plastics corresponding to the material indices A_1, A_2 and A_3, one for the copper index A_4, and one for the material of the twisted pair, corresponding to the index A_5. The material of the embedding sheath is the default attribute, and does not require an additional tree. Figure 2.2 shows these trees.

In Fig. 2.2*a*, the tree corresponding to the index A_1 is shown. It is used to define the area where the plastic surrounding the central cable is located. The visual representation of the tree is shown on the right side of the tree, and is defined as an intersection of a cylinder C_{o_1} with another of smaller diameter C_{i_1} . Figure 2.2*b* shows the tree for the index A_2 with a front and a side view of its representation. It is similar to the tree for A_1, with an additional twisting operation. Corresponding cylinders are C_{o_2} and C_{o_3}. The tree for the index A_3 is not shown as it is similar to the one for A_2, simply mirrored along an axis. The tree for the copper index A_4

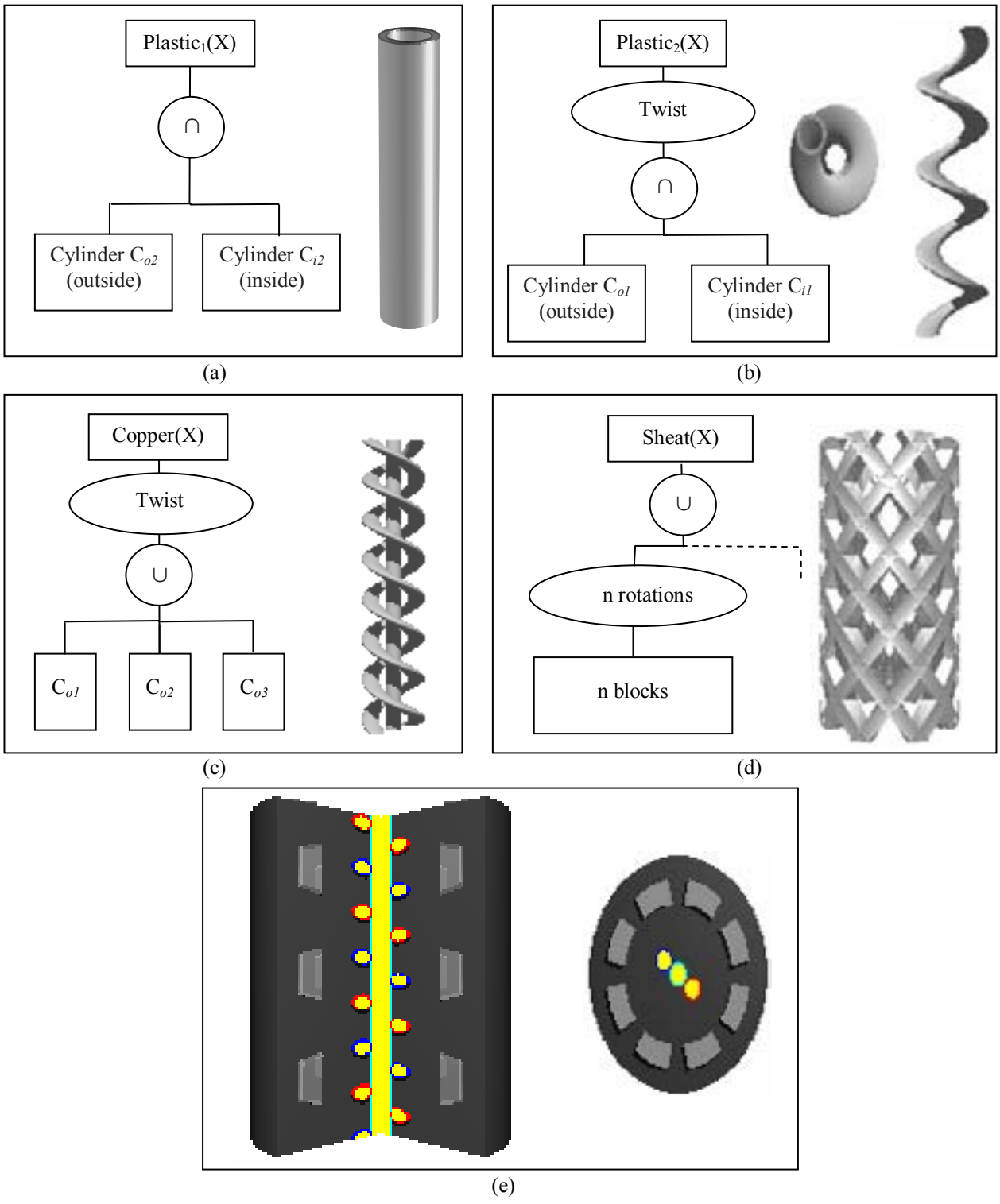


Figure 2.2: An electric cable with metallic wire. The geometry is defined as a cylinder. Constructive trees to determine the location of each material, corresponding to the indices A_1 to A_5 are shown respectively in (a) to (d). Each material is mapped to a colour. The heterogeneous object is shown in (e), with a cut according to a quadrant (left), and a top view (right).

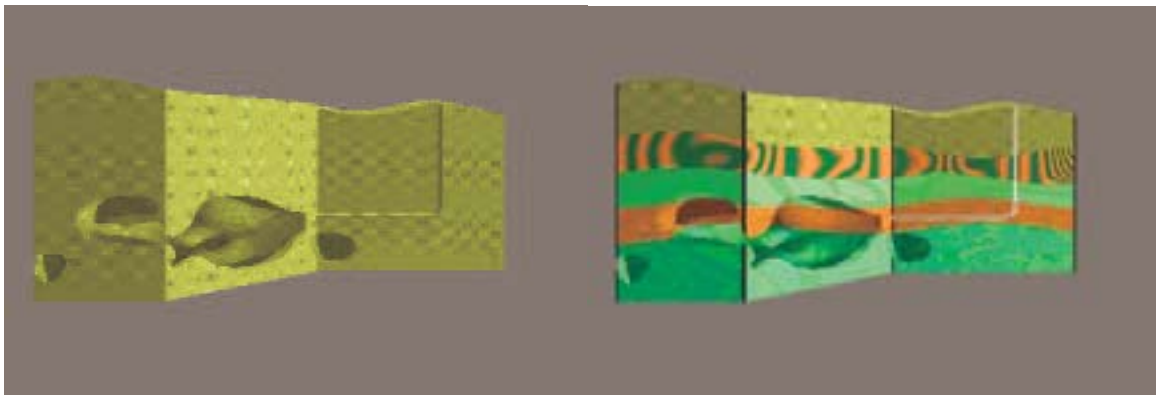


Figure 2.3: Geological structure. A geometric model (left) without attributes, (right) with attributes.

is shown in Fig. 2.2*c*, and defined as a union of the three cylinders (of smaller radii) C_{i_1}, C_{i_2} and C_{i_3}. The last index A_5 for the twisted pair is partially illustrated by Fig. 2.2*d*, and is defined as a union of several rotated blocks.

In order to visualise the different components of the cable, each attribute is mapped to a colour vector. The copper attribute is mapped to a yellow colour, each plastic attribute around the copper is mapped to red, blue, or a cyan colour, and the twisted pair material is mapped to light grey colour. The default attribute for the sheath is mapped to dark grey. Then, to determine the colour at the given point, when $F_{geom} \geq 0$, the tree traversing procedures are applied to each attribute tree. Note that this definition is possible as long as for a given point X, it does not belong to more than one attribute tree. Similar to the reality, a point can not belong to more than one attribute tree, i.e., single material, either plastic or metal in this example, is assigned to each point. Figure 2.2*e* shows different views of the cable. On the right side a top view, and on the left side, a side view of model, where a quadrant has been removed for the visualisation purposes.

2.3.2 Modelling geological structure

Heterogeneous objects in geo-sciences usually consist of multiple layers of different materials with cavities, wells, and other irregularities. We present here a simplified example of a functional model of such a geological object. The basic geometric model shown in Fig. 2.3(left) is described by a single function $F_{geom}(X) \geq 0$, where X is a vector of 3D point coordinates :

$$F_{geom} = (F_{relief} \& F_{bbox} \& (-F_{cavity}) \& (F_{cut})) \,|F_{well} \tag{2.14}$$

F_{relief} defines a solid bounded by the top curvilinear surface and the bottom plane, F_{bbox} is a function defining a bounding box for the model, F_{cavity} is a model of cavities made using an algebraic sum between the functions of an ellipsoid and solid noise, F_{cut} serves for producing a zigzag cut of the full object, and F_{well} represents a gas well modelled as union of two cylinders and a toroidal segment. The symbol & stands for the R-function defining set-theoretic intersection between two functionally defined solids, and | stands for set-theoretic union.

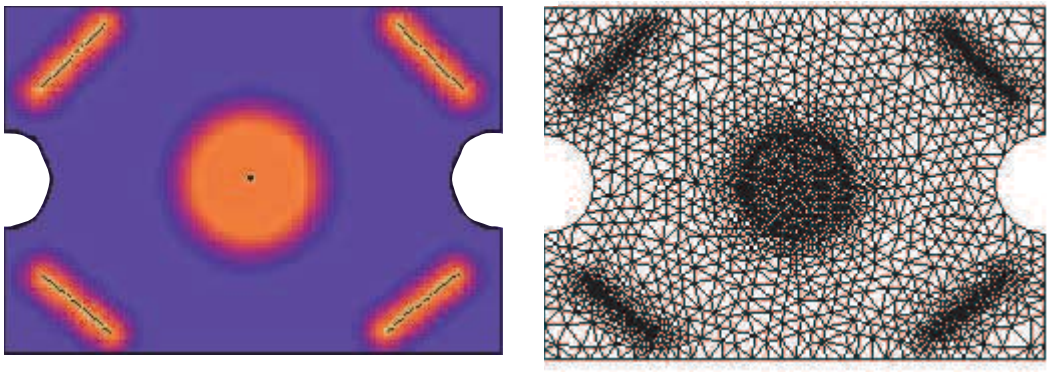


Figure 2.4: Adaptive mesh generation for heat transfer simulation. (Left) Distribution of the mesh density attribute values. (Right) Adaptive mesh.

For the model of the "relief" solid, we used the following expression:

$$F_{relief} = (f_{relief}(x, y) - z)\&z \tag{2.15}$$

where $z = f_{relief}(x, y)$ defines the top curvilinear surface of the object, and z value of the bottom plane is zero.

The five material layers shown in Fig. 2.3(right) using different colours are presented in the attribute model by the space partition different from the basic geometric model. For the i-th layer, the defining function is

$$F_i = (f_{i+1}(x, y) - z)\&(-f_i(x, y) + z) \tag{2.16}$$

where $i = 1 \ldots 4$, $f_5 = f_{relief}$, and $z = f_i(x, y)$ defines the top surface of the layer. In the simplest case of the homogeneous material distribution inside the layer, the single material attribute can be defined as

$$A = \begin{cases} M_i & F_{geom} \geq 0, F_i \geq 0 \\ \theta & F_{geom} < 0 \end{cases} \tag{2.17}$$

where M_i is a material index, and θ stands for the undefined value. The priority to the material index on the surfaces separating the layers is given by the procedure of scanning the indices for checking the condition $F_i \geq 0$. The complete functional hypervolume model is defined as $o = (F_{geom}, A)$. In fact, the visual appearance of the layers in Fig. 2.3 is achieved not by attaching surface textures, but is automatically obtained by volume rendering of the hypervolume with the set of attributes assigned to each point instead of the material index (more details about the textures can be found in the next chapter).

2.3.3 Adaptive mesh generation

We present here examples concerning adaptive mesh generation to illustrate the treatment of attributes other than material distributions. From the theoretical point of view, the presented application is based on the conversion between a functionally defined 2D object with an attribute

to a triangular mesh. The next step is made to a one dimension higher problem, where the object attribute and the resulting mesh becomes time-dependent.

Unstructured grids have found widespread use in computational physics, visualisation, and data interpolation. An accurate numerical solution requires the domain being discretised sufficiently to describe the geometry and physics of the problem. The geometrical compatibility can be achieved by automatic mesh refinement in regions of high curvatures of the boundary surface or contour. The physical compatibility dictates a close correlation between the size and shape of mesh cells and the behaviour of the solution that is sought. A varying density of mesh elements can be used to model complex problems with high accuracy. The appropriate element size distribution may be prescribed by the user if he has knowledge of the physical situation a priori. But it is often the case that the detailed information is not available prior to the numerical simulation. In these cases, the element size distribution is provided by a mesh size prediction algorithm in the adaptive analysis procedure. The information about the desired element size and shape is used by mesh generation algorithms.

It may be necessary to describe an appropriate element size not only on the boundary but also in some spatial regions inside the domain (e.g., heat-source, oblique shock in supersonic flow, etc.). One of the techniques commonly used for these purposes is based on so-called *sources* [Loh97]. The element size for a location X in the domain is given as a function of the closest distance to source $\rho(X)$. Typically, a small element size is desired close to the source, and a large element size is more preferable far from it. Moreover, the element size should be, in many cases, constant (and small) in the vicinity $\rho < \rho_0$ of the source. Power, exponential or polynomial functions of ρ [Loh97] are usually used to specify the proper element size. Given a set of m sources, the minimum element size computed for each of them is taken whenever an element is to be generated. For moving bodies (solid, liquid, gaseous), the points defining the relevant sources may be synchronised in their movement with the movement of the respective body. This allows for high quality remeshing for non-stationary problems.

Let us consider the application of hypervolumes with a functional model of attributes for the description of mesh elements density. We assume that an attribute $S(X)$ is defined everywhere in the space $\Re^n$ and an arbitrary value $S_j = S(X_j)$ is interpreted as the desired mesh element size at the point X_i. Geometry of the sources is specified by FRep. Then, we can use a value of the function describing the source at a point X as a measure of the closest distance from X to the source, $\rho(X) = \rho(F(X))$. For example, the shape of a point-source may have the following functional definition: $F(x) = a - |X - X_0| \geq 0$, where X_0 is the center of the source, a is its radius, $|X - X_0|$ is the distance between the points X and X_0. Here, we use normalized $F(X)$ so $\rho(X) = F(X)$. An analogous functional description can be created for sources of different shapes. The element size attribute A_i generated by the i^{th} source is defined in the following way:

$$M_{A_i} = (\Re, S_i) \tag{2.18}$$

where S_i is a function defined as :

$$\begin{aligned} &S_i : E^2 \rightarrow \Re \\ &S_i(X) = \begin{cases} h_{min} & \text{if} \quad F_i(X) \geq 0 \\ min(h_{max}, \frac{F_i(X) \times (k_i - 1) + h_m in}{k_i}) & \text{if} \quad F_i(X) < 0 \end{cases} \end{aligned} \tag{2.19}$$

This formula provides the geometrical progression law of the increasing element size. Here h_{min}, and h_{max} are the minimal and maximal admissible sizes of the elements and k_i is the coefficient of the progression ($k_i \geq 1$), $F_i(X)$ is the functional description of the i^{th} source. The overall element size distribution attribute A depending on all m sources is calculated as follows:

$$M_A = (\Re, S) \tag{2.20}$$

where S is defined as :

$$\begin{array}{l} S : E^2 \rightarrow \Re \\ S(X) = min(S_1(X), S_2(X), \ldots, S_m(X)) \end{array} \tag{2.21}$$

with $X \in \Re^2$. Figure 2.4 illustrates the example concerning the heat transfer problem. In this example, it is necessary to generate an adaptive mesh in the two-dimensional object, described by FRep as follows:

$$F(X) = F_b \& (-F_{o_1}(X)) \& (-F_{o_2}(X)) \tag{2.22}$$

where F_b describes the rectangle, F_{o1} and F_{o_2} define left and right holes, and the symbol & stands for the R-function defining set-theoretic intersection (see previous subsection). Taking into account the problem's conditions, we have introduced five mesh density sources (one of the point and four of the segment types) which coincided with the heat sources. The sources' locations are shown in Fig. 2.4(*left*) with colour distribution visualizing the corresponding distribution of the element size attribute values. The element size attribute was used for automatic mesh generation based on the advancing front technique. The final mesh is shown in Fig. 2.4(*right*).

2.4 Implementation

The case studies section has shown that several applications of the proposed constructive hypervolume model can be found. Depending on the nature of the attributes, different applications are used. Therefore, one needs to define a language that enables the definition of attributes in an abstract manner, regardless of its nature, and can then be plugged in a more specific application.

For different reasons that will be explained below, we propose to use the HyperFun language [ACF+99]. A general presentation of HyperFun is given, introducing the different features of the language, as well as its extension to the constructive hypervolume model. A small tutorial and examples of HyperFun are given in the Appendix.

2.4.1 Language for hypervolume modelling

HyperFun [ACF+99] 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, it supports all main notions of FRep. The current version of the language that is publicly available [Pro] only allows for the description of geometry. Here, we introduce a new version that allows us 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 coordinates x_i and numerical parameters a_i whose values are to be passed from outside the object. Each object is defined by a function describing its geometry (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 allows for increasing flexibility while dealing with attributes: values of scalar functions s_i can not only be defined and calculated within the HyperFun object definition but can be passed from the outside the object to be modified within the program describing the object definition.

The functions defined in HyperFun are actually a symbolic embodiment of the corresponding trees whose structure reflects constructive logic of building both the object's geometry and its attributes. Not only primitives (that can be library functions and local variables defined by algebraic expressions with an appropriate semantic), but other objects can also be the leaves of the tree. At the language level, this means that references to objects that have already been specified can be present in functional expressions. The functions describing geometry and attributes can be built in a step by step manner using assignment statements with introducing local variables and arrays. Conditional selection ('if-then-else') and iterative ('while-loop') structures are also available. Functional expressions are built using conventional arithmetic and relational operators by utilising 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 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 customised 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 [Bli82], soft object [WMW86], metaballs [NHK$^+$85]), and convolution objects [MS98] with skeletons of different types (points, line segments, arcs, triangles, curve, and mesh). Primitives derived from parametric functions (Bézier objects [SPS99]) have also been included into the library. As to the 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.

2.4.2 HyperFun software tools

Application software deals with HyperFun models through using either a built-in interpreter or HyperFun-to-C/HyperFun-to-Java compilers and utilities of the HyperFun API. The latter way concerned with intermediate generation of C/Java code insures more efficient function evaluation but is much more demanding for developers of application software in a multi-platform environment. All case studies presented in this paper have been developed with a help of software

tools with a built-in interpreter.

The HyperFun interpreter has been implemented as a small set of functions in ANSI C. It is quite easy to integrate them into the application software since the developer needs to deal with only two C-functions. The 'Parse' function performs syntax analysis in accordance with the language grammar and semantic rules. For each object described in the HyperFun program, the function generates an internal representation that is actually a collection of the tree structures optimised 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') is called every time when there is a need to evaluate the function at a given point in the modelling space and for the given external numerical parameters. Externally defined values for attribute scalar functions can be passed too. The object's internal representation serves as an input parameter for 'Calc' function that returns both the value of the "geometric" function and a set of values for "attribute" scalar functions - all evaluated at the given point.

The formal specification of the internal representation and of the function evaluation procedure was given in [PSAS93]. Note, that the function 'Parse' is invoked just once while 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 easy to use.

Software tools for HyperFun creation and processing are being developed in an open source project manner by an international team of developers. Some of them are currently available for free download at the Web site [Pro]: HyperFun Polygoniser for the surface mesh generation with VRML output and HyperFun plug-in to POVRay [Pov] and Vlib [Win], which makes it possible to generate high quality photorealistic images on an ordinary PC. A plug-in for Maya [May] is also under development.

Conceptually, we strive to separate the modelling in multidimensional space with abstract coordinate variables $x_1, \ldots, x_n$ from the subsequent interpretation of the model in "real world" terms (that can be, in particular, a visualisation). The concept of multimedia types [ACF+99] is exploited here. A special mapping with giving each coordinate an interpretation has been. For instance, 'x', 'y', 'z' types can correspond to Cartesian coordinates; 't' - to "dynamic" coordinate representing continuous values that can be linearly or non-linearly mapped onto physical time; 'u' and 'v' - to 2D "spreadsheet" coordinates, etc. -(the corresponding example will be given in the next chapter).

HyperFun tools have special features allowing users to implement this mapping procedure. By 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. Such a technology allows us to introduce "generic" objects with subsequent generation of their different instances. For example, the same attribute can be treated (without any change in the HyperFun program) as colour, 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.

Chapter 3

Constructive hypervolume texturing

3.1 Introduction

In the previous chapter, we proposed a new mathematical framework for modelling point sets with attributes. Simple examples of heterogeneous objects were given, as well as applications of the proposed model in various areas, namely in geology and physics. We briefly mentioned that in order to visualise the resulting objects, one has to define an additional mapping, from the attribute value to, for instance, a colour space.

In this chapter, we will focus on the colour attributes, and, in a more general way, photometric attributes[1]. The abstract attributes A_i define the photometric properties of an object, namely the texture properties. Here, the term "texture" is used in accordance with the following definition: *anything that is evaluated at a point using only information local to that point is a texture* [Gla95].

The examples of this chapter are modelled using the HyperFun language, and rendered using different tools, namely the HyperFun polygonaliser [Pro, PPP88], PovRay [Pov], and Vlib [Win]. The first one polygonises the iso-surface of an object and uses hardware rendering, the second one is a direct surface rendering, and the third one is a direct volume rendering. Depending on the tool being used, the attributes A_i correspond to different shading parameters such as colour, transparency, ambient, diffuse, and specular reflection, and the reflectance property.

After recalling in the next section the existing texturing techniques in general, we define a new technique for texturing objects, based on the constructive hypervolume framework. The proposed method consists in defining a space partition using constructive trees. In each subset, different attributes are defined. Several examples are given to illustrate the proposed approach while describing it. Then, descriptions of some particularities of the trees for the attributes are given. Extensions to objects of higher dimension (4D (time-dependent) and 6D cases) are also considered. The last section deals with two specific attributes that require a special interest as they can not be defined as photometric or shading parameters: the first one corresponds to bump mapping, and the second is used to speed up the rendering process.

[1] This work has been published in [SPAS01]

3.2 Existing texturing techniques

Several texturing techniques exist in computer graphics to embellish geometric objects. 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 [BN76], or on a fractal subdivision [HB84]. In the book [Ea98], 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 [Cat74] was the first solution proposed allowing applying some 2D colour patterns (digital images or procedural function) to a parametric surface. Many other works in this area followed and extended the application of a colour to some other shading parameters [BN76], or to the modification of the surface properties such as the normal vector perturbations, i.e., bump mapping [Bli78]. 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 [Ped85, SS96], to skeleton-based models [TW99], both with a special parameterisation process, or even a completely different approach, such as the one in [PRA00], for instance, where the idea is to cover an arbitrary surface using overlapping copies of a texture patch, and to define some local parameterisations ("lapped texture").

The previously exposed texture mapping is widely used, but this solution requires a parameterisation of the objects surface, which is often a non-trivial problem, and encounters some problems, like distortions. A powerful approach is proposed in [NC99], where a local parameterisation of the objects surface is proposed, and avoids several problems of traditional texture mapping. Other texturing methods exist, like cellular texturing, which is applicable to any kind of surfaces, but is restricted to a certain kind of texture. Another solution for texturing implicitly surface is defined in [ZGVdF97], where the gradient field of the implicit surface is used to define a particle system and then an appropriate mapping.

An alternative is the concept of solid texturing initially introduced in [Pea85, Per85]. 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 parameterisation. 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 [WGG99] 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 chapter, 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.

3.3 Constructive solid texturing approach

Within this section, we define a new texturing approach by applying constructive hypervolume modelling in the 3D case. Attributes are considered as the photometric properties of an object. Our approach can be considered as an extension of the solid texture concept, introduced in the previous section, and we call it *constructive solid texturing*.

When applying solid texturing to an object, one has to create a partition of the object space, where each subset contains different material property. The constructive solid texturing approach proposes a robust method for creating such a space partition, which involves additional trees for the attributes. Those trees are called in the following indifferently either *constructive texturing trees* or *constructive attribute trees*.

An introductory example is first given followed by the general formulation of constructive solid texturing, and then some specific features of the constructive texturing tree are described.

3.3.1 Forestalling example

We propose in this subsection to apply a simple texture pattern to a constructive hypervolume object, where its geometry is shown in Fig. 3.1*a*. The point set is defined as $F(X) \geq 0$, where $F(X)$ is real-valued function, in which an ellipsoid, a torus and a soft object are used for the leaves, and the union operations stand in the nodes. Attributes are the colour of this object, equivalent to a simple red and blue 3D checker-board pattern. To define such pattern, one can use a constructive tree composed of blocks in the leaves and set-theoretic unions in the nodes, shown in Fig. 3.1*b*. Let us call this tree F_A. To define the colour at the given point, one can state that if the point belongs to a block, then its corresponding colour is red, and otherwise blue. It is equivalent to say that a partition of the object space has been done while defining this tree. Figure 3.1*c* shows the initial object placed inside the obtained space partition and the resulting textured object is shown in Fig. 3.1*d*.

To visualise the object, the following scheme is followed. A tree traversing procedure first evaluates F using the constructive tree that defines the geometry. If F is greater or equal to zero, then another tree traversing procedure is evaluates F_A to determine which colour is defined at this point, and the result is either red if $F_A \geq 0$, or blue otherwise.

A special attention has to be paid when one says “the colour of this point is red”. In this example, when the tree traversing is applied to the constructive texturing F_A, the colour attribute can be defined in two different manners. Either it corresponds to an index, that will be used later to define the appropriate colour, or it implicitly defines a set of three attributes

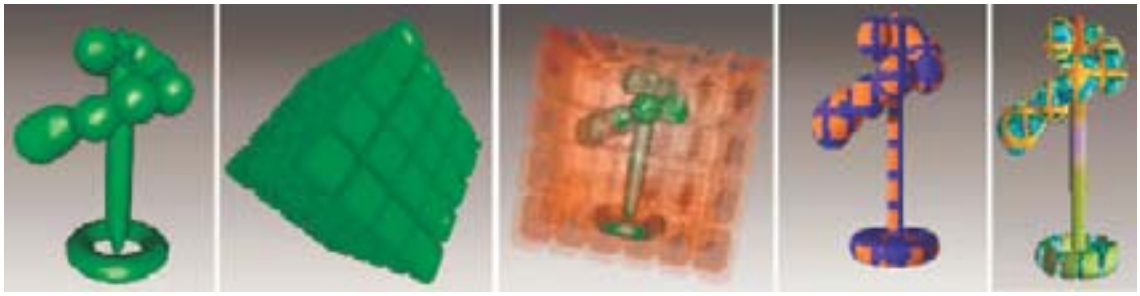


Figure 3.1: Constructive solid texturing. Forestalling example of space partition using a constructive texturing tree: (a) FRep model to be textured; (b) Space partition; (c) Initial object placed in the texture space; (d) Result of texturing. (e) The same FRep model textured using another space partition.

corresponding to an RGB colour vector. In the first case, it means that the same constructive texturing tree is defined for the three colour attributes, and in the second case, three trees are defined for each component of the colour, and thus requires three tree traversing procedures to evaluate the colour of a given point. Figure 3.1*e* illustrates this case, where different trees for each attribute are introduced.

3.3.2 Constructive solid texturing definition

We propose to define a robust method for creating a space partition into several subsets, where different photometric attributes are defined for each subset. A subset is defined as a solid object and has to provide for each point an answer to the question: "Is this point inside or outside the subset?" In the affirmative case, the corresponding solid texture function is applied. One powerful solution for point membership classification against a 3D solid is to use the FRep model. The name we propose for this texturing method is *constructive solid texturing*, to emphasise the constructive approach inducted by the FRep model.

The constructive texturing tree has somewhat different meaning if one compares it with a FRep constructive tree. In both cases, they are used to evaluate a real-valued function obtained by 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 the defining function of the space partitioning solid is positive at the given point (i.e., the point belongs to this solid), then, one can evaluate an attribute function. Thus, we add the operator "if" as a node to the constructive texturing tree, as "jumps" between colours are allowed (consider a simple checker-board pattern for instance to be convinced).

While considering a constructive hypervolume object, the geometry of the object can be defined in different manners. In the previous example, it was defined as a real-valued function. The application of the proposed texturing method can be also applied to other types of representations. For instance, if the object is defined as a polygonal mesh, then the tree traversing procedure of the function defining the geometry is replaced. For each given vertex of the object,

tree traversing procedures are applied only to the constructive texturing trees of attributes. The same modification in the texturing process can be done for BRep, voxel and other models of object geometry.

According to the general description of the constructive solid texturing given above, we propose to formally define it, in a specific "FRep context, as follows. Given a constructive hypervolume object $o = (G, A_1, \ldots, A_n) : (F(X), S_1(X), \ldots, S_n(X))$, given a partition of the object space for each attribute, defined as a set of n defining functions F_i, and given two sets of default attribute values θ_0 and θ_1, the i^{th} attribute is evaluated according to the following procedure :

$$A_i : \begin{cases} S_i(X) & \text{if} \quad F(X) \geq 0 \quad \text{and} \quad F_i(X) \geq 0 \\ \theta_{1_i}(X) & \text{if} \quad F(X) \geq 0 \quad \text{and} \quad F_i(X) < 0 \\ \theta_{0_i}(X) & \text{if} \quad F(X) < 0 \quad \text{and} \quad F_i(X) < 0 \end{cases} \tag{3.1}$$

where S_i can be any real-valued function, such as a constant function, a noisy function, or, a more complex procedural function such as the ones proposed in [Pea85, Per85] or in the book [Ea98]. The existence of the two sets of default attributes $\theta 1_1$ and $\theta 1_2$ are justified as follows. Given an object, one defines a partition of the object using different FRep trees. When a point belongs to the object, nothing prevents from the case where for this point, all the defining functions of the attribute space partition are negative. The first default set of values θ_1 is then defined. Another set of default values θ_0 needs to be also defined. This second set is deeply connected with the rendering engine used to visualise the object. Indeed, if the volume rendering engine Vlib is used, the definition of the constructive hypervolume has to be expressed in such a manner that it fits to the CVG model definition. As it was explained in the survey section of the Chapter 1, in CVG, the geometry of an object is defined by its opacity field. To fulfil this requirement, an attribute of the constructive hypervolume object is then dedicated to this. For a given point, when it belongs to the point set, any value can be given. When this point is outside the point set, the opacity attribute is then set to full transparency. This condition is sufficient to express a constructive hypervolume object in the CVG model.

3.3.3 Complex object space partitions

In this subsection, we provide more complex examples illustrating the application of the proposed texturing techniques. Photometric attributes, and not only RGB colours, are evaluated for each given point according to the given definition.

Complex space partitioning can be obtained using a constructive FRep tree, as it is shown in Fig. 3.4 and in Fig. 3.5. Figure 3.2 shows the simplified constructive geometric and texturing trees for the example of Fig. 3.4. 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. 3.2(*Left*)). The attribute tree is more complex. Nodes of the geometrical tree (marked in blue) are combined with some other primitives. For instance, a union operation is applied to the bottom ellipsoid and to four spheres, as shown in Fig. 3.3. 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 ellipsoid and the union of spheres. Then, different shading parameters are applied to each sub-point set. The colours in Fig. 3.4 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, using an algebraic sum. Then, some attributes are assigned to each noisy block. Finally, an additional partition is defined as a union between the water and a fish (Bézier volume), each with its own attributes. Because the space partition for texturing is defined with a constructive tree, we call this method constructive solid texturing.

More complex space partitions can be obtained using the FRep constructive tree as it is shown in Fig. 3.5. The initial object in Fig. 3.5*b* is also an FRep object (a model of the real sculpture "Naked" by Russian artist I. Seleznev shown in Fig. 3.5*a*) mainly composed of blending unions between convolution surfaces. The space partition has been defined by the union of four swept spirals (Fig. 3.5*c*). Different shading parameters are assigned to each spiral.

The usage of FRep trees for modelling and texturing is illustrated in Fig. 3.6. The presented objects have been defined using several different primitives. The texturing tree used for the left object is the same as the geometric one, and is mainly composed of unions of ellipsoids and B-spline primitives with different shading parameters assigned to each primitive. To define the texturing tree for the right object, some parts from the geometrical tree were used, such as tori, blocks, and cones, but it also contains parts of another origin. For instance, the nails on the box are defined as a set of cylinders regularly placed on the box, and are only defined in the texturing tree.

3.3.4 Operations on attributes

This subsection gives a general overview of possible operations on attributes. As it will be shown, a unified framework is very difficult to define, and additional research would be certainly required to explore all the different possibilities.

When one builds a constructive tree to model a geometric solid, the use of set-theoretic and other operations such as blending is needed. If one considers only the geometry of the object, the result is unambiguous. But, if one is considers the attributes of the object, the use and the result of such operations need further discussions. Let us consider a simple solid, defined as the union of three horizontal blocks and three vertical blocks as shown in Fig. 3.7. A single definition of the union is used for the geometric FRep model, but several definitions are possible for the attributes. In Fig. 3.7, different textures are applied to each block (simple colour for the block 1; fully opaque checker-board pattern for the block 2, with noisy colours for the blocks, and grey colour 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 of 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 the proposed definition of procedure for the attribute evaluation, we introduced the "if" statement. In some case, one may prefer to obtain a blending of attributes between the partitions, rather than jumps. Then, the following technique can be applied. Any defining function F of a

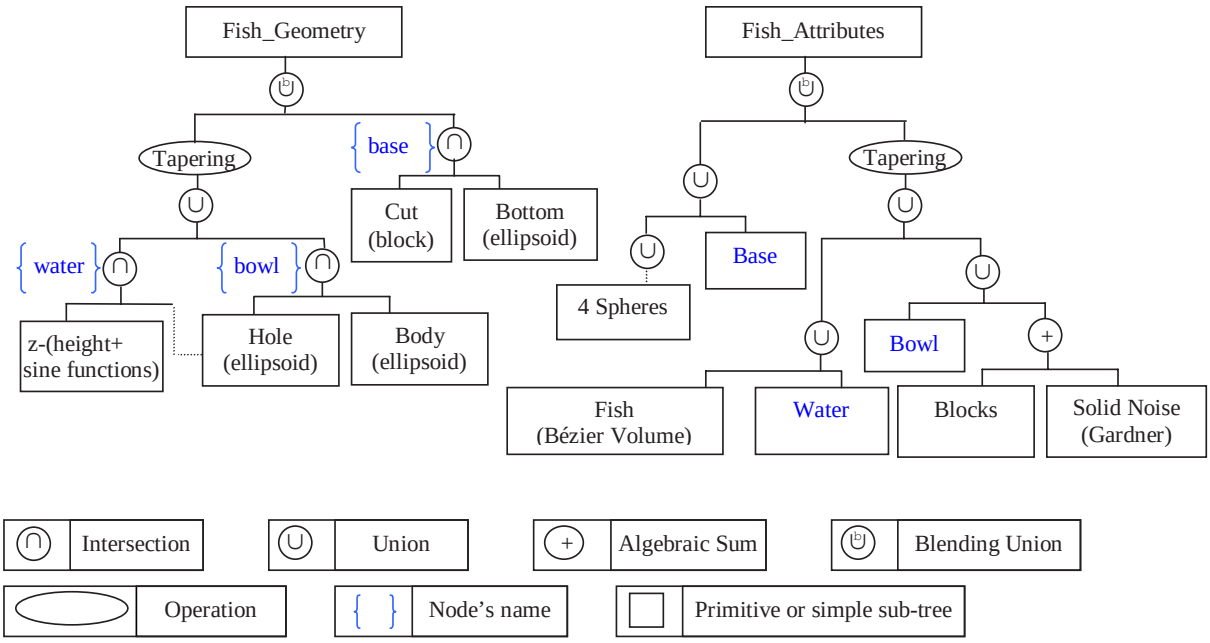


Figure 3.2: Constructive trees of Fig. 3.4. (Left) Geometrical tree. (Right) Texturing tree, composed of internal nodes of the geometrical tree, and of some other primitives. Different attributes are set to each node.

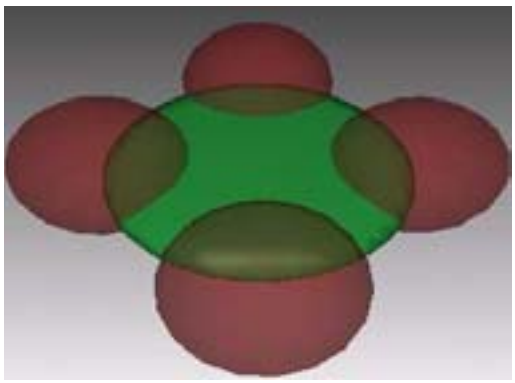


Figure 3.3: Visual representation of a part of the constructive texturing tree of Fig. 3.2. The geometrical object (an ellipsoid) is green. The attributes tree is defined as a union of four spheres (red) and of the ellipsoid used for the geometry. One set of attributes is defined for points of the ellipsoid inside the spheres, and another set for outside.

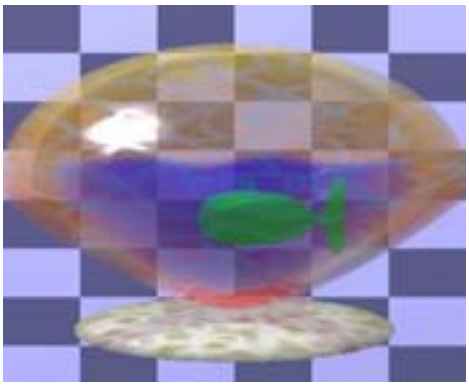


Figure 3.4: Example of constructive solid texturing. The geometrical model is defined as a blending union of an ellipsoid and a sphere. The water, the fish, and the different mosaic patterns are defined using a tree composed of spheres, Bézier volume, blocks and solid noise. Details of the tree can be found in Fig. 3.2

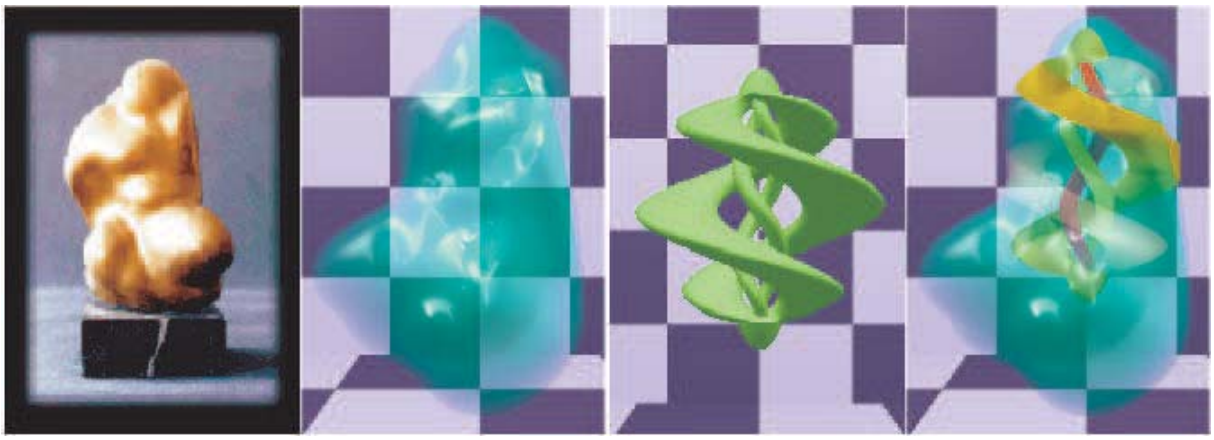


Figure 3.5: Constructive solid texturing: (a) Original sculpture (photo); (b) 3D FRep model of the sculpture; (c) Constructive 3D solid representing the space partitioning for texturing; (d) Textured shape.

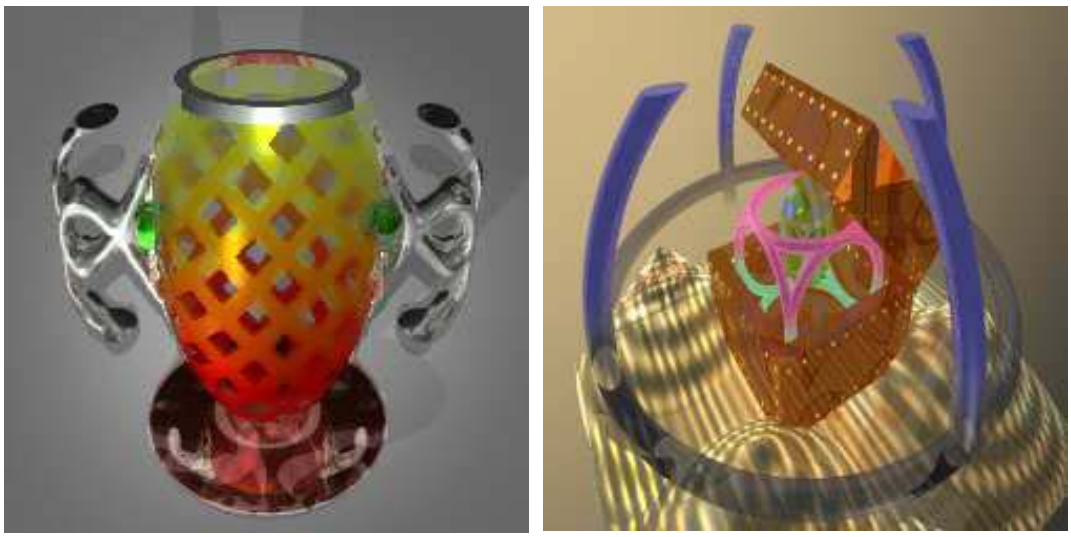


Figure 3.6: Application of the constructive solid texturing to FRep objects, using 13 different shading parameters: (left) The tree used for the space partition is similar to the constructive tree for the object's geometry; (right) Texturing and geometric trees are different.

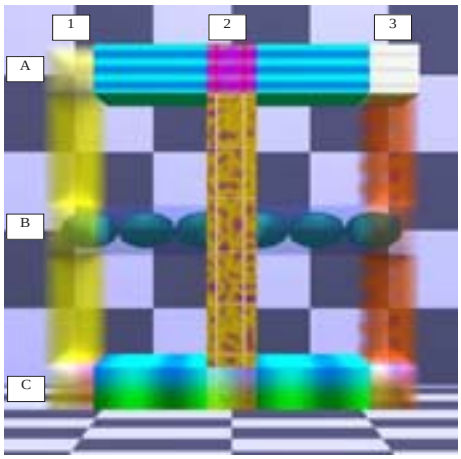


Figure 3.7: Examples of different union operations. Six different blocks, and nine different unions. For instance, the union A1 gives priority to the attributes of the block 1; the attributes of the union A2 are defined as the difference of the attributes of blocks A and 2, and as the sum of attributes for the union A3.

FRep object can be mapped to the interval $[0, 1]$ using the following function:

$$\tilde{F} = \frac{1}{2}\left(1 + \frac{F}{\sqrt{p + F^2}}\right) \tag{3.2}$$

where the parameter p controls the fall of the function. Let us consider an attribute A_i, defined by a function S_i, and evaluated when its corresponding partition F_i is greater or equal to zero. Let us also consider a default attribute function $\theta(X)$. Then, the resulting attribute A generated by the blend can be defined as:

$$A = \tilde{F}_i(X) \times S_i(X) + (1 - \tilde{F}_{theta}(X)) \times \theta(X) \tag{3.3}$$

Figure 3.8 shows an example of application of this formula. The geometric object is a block, and the space partition for attributes includes two torii, F_1 and F_2. The default attribute is an RGB colour vector defined according to some noise function, and the attribute for the torii is a red and blue checker-board pattern. To calculate the functions $\tilde{F}_1$ and $\tilde{F}_2$, two different values for the parameter p have been used, respectively 0.1 for the torus at the bottom of Fig. 3.8 (*right*), and 5 for the top one. Depending on the value of p, the result of the blending operations for the attributes is different. When p is small, no blend occurs and as p increases, as the blend is significant. The resulting colour is a combination of the torus colour attributes and the default attributes, i.e., outside the torii. As one can see, the "green noisy texture pattern of the block appears in the blue area defined by the torus.

This definition for the blend is global, i.e., applied to the whole partition. Other definitions can be found while considering existing blending functions, such as the one proposed in [Ric73b, PS94, MK85] for global blends, and [PPIK02] for bounded blends.

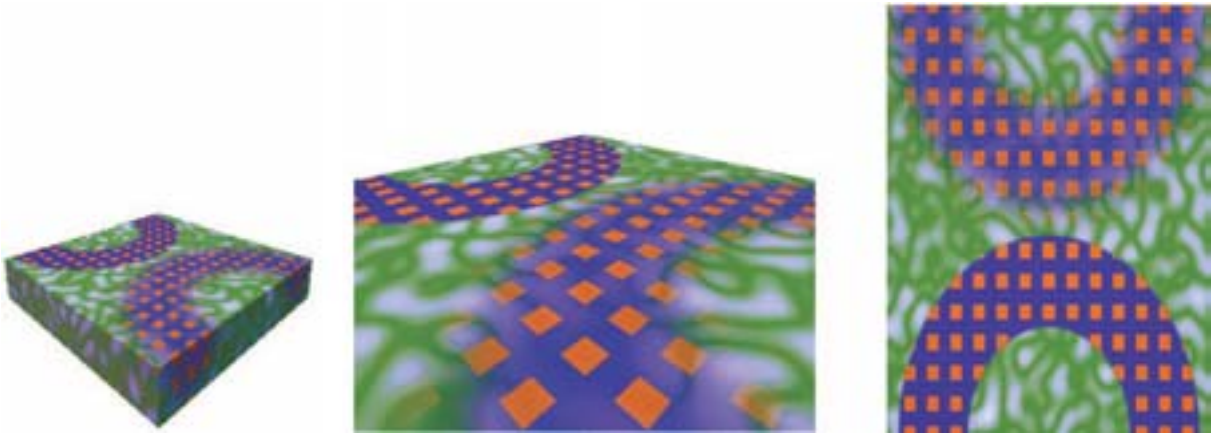
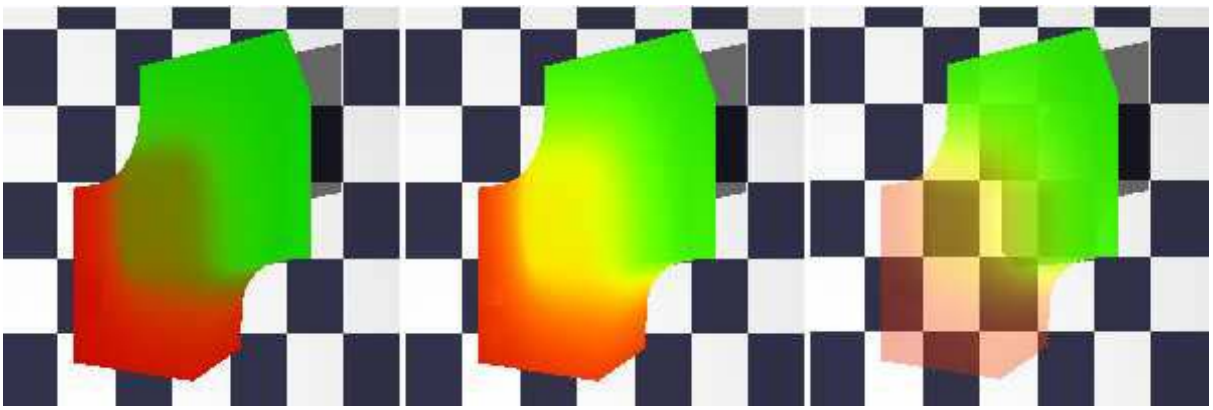


Figure 3.8: Blending of attributes. A block object, shown from different points of view, is textured. Different parameters for mapping the defining functions S_1 and S_2 of the space partition to $\tilde{S}_1$ and $\tilde{S}_2$ are used, to illustrate their influence in the blending of attributes.



(a) (b) (c)

Figure 3.9: Blending operation between two cubes. (a) Blending is applied to the three components of the RGB colour vector. (b) Blending is applied to the Hue component of an HLS colour vector. (c) Blending function is also applied to the opacity attribute.

In the proposed example, the colours were defined in the RGB space. It appears that the number of different definitions increases if one considers another colour space, such as HLS, HSV, XYZ, La*b* and others. Figure 3.9 illustrates this consideration, where a geometric blending union operation is applied to two blocks. Another blending function has also been used for the colour attributes. Let the colour of one of the blocks be bright green, and the colour of another be bright red. Obviously, the most expected resulting colour of the blended material should be bright yellow. It is apparent that colour mixing in the RGB colour space does not produce the expected colour. In Fig. 3.9*a*, the resulting colour in the RGB space is a dark yellow. In fact, because the brightness component is not explicitly defined in the RGB colour space, it is difficult to obtain bright colours by colour mixing. Thus, we have to change from one colour space to another, HLS in this example, and it then becomes easier to obtain only bright colours. In Fig. 3.9*b*, a blending operation has been applied only to the Hue component, whereas the Lightness and Saturation components remain the same. The obtained result is a bright colour for the entire solid including the blocks and the added material. In Fig. 3.9*c*, we add an opacity component. The red cube is transparent, and the green one is completely opaque. The same blending function is applied, and as one can see there is a smooth transition of attributes from one object to another.

The proposed examples are presented here just to demonstrate on the diversity of available ways to define the resulting texture. They lead to the important fact that a unique solution does not exist, and, from the user's point of view, a tool using the union of textures should allow enough freedom in selecting its definition. Examples were produced using the HyperFun language. In the Appendix, some HyperFun models are presented to show some of the possibilities offered by this language.

3.4 Constructive solid texturing in higher dimension

3.4.1 Constructive time-dependent texturing

There is a long-standing interest in time-dependent texturing. It was addressed in the concept of the "shade tree" model [Coo84] and later used in a scene-graph based rendering environment [Ups90]. The extension of our method to time-dependent texturing is straightforward and is described below.

When one creates a 4D model, e.g., with time as an additional coordinate, 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-related techniques. The similarity with the way these trees are created is significant for our framework. It implies that each transformation that occurs in the geometric tree can also occur and is supported in the attribute tree.

Figure 3.10 presents an example. The initial object (Fig. 3.10*a*) includes different textures based (from top to bottom respectively) on trigonometric functions, gradient function along an axis, and a hand-defined pattern function (similar to the checker-board). Different deformations are applied to this object. The problem with the standard definition of solid textures is that the space partitioning is done during a separate step than the modelling one. The application of a transformation of the object's geometry may be difficult to apply to the space partition, simply

because they are defined in a different manner. In Fig. 3.10*b*, a twisting operation is applied. The patterns follow the twist defined as time-dependent transformation. The same operation is applied to, and supported by both constructive trees, the geometric and attributes ones. Figure 3.10*c* shows a 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 is time-dependent. While these operations are applied, both the textures and the space partition change. As it can be seen in Fig. 3.10*c*, some of the grid stripes at the bottom of the textured model become yellow according to a sine function with a time-dependent period. At the same time interval, the texture based on trigonometric functions also changes according to some other time-dependent mappings. The important advantage of constructive solid texturing is that the space partition can also change in time. As it can be seen in Fig. 3.10*d*, two blocks (one with a blue colour pattern, and the other with a yellow) were added at the corners of the original model to influence only the constructive solid texturing tree at a certain time step. Transformations of the FRep object geometry also occur in the constructive texturing tree, and the visual result is the transformation of all texture patterns.

Another example is given in Fig. 3.11 to illustrate the possibility to create time-dependent space partition and to texture both the interior and the surface of the object. The geometric object here is a volumetric head. We applied a voxel-to-function conversion in order to use this head as a FRep object. The separation of space is modelled as an algebraic sum of two functions, one defining a planar halfspace and another one defining Gardner's solid noise. This sum results in the deformation of the planar halfspace and in the generation of many separate components near its boundary, thus simulating an amorphous or a liquid substance. The planar boundary of the halfspace is translated along the x-axis during some time interval. Figure 3.11*a* corresponds to the initial step, where the texture for the voxel head is defined as a combination of different values of green colour and transparency, and Fig. 3.11*d* corresponds to the last step, where a simple grey colour applied to the entire object. Figure 3.11*b* shows the middle step in the animation. In one half space, the green texture is applied, and in the other half space, the grey texture. Figure 3.11*c* shows the space partition, where only one side of the head is shown. As one can see, a complex time-dependent space partition can be obtained.

So, the two previous examples show that arbitrary time-dependent transformations can be applied to both geometry and attributes, either in the uniform manner or in completely different ways.

3.4.2 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 iso-surface as an elementary object in the cells of the animated spreadsheet (Fig. 3.12) as illustrated in Fig. 3.13. The elementary shape illustrates function dependence on three variables x[1], x[2], and x[3]. Changes of iso-surfaces 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].

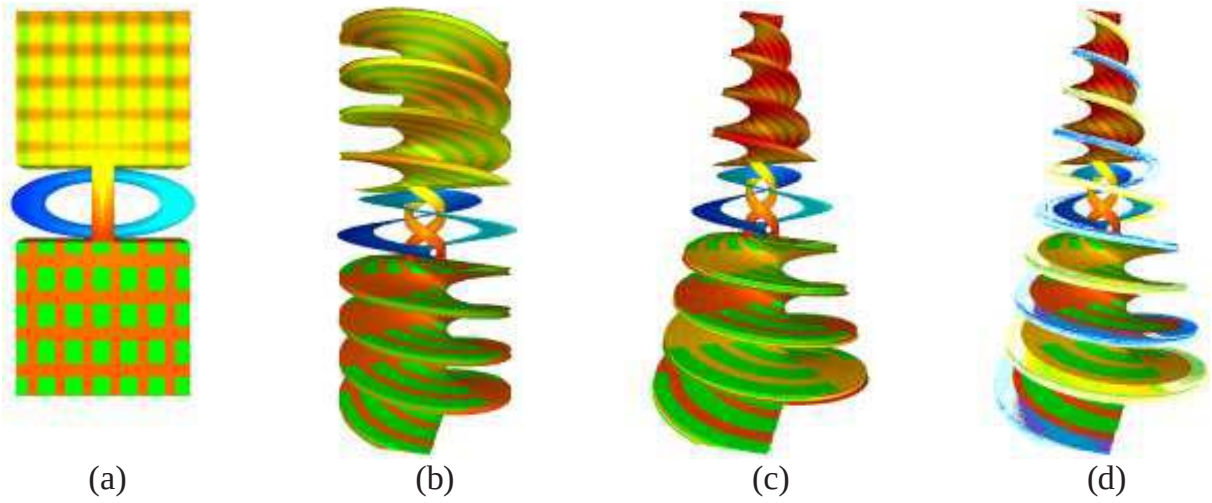


Figure 3.10: Time-dependent texture. (a) The original patterned model.(b) Twist operation applied to both constructive trees, geometric and texturing one. (c and d) Time-dependent twisting, tapering and texture pattern.

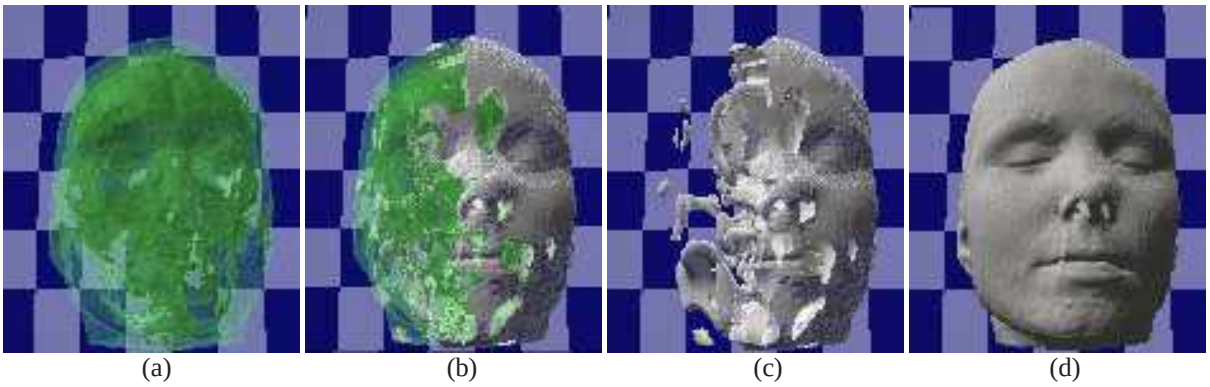


Figure 3.11: Time-dependent space partition and texturing. A plane geometrically deformed by solid noise is translated from left to right (from (a) to (d)): (b) Middle frame of the animation. (c) The same time step as (b) with only one half of the head is shown to illustrate the complex space partition inside the head.

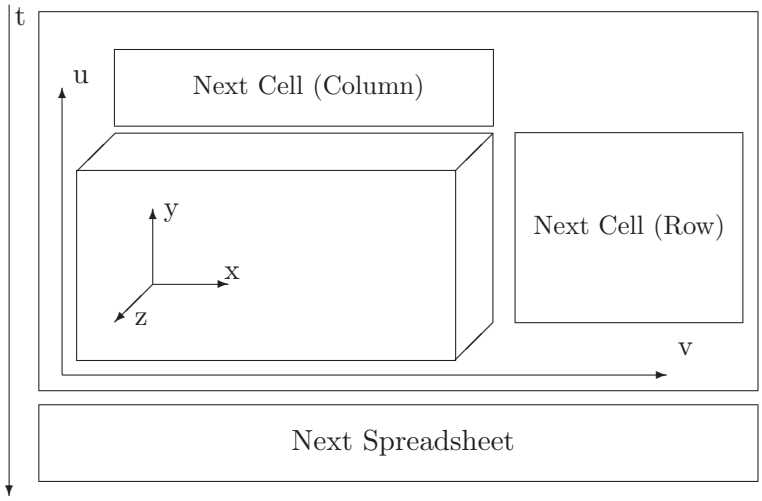


Figure 3.12: Animated spreadsheet concept: a spreadsheet has (u, v) coordinates. Each (u, v) pair corresponds to a cell containing a 3D object with (x, y, z) coordinates. Spreadsheet changes in time with the t coordinate.

Formally, the following types are assigned to the geometric coordinates:

(3.4)

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

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. 3.12.

We used this function as a basis to show that the way the constructive solid texturing method was defined is independent of the model's dimensionality. Figure 3.13 shows colouring 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 union of 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 the visualisation of the dependence between variables becomes easier and more graphical.

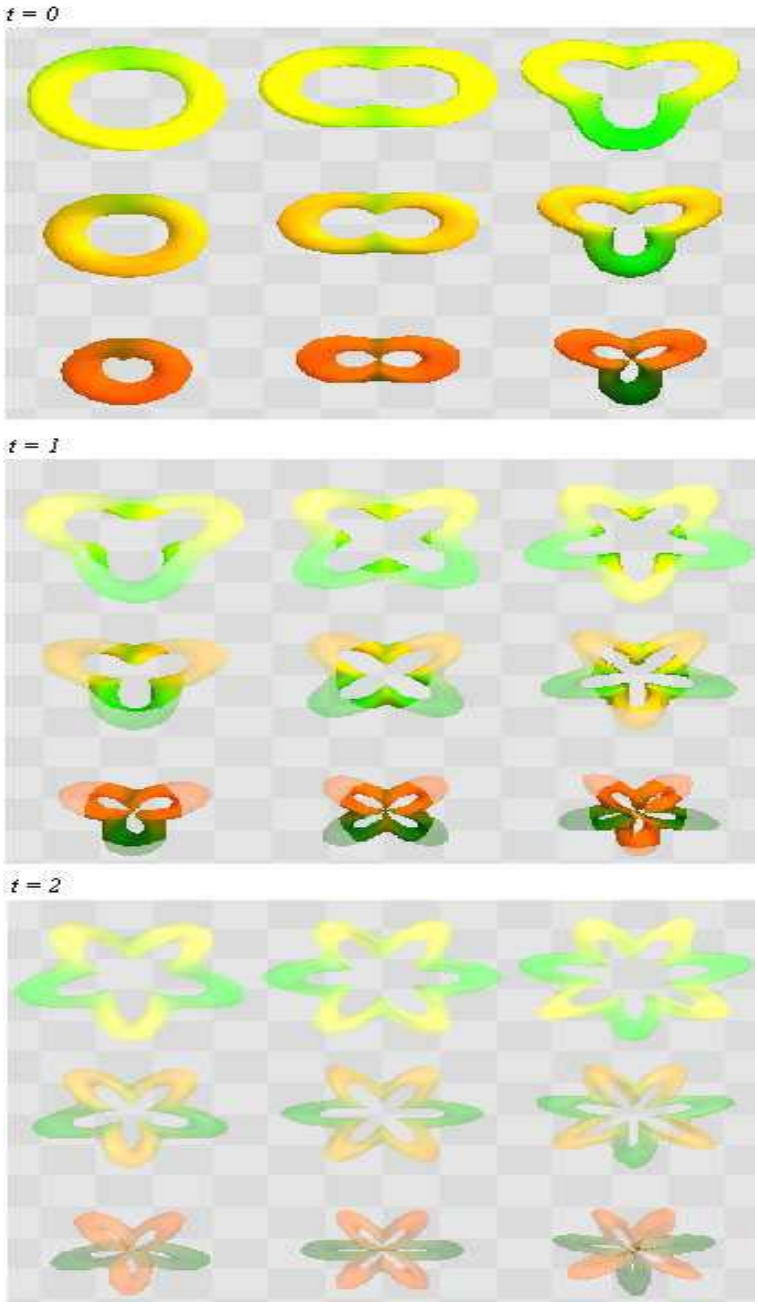


Figure 3.13: Animated spreadsheet of a constructive hypervolume 6D point set with 13 photometric attributes : three frames of animation for three time steps $t = 0, 1, 2$.

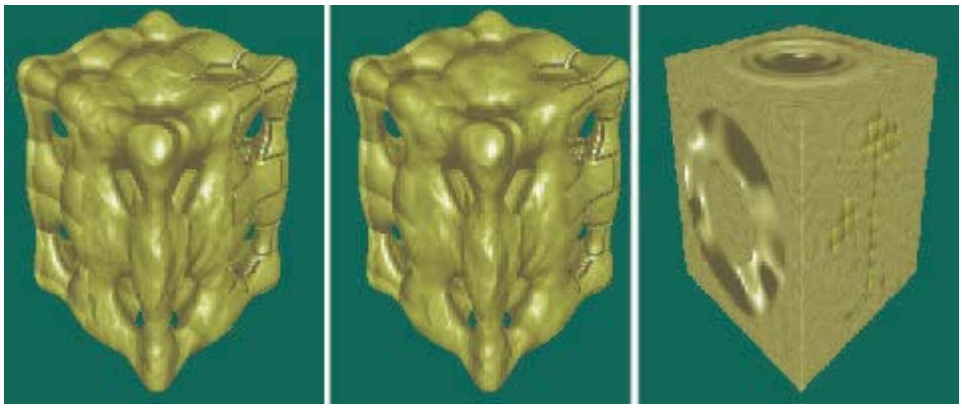


Figure 3.14: Bump attribute. From left to right. The geometric object is defined using Chebichevs polynomial of degree 6. The space partition is equivalent to the four quadrants. Different bump mapping are defined, with no blending and then with blending of attributes. The most right pictures shows that different bump mapping can be defined in arbitrary space partition.

3.5 Special attributes

This section considers two non-photometric attributes that have deep connections with texturing and visual enhancement of an object. A space partition is still used to define real-valued attributes. In the first subsection, the real value is used to apply a bump mapping on the surface of the object. The second subsection proposes to use a special attribute in an adaptive rendering context.

3.5.1 Bump mapping

The bump mapping [Bli78] is a visual "trick" enabling a surface to appear either bumpy or wrinkly. In contrast with the displacement mapping [Coo84] where the object's surface is effectively deformed, the bump mapping alters the normal of the surface according to some perturbation functions. While rendering the object, this change in the normal orientation produces a change in the local illumination model, and simulated details appear on the surface of the object. The modified normal N' is defined as :

$$N' = N + D \tag{3.5}$$

where N is the original, and D the perturbation vector. As a matter of fact, the normal deformations are generated by the perturbation vector D. In [Bli78, Ea98], details of the calculation of D as well as several examples are given.

It appears that the constructive approach to define space partitions can be applied in this case. A perturbation function defining the bumps corresponds to each partition. Figure 3.14 gives some examples of bump mapping. In the left and middle pictures, the geometry is based on Chebychevs polynomial of degree 6. The space partition consists of the four regions defined

as $(x \geq 0, y \geq 0)$, $(x < 0, y > 0)$, $(x > 0, y < 0)$ and $(x < 0, y < 0)$ (i.e., the four quadrants for short). The same normal perturbation is defined in the first and last quadrant. A brick and a quilted perturbation are defined in the remaining quadrants. On the left side, no blending operation is defined, and as one can see the perturbation "jumps" from one quadrant to another, and in the middle, a smooth blending has been applied, according to eq. 3.3. The right picture shows another example, where the object is a block. The space partition is composed of a sphere, a torus, and the object shown in Fig. 3.14*right*. A different normal perturbation is assigned to each primitive.

3.5.2 Speed-up attribute

Herein, we will consider a special attribute that has a meaning only at the rendering stage, and allows for an important speed-up while rendering constructive hypervolumes based on FRep.

The "brute force" ray-tracing algorithm throws ray in the scene and strives to find an intersection with an object. When the object is functionally defined, the intersection point coordinates satisfy the equation $F(X) = 0$, where F is the defining function of the object. To find this intersection, the ray is regularly sampled along the path. It can be written as:

$$X = u + vt \tag{3.6}$$

where u is the starting point of the ray, and v is the direction of the ray, and t is the sampling step.

We define here a simple method that uses directly FRep property to increase the speed of the rendering process. We suppose that a FRep object, its corresponding bounding box, as well as the user-defined accuracy are given.

The problem is to find $\bar{t}$ such as $F(u + v\bar{t}) = 0$. A large variety of optimisation methods exist to find in a quicker manner the solution of this equation. But all the proposed methods are dedicated to some specific cases, and can not be applied in general. It results that often a constant sampling step is used to ray-trace such an object.

Herein, we propose to use one attribute of a constructive hypervolume object to help the ray-tracing process. In some sense, it is an extension of the existing methods based on bounding volumes used while rendering implicit surfaces [JW88, WT90]. Let us consider that the aim is to visualise the iso-surface I_0 of an object, defined as $F = 0$, and consider another iso-surface I_k, defined as $F = k < 0$. The space can be then subdivided in three regions. For a given point X, one can state the following:

1. if $F(X) \leq k$, then the point X is "far" from the iso-surface I_0,
2. if $k < F(X) < 0$, then the point X is "close" to the iso-surface I_0,
3. if $F(X) \geq 0$, then the point X is on the iso-surface I_0, or inside the object.

For each sample of a ray, the function F is evaluated, and thus the attribute function value. Then, instead of considering a constant sampling step, one can define two sampling values according to the above remarks. In the first case, a greater sampling value will be used, and in the second and third case, a smaller one.

This method provides an important speed-up of the rendering process. Nevertheless, the following problem arises: "How to select the proper I_k iso-surface? ". As there are no constraints on the function F, its behaviour is difficult to predict. An inelegant solution, but reasonable, is to visualise first the object using faster rendering engines, such as a polygonalisation, or to ray-trace it while using a single ray and a simple illumination model. Even a simple visualisation of 2D iso-contours is sometimes enough to find the appropriate iso-surface. We used this solution to ray-trace a "noisy" sphere. Without this optimisation, it takes 60 seconds to render this object, whereas with the use of the speed-up attribute, it takes 10 seconds (on a 450 Mhz processor). Results are promising, and further investigations in rendering and visualisation of constructive hypervolume objects are needed.

3.6 Conclusion

In this Chapter, we proposed to apply the constructive hypervolume model to texturing objects. The point set is defined using different representations, such as FRep, BRep and voxels, and the attributes are photometric attributes, such as colour, ambient, diffuse, specular, and other. For every given attribute, we propose to define its value according to a space partition. Such partitions are defined using the FRep model, and its underlying constructive approach. To determine the attributes of a given point, the tree traversing procedures are applied. Examples of objects of different dimensions are given.

This approach for texturing objects can be extended in various directions. Combining attributes for instance is a non-trivial task, and a certain formalism may be needed. A promising direction to follow is to use an approach similar to the one proposed by Schlick in [BGS94], where some elementary nodes and meta-nodes are defined. Then, combining attributes would become a combination of those nodes. Another extension of this method concerns time-dependent objects. It would be very interesting to apply the constructive hypervolume model in physically-based animation.

Chapter 4

Constructive hypervolume sculpting

4.1 Introduction

In the previous chapters, to model an object, only the constructive approach was considered. Building a constructive tree to generate an object is usually a time-consuming and sometimes difficult process. Usually, a complex shape contains both regular parts that can be easily decomposed into a set of primitives, and some other parts more difficult to decompose. An alternative to the constructive approach is the free-form design. With this method, parts that can be hardly decomposed in elementary elements are modelled directly, using a sculpting approach. This approach can be divided in two main families. The first one consists, starting from a given object, in deforming it until the desired shape is obtained. The next chapter is dedicated to this technique. The second approach can be usually compared to a sculpting metaphor. An object is sculpted while adding or removing some material at the desired location. The term virtual sculpting was first introduced in [Par77], and several works and researches have been conducted in this direction. A very good survey on the existing techniques can be found in the PhD thesis [Fer02]. Several existing tools are dedicated to a surface manipulation only, i.e., polygonal [Par77, BL95] and parametric [FB88, GOP99].

In [GH91, WK95], an environment in which a 2D painting scheme is proposed to add or to carve an existing object, and in [AS96, FCG00] a 3D interaction is proposed using a haptic device to sculpt the object. Those methods use a discrete characteristic discrete function, resulting into a 3D grid of uniform voxels, called *voxmap*, containing in each vertex a function value. The major advantage is the constant time evaluation procedure, i.e., a trilinear interpolation, used to visualise the object after a modification. Another approach of the sculpting metaphor using real-valued functions has been proposed by Elber et al. [RE99] and by Schmitt et al. [SPS99]. In those works, the object is represented as a zero set of a trivariate B-spline [RE99] or trivariate Bézier function [MPS96, SPS99].

A valuable approach would be to combine constructive modelling and volume sculpting, because both approaches contain useful features. To combine these approaches, one has to define a primitive that can be both sculpted and included in a constructive tree [SPS01]. To add a new primitive to a FRep tree, one has to verify the function continuity property (at least C^0 continuity of the defining function everywhere in the space). The characteristic functions used

in [GH91, WK95, AS96, FCG00] are C^0 continuous as the representation is piecewise linear. In the three first works, the grid resolution is fixed making difficult the sculpting process. In the latter one, [FCG00], the sculpting process can be achieved at any level of resolution, and results in a powerful tool.

In [RE99], the sculpting area depends on the space where the parametric function is defined, i.e., it is restricted within the boundary of the parameter space of the trivariate B-spline function. Therefore, such definition makes it difficult to use the sculpted object in another context, and especially to use it as a constructive primitive. A similar approach has been proposed for trivariate Bézier functions in [MPS96, SPS99] and extended to B-spline functions in [SKPS00, SPS01]. The main difference is in the mathematical framework. The same parametric function is used to define a solid as a point set with non-negative function values. A so-called functional clipping is employed so that the function gets negative outside the parameter space. This insures the function to take positive values only where the sculptor chooses to create some material.

We propose in this chapter to model geometry and attributes of the heterogeneous object using a combination of sculpting and constructive modelling based on solid primitives defined by trivariate B-splines. We propose to combine sculpting of the trivariate B-spline primitives with FRep constructive modelling. Sculpting of the B-spline primitives [SKPS00, SPS01] is improved by a multiresolution scheme and real-time visualisation. Complex shapes modelled in this way can be then combined with traditional primitives using different high-level operations like blending or twisting.

4.2 Sculpting constructive hypervolumes

The concept of extended space mapping introduced in [SP98] is used as the underlying framework for modelling B-spline primitives and complex heterogeneous objects. A space mapping establishes a one-to-one correspondence between points of a given space and, if applied to one point set in the space, it changes this set to a different one. A mapping can be defined by the functional dependence between the new and old coordinates of points. A formal definition of the extended space mapping can be found in [SP98]. It generalises both space mapping and function mapping (i.e., manipulations done on the defining function values) by considering an extended space with geometric coordinates and the additional functional coordinate.

Let us discuss the example given in Fig. 4.1. The intention is to model a 1D point set with attributes, i.e., a segment along the x-axis. Let f be a real-valued function of one variable such as $\xi = f(x)$. The inequality $f(x) \geq 0$ defines a segment. At the same time, a curve is defined in the (x, ξ) plane. This plane is called an *extended space*, because it has a geometric coordinate and an additional function coordinate. Fig. 4.1*a* shows such a curve and its corresponding segment at some given step. The function $f(x)$ is drawn in black, and its corresponding point set in red. Different transformations can then be achieved, and are called under the general term *extended space mapping*. For instance, if a translation is applied to the curve, the extended space is mapped onto itself, and thus defines an extended space mapping. As said, extended space mapping combines two main families of mapping, namely the *function mapping* and the *space mapping*. The first one gather transformations that occur at the function level (see Fig. 4.1*b*), and the second at the coordinate level (see Fig. 4.1*c*).

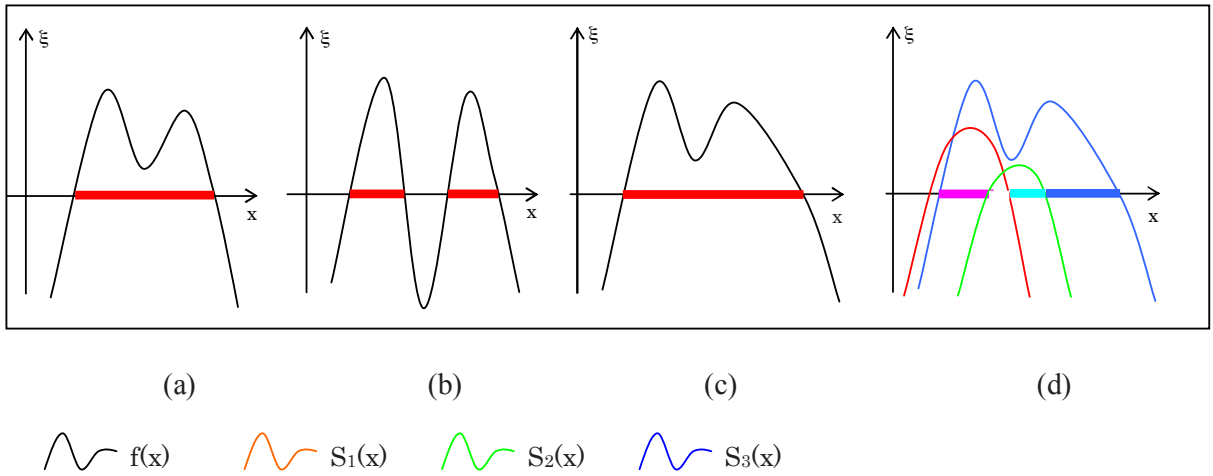


Figure 4.1: Extended space mapping. (a) 1D point set (red) is defined by a curve $\xi = f(x)$, and the projection of its positive part onto the x-axis. (b) Modification using function mapping (the function f is modified).(c) Modification using space mapping (the x coordinate is scaled non-linearly) (d) Heterogeneous object with attribute functions added to the model. To each function S_i corresponds a component of an RGB colour. For each point belonging to the 1D solid, i.e., $f(x) \geq 0$, the three functions $S_i(x)$ are evaluated, and a RGB colour is then defined.

In a similar way, extended space mappings can be used for attribute functions, as Fig. 4.1d shows, where attributes consist of red, green and blue colour values. The function $S_i(x)$ defined in the extended space corresponds to an attribute A_i. In this example, the function corresponding to the blue attribute is similar to the function used to define the point set, and the functions for the red and green attributes are arbitrarily defined. One can notice that the shown white part of the segment is obtained as the intersection of the red and green partition.

As a point set and the space partitions are defined in a similar way, it naturally comes that they can be modelled in a uniform manner. The sculpting paradigm we propose can thus be used for both of them. Then, in the remaining part of this chapter, we use the term *object* to identify without any difference either the geometry of the point set or its attributes.

To model an object using a B-spline function, only the function mapping is needed, as it will be shown in the next sections. In the next chapter, the space mapping will be used to define deformations of functionally defined objects.

4.3 Trivariate B-spline primitive

4.3.1 Framework for the primitive

To create objects, we propose to use a sculpting scheme close to the one proposed in [RE99, SKPS00], where uniform cubic trivariate B-spline functions are used. The basic definition of the B-spline is similar, but as it will be shown in the next subsections, additional properties are required in order to be able to provide a new primitive for a FRep constructive tree.

Let the defining function f be a trivariate cubic uniform B-spline function defined in a parametric space by a set of $l \times m \times n$ scalar coefficients λ_{ijk}, called control coefficients or control points:

$$f(u,v,w) = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} N_i(u)N_j(v)N_k(w)\lambda_{ijk} \tag{4.1}$$

where $N(t)$ are the cubic B-spline basis functions [Far90] (or blending functions) defined over a uniform knot vector, and u, v and w belong to the parametric space $[0,1]$.

The control points λ_{ijk} are regularly placed in the space to insure that along each axis the following equalities are verified (using the property of the cubic B-spline basis functions) :

$$\begin{aligned} f^x(u,v,w) &= u \\ f^y(u,v,w) &= v \\ f^z(u,v,w) &= w \end{aligned} \tag{4.2}$$

The tensor product used for the B-spline definition will be applied only to the ξ coordinate of each control point. The resulting 3D point set will be defined as :

$$G = \{(x,y,z)/f^{\xi}(x,y,z) \geq 0\} \tag{4.3}$$

One can recall the example given in Fig. 4.1, and consider the curve S in the extended space as a traditional B-spline curve. Then, to model a 1D object, one has to use a B-spline curve defined by a set of 2D control points. By analogy, to model a 3D object, the use of 4D control points is required. While the first three coordinates are used to locate a control point in the space (i.e., the usual xyz coordinates), the fourth coordinate ξ contains the scalar coefficient. By changing this value, different shapes can be obtained, and thus different function mappings are defined. To visualise the object, we polygonise the iso-surface defined as $f(x,y,z) = 0$ using an algorithm based on hyperbolic arcs [PASS95]. More details about our implementation can be found in the following sections.

4.3.2 Providing distance property

Motivation

The proposed definition allows one to model complex free-form objects, but they cannot yet be combined with other primitives using FRep. Indeed, if the B-spline primitive is used alone, the behaviour of the defining function outside the parametric domain does not matter. The space coordinates can be easily mapped to the parametric one. But as the aim is to use the new primitive in a FRep constructive tree, it has to respect the FRep definition, that is to be positive only in the domain of interest, and negative everywhere else. As trivariate B-spline objects are parametrically defined, one has to insure that outside the domain of interest, the function remains not only negative, but also decreasing. To be convinced of the importance of both properties, one can first consider for instance a blending union, such as the one defined in [PASS95]:

$$blend(f,g) = f + g + \sqrt{f^2 + g^2} + disp(f,g) \tag{4.4}$$

where f and g are two functions and $disp(f, g)$ is the function of the added material:

$$disp(f, g) = \frac{a_1}{1 + \left(\frac{f}{a_2}\right)^2 + \left(\frac{g}{a_3}\right)^2} \tag{4.5}$$

The parameters a_1, a_2 and a_3 correspond respectively to the material added symmetrically to f, g, and asymmetrically to f and to g. If the aim to apply a blending union of two B-spline primitives, corresponding to the functions f an g, it is clear that if they remain constant outside the parametric space, the result of the blending operation will not be the one expected. Indeed, below a given value (depending on a_1, a_2 and a_3), the blending operation is similar to a union operation, and above this value, the resulting function will be positive for every given points.

no material will be added, and above this value, the added material will fill the whole parametric space.

Functional clipping definition

To insure that the B-spline primitive remains negative and decreasing outside the parametric space, we use the functional clipping defined in [SPS99]. We can force the trivariate B-spline function, or more generally any functions, to become negative outside a certain domain, i.e., the parametric space in our case. This space can be considered as a unit cube with the use of some simple scaling operations. Then, the B-spline function has to be negative outside this cube.

The functional clipping is defined as follows. Let & be the intersection operation defined with the R-function as:

$$f(x) \,\&\, g(x) = f(x) + g(x) - \sqrt{f^2(x) + g^2(x)} \tag{4.6}$$

and let ω be the defining function of the unit strip of one variable:

$$\omega(t) = t(1 - t) \tag{4.7}$$

The defining function of the unit cube can be expressed as follows:

$$\Omega(u, v, w) = \omega(u) \,\&\, \omega(v) \,\&\, \omega(w), \tag{4.8}$$

The inequality $\Omega(u, v, w) \geq 0$ defines a closed subset, which corresponds to the parametric space.

The functional clipping can now be defined as the intersection of this subset with the trivariate B-spline function :

$$F_{clip}(u, v, w) = F(u, v, w) \,\&\, \Omega(u, v, w), \tag{4.9}$$

The application of the functional clipping insures that outside the parameter space, i.e., the unit cube, the trivariate B-spline function will remain negative. Furthermore, the functional clipping provides a distance property of the trivariate B-spline function outside the domain, but does not change the solid primitive.

Initialization and functional clipping application

Under some circumstances, the function value of the B-spline can remain constant between the iso-surface and the boundary of the parametric space. As required in the motivation subsection, the first step is to force the function to be decreasing inside the parametric space. A simple solution is to initialise the ξ value of the control points with real values, according to a Gaussian-type function, where the highest negative values are set on the boundary of the grid of control points, and values increase until zero as the control points are getting closer to the centre of the grid. To insure that the modelling process will not be affected by this initialisation, the real values used for initialisation are small enough.

The next step is to force the B-spline function to be negative outside the parametric space by applying the proposed functional clipping. If it is applied exactly on the boundary of the parametric space, it can produce C^0 discontinuities. This is due to the definition of the B-spline basis functions, where the first and last basis functions (i.e., N_0 and N_k in eq. 4.1, where k is the number of control points) drop to zero on the boundary and remain constant outside the parametric space.

Figure 4.2 is given as illustration of the problem, in the 1D case. A cubic B-spline curve (in blue) is used to define a 1D solid (in red). The parametric space is defined between the first and the last control point. In Fig 4.2(*top*), no functional clipping is applied, and the B-spline function remains equal to zero outside the parametric space. In Fig 4.2(*middle*), the proposed functional clipping is applied exactly on the boundary. As one can see, a discontinuity is generated on one side of the parametric space. One solution is to a ghost point at each side of the control point set, shown in green in Fig 4.2(*bottom*). A new parametric space corresponds to this new B-spline curve, which can be called augmented parametric space. Then, instead of applying the functional clipping exactly on the boundary of this new space, it is applied inside, on the boundary of the previous one, and as one can see, no discontinuity is generated.

The extension to the 3D case is straightforward. A grid of $l \times m \times n$ is extended to a grid of $(1+l+1) \times (1+m+1) \times (1+n+1)$ control points. The functional clipping is then applied in the sub-space of the augmented parametric space defined between the control points λ_{111} and $\lambda_{(l+1)(m+1)(n+1)}$ of the new B-spline function. As the control points are regularly placed, their exact locations are easy to calculate. Figure 4.3 shows an example in the 3D case. The model, Fig 4.3*a*, is defined by a FRep constructive tree. The primitives in the leaves are two spheres and a trivariate cubic B-spline function (central star-shape). In the nodes of the tree, different operations are used to combine these primitives. A blending union operation is applied between the spline and a sphere, and the resulting object is then combined with a second sphere with a union operation. Figure 4.3*b* is a zoom on one of the boundary of the parametric space, and illustrates the problem generated by the discontinuity: some materials generated by the blending union have appeared on the boundary, exactly where the B-spline function drops to zero. Figure 4.3*c* shows the result after adding the ghost points.

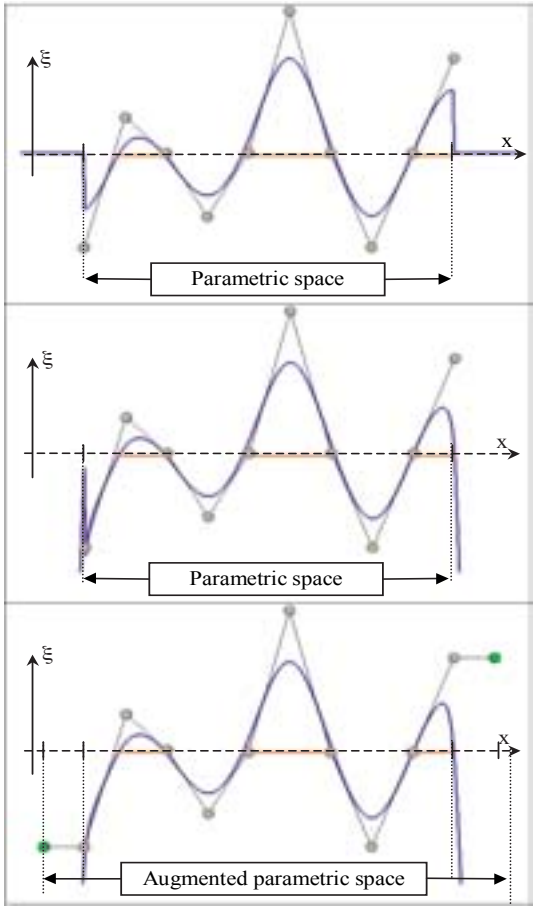
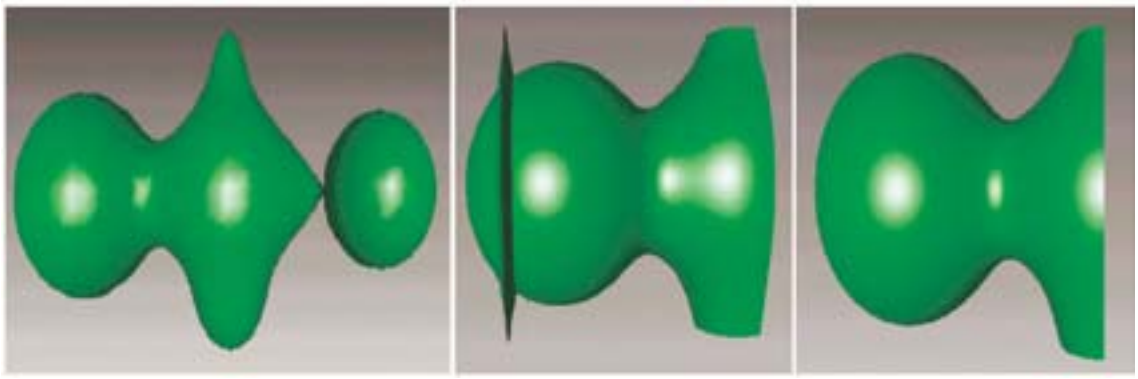


Figure 4.2: Discontinuity generated by the functional clipping, 1D case. A set of 2D control points (grey spheres), and its corresponding cubic B-spline curve (in blue), are used to define a 1D solid (red segments). (Top) Outside the parametric space, the function remains equal to zero. (Middle) Application of the functional clipping on the boundary of the parametric space. A discontinuity appears. (Bottom) After adding two ghost points (green spheres), and applying the functional clipping on the correct region, i.e., equal to the previous parametric space, the discontinuity disappears.



(a) (b) (c)

Figure 4.3: Discontinuity generated by the functional clipping, 3D case. (a) An FRep object composed of two spheres and a B-spline primitive (star shape) combined using the union (right) and the blending union (left) operations. (b) While applying a blending union between a B-spline primitive and a sphere, an undesirable part appears.(c) After adding the ghost points, the discontinuity disappears.

4.4 Multiresolution approach for the trivariate B-spline primitive

Herein, an approach is proposed to facilitate modelling using the cubic B-spline primitive. While such a primitive is defined using a set of control points, we propose to use multiresolution analysis based on cubic B-spline wavelets. The aim of the multiresolution scheme is to provide a possibility to change the number of control points of the spline during modelling in order to achieve local or global modifications. Methods to increase the number of them can be easily found in books on splines [Far90]. Nevertheless, decreasing the number of control points requires a different approach. Indeed, if one starts to model an object using a certain set of control points, and then in order to apply a global modification, decreases the number of them, he/she should be able to obtain a new set of control points similar to the original one, which includes the original state and the global modification. The main guideline is the preservation of the information contained in the set of control points.

A method for non-uniform B-spline functions was proposed in [EG01]. In the case of a curve (defined in the Euclidean space, and not in the extended space), to decrease the number of control points, about half of them are removed, and the resulting curve is then defined by the best least squares approximation for the original curve. "Details" are defined by the difference between the original curve and its approximation. The main drawback of this method is the critical need for large amount of memory. Storing details requires the same memory size as storing the curve itself. This memory requirement becomes especially significant when one considers a B-spline function defined with a set of, for instance, $256 \times 256 \times 256$ control points.

This problem can be solved while using a multiresolution analysis based on the wavelet transform. Indeed, the inherent properties of this transformation provide a powerful solution to increase or decrease the number of control points, with constant memory size. In the first

subsection, we provide a short overview of the mathematical framework of a multiresolution analysis, its application to a B-spline curve and surface, and then extend it in order to apply it to the trivariate B-spline primitive. Examples of modelling using the proposed multiresolution scheme are then given.

4.4.1 Multiresolution analysis

A multiresolution analysis of $L_2(\Re)$ is a sequence of nested spaces V^i, such that the union of all the V^i is dense in $L_2(\Re)$. For each V^i, we define a column matrix of basis functions:

$$\Phi^i(u) = \begin{bmatrix} \varphi^i_1(u) \\ \vdots \\ \varphi^i_{m_i}(u) \end{bmatrix} \tag{4.10}$$

where the set $\{\varphi^i_k\}_{k=1..m_i}$ is a basis of V^i. Functions φ^i_k are called *scaling functions*. As V^i are nested spaces, there exists a matrix P^i such as:

$$\Phi^i = P^i \Phi^{i+1} \tag{4.11}$$

Under such conditions, the scaling functions are said to be *refinable*. The *detail spaces* W^i are defined as complements of V^i in V^{i+1}:

$$V^i \oplus W^i = V^{i+1} \tag{4.12}$$

Each space W^i can be viewed as the space containing the information needed to reconstruct a function f^{i+1} in V^{i+1} from its projection f^i on V^i. For each W^i, we define a column matrix of basis functions:

$$\Psi^i(u) = \begin{bmatrix} \psi^i_1(u) \\ \vdots \\ \psi^i_{m_{i+1}-m_i}(u) \end{bmatrix} \tag{4.13}$$

where the set $\{\psi^i_k\}_{k=1..m_{i+1}-m_i}$ is a basis of W^i. Functions ψ^i_k are called *detail functions* or *wavelets*. As $W^i \subset V^{i+1}$, there also exists a matrix Q^i such as:

$$\Psi^i = Q^i \Phi^{i+i} \tag{4.14}$$

4.4.2 Multiresolution B-spline curve

One possible application of the multiresolution analysis is to define a multiresolution curve. Let us consider f^n as a B-spline curve, where Λ^n is a vector of k_n control points, which can be written as:

$$\Lambda^n = \begin{pmatrix} \lambda^n_1 \\ \lambda^n_2 \\ \vdots \\ \lambda^n_{k_n} \end{pmatrix} \tag{4.15}$$

When uniform, cubic B-spline blending functions are good candidates for a multiresolution analysis, as they respect important requirements, such as finite support, refinability and other. A complete study for curves and cubic B-spline surfaces can be found in [DSS95] and a full study of multiresolution in general in the thesis [Gri99].

Furthermore, as it was shown in [DSS95], the cubic B-spline blending functions, in the case they are uniform, fit to the definition and to the requirements of the scaling functions (finite support, refinable). Thus, in the remaining of this subsection, the basis Φ^j will be defined as a set of cubic B-spline blending functions.

As it was introduced at the beginning of this section, the aim of a wavelet transform in the multiresolution analysis is:

1. to find an approximation Λ^{n-1} of Λ^n using a lower number of components k_{n-1}. We can say that a concept of *resolution* corresponds to each n. For instance, if one considers two representations of a function f^n, f^j and f^i, with $i < j < n$, the number of components of the vector Λ^i corresponding to f^i is lower than the one used for f^j. Thus, one can say that f^i is a coarse representation of f^n, and also that f^j contains more details than f^i and is a finer representation of f^n.

2. to store some information in order to be able to reconstruct the vector Λ^n from Λ^{n-1}. Those stored data are usually called *details*.

Practically, one needs to calculate different matrices of constants in order to be able to change the considered vector space. In other words, one can consider those changes as increasing or decreasing the number of control points. In matrix notation, the above points can be expressed as:

$$\Lambda^{n-1} = A^n \Lambda^n \tag{4.16}$$

where A^n is a $k_{n-1} \times k_n$ matrix, and

$$D^{n-1} = B^n \Lambda^n \tag{4.17}$$

Since the matrix Λ^n contains k_n components and Λ^{n-1} contains k_{n-1} components, then to reconstruct Λ^n from Λ^{n-1} and D^{n-1}, D^{n-1} has to be a vector composed of $k_n - k_{n-1}$ components. Thus, the matrix B^n is a $(k_n - k_{n-1}) \times k_n$ matrix. The calculation of Λ^{n-1} and D^{n-1} is called the decomposition step, and the matrices A^n and B^n are usually called analysis filters.

Once Λ^n is decomposed into Λ^{n-1} and D^{n-1}, one should also be able to reconstruct Λ^n from Λ^{n-1} and D^{n-1}. Thus, two other matrices are needed, and this reconstruction step will be expressed as:

$$\Lambda^n = P^n \Lambda^{n-1} + Q^n D^{n-1} \tag{4.18}$$

The matrices P^n and Q^n are called synthesis filters. Details on the calculation of A, B and P, Q can be found in [DSS95].

The extension to the case of a cubic B-spline surface [DSS95] is straightforward. To build a two dimensional wavelet basis, one has to consider all the tensor products of the one dimensional basis. Thus, the basis for the scaling functions is:

$$\phi\phi(x, y) := \phi(x)\phi(y) \tag{4.19}$$

Similarly, the wavelet functions are given as follows:

$$\begin{cases} \phi\psi(x,y) := \phi(x)\psi(y) \\ \psi\phi(x,y) := \psi(x)\phi(y) \\ \psi\psi(x,y) := \psi(x)\psi(y) \end{cases} \tag{4.20}$$

For the decomposition step, the explicit calculation corresponding to eq. 4.19 using matrices is:

$$\Lambda^{n-1} = A^n(A^n\Lambda^n)^t \tag{4.21}$$

4.4.3 Modelling using a multiresolution B-spline primitive

The application of the wavelet transform to the proposed trivariate B-spline primitive is straightforward. The only requirement is that the control points have to be regularly placed along each axis. This is insured by the inherent properties of the B-spline basis functions, and the calculated analysis and synthesis filters. Indeed, each control point is located along a local maximum of a blending B-spline function. The application of either a decomposition or a reconstruction step produces another set of control points, also located along the maximum of their corresponding blending function. Then, one can first calculate the position of each control point along the axis using only the level of details j, and then apply the wavelet transform considering only the ξ coordinates as the component of Λ^j.

The extension to the 3D case, i.e., a 4D B-spline function, is then achieved in a similar way as above with the following basis for the scaling functions:

$$\phi\phi\phi(x,y,z) := \phi(x)\phi(y)\phi(z) \tag{4.22}$$

and for the wavelets functions, the seven other tensor products are given as:

$$\begin{cases} \phi\phi\psi(x,y,z) := \phi(x)\phi(y)\psi(z) \\ \phi\psi\phi(x,y,z) := \phi(x)\psi(y)\phi(z) \\ \phi\psi\psi(x,y,z) := \phi(x)\psi(y)\psi(z) \\ \psi\phi\phi(x,y,z) := \psi(x)\phi(y)\phi(z) \\ \psi\psi\phi(x,y,z) := \psi(x)\psi(y)\phi(z) \\ \psi\phi\psi(x,y,z) := \psi(x)\phi(y)\psi(z) \\ \psi\psi\psi(x,y,z) := \psi(x)\psi(y)\psi(z) \end{cases} \tag{4.23}$$

Using this multiresolution analysis while modelling allows one to switch from one set of control points to another. The main benefit of the use of wavelets is the preservation of the details, as shown in the two following examples.

Figure 4.4 shows different steps of modelling a "monster" using the wavelets transform. First, the cubic trivariate B-spline function is defined by a grid of $5 \times 5 \times 5$ control points (Fig. 4.4*a*), corresponding to the set V^1. For better understanding, we choose to show only a cross-section of the model. Then, a single refinement (synthesis step) is applied to generate the set V^2; as it can be seen, the shape does not change (Fig. 4.4*b*). Some scalar values are then changed in the set V^2 to generate Fig. 4.4*c*. The same two steps (synthesis and edition steps) are then done in space V^3 (Fig. 4.4*d*). Now, let us consider that the "ears" of the monster are too close to the

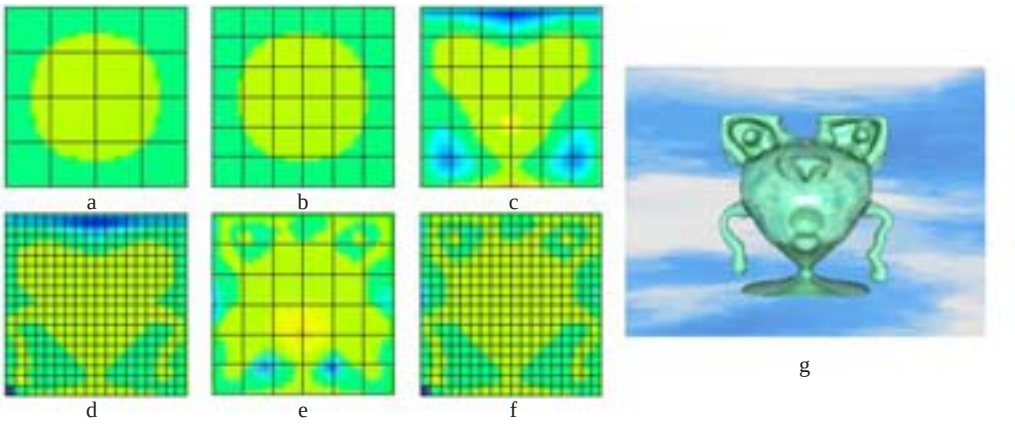


Figure 4.4: Different steps (from *a* to *f*) of the modelling process of a "monster" using multiresolution. The final single trivariate B-spline primitive is shown on the right.

arms. By applying a wavelets transform (analysis step) and editing the coefficients at level V^2, one can change the positions of the ears (Fig. 4.4*e*). If the ears are moved far enough from the arms in V^2, when going back to V^3, the arms are exactly recovered (Fig. 4.4*f*). The monster is now complete, with the correct position for the ears and the arms, Fig. 4.4*g* shows the result in 3D.

Another example is shown in Fig. 4.5. Figure 4.5*a* shows the first step of modelling an object (teapot-like shape), where the semi-transparent object is a zero-value iso-surface of the trivariate B-spline function defined by a set of $7 \times 7 \times 7$ control points. Fig. 4.5*b* shows the next step, where the number of control points has been increased to $19 \times 19 \times 19$. In this figure, finer parts have been added. Now, let us suppose that one wants to make some global modifications to this model, for instance, to create a cavity inside the object. Then, as it is shown in Fig. 4.5*c*, we can decrease the number of control points, and change the scalar value of few control points. If one wants to apply the same modification with the previous set of control points, he/she would have to change several points, and the expected result, a sphere-shaped cavity, would be difficult to obtain. Figure 4.5*d* shows the final result, with the previously created cavity modified at a finer level.

4.5 Interactive Modelling

4.5.1 Visualisation

We have implemented an interactive modeller on the base of the proposed primitive. Interactive rates are obtained as the B-spline functions are evaluated in constant time, due to the local support of the basis functions. It implies that when one changes the scalar value of a given point, the amount of updates is constant. Interactive rates are above 25 frames per second. As it is noticed in [FCG00], this system is limited by the number of primitives. As matter of fact, to combine several primitives together during the modelling increases the depth of the constructive tree, and results in an increasing complexity of the resulting function. Nevertheless, interactive

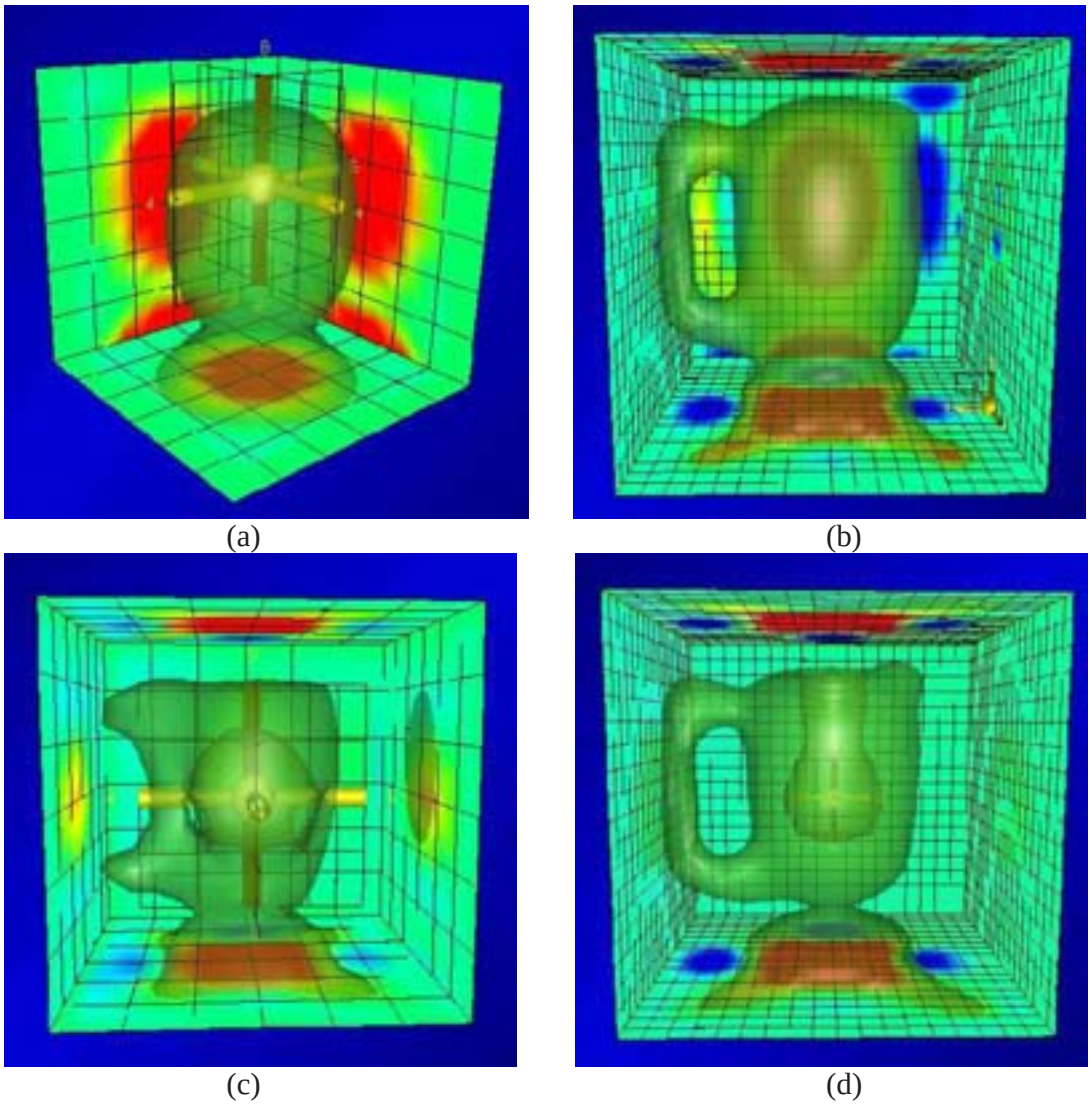


Figure 4.5: Application of the wavelet transform. A teapot-like object is modelled using a single cubic B-spline primitive.(a) Low level of details. The number of control points is $7 \times 7 \times 7$. Increasing number of control points between (a) and (b). Details are modelled in (b), i.e., using $19 \times 19 \times 19$ control points. The number of control points decreases in (c), where a cavity is created. This modification can be considered as global. In (d), the level of details is the same as in (b), all details, (e.g. the handle), are recovered without loss of any information, and local modifications of the cavity are made.

rates are maintained as long as one uses less than 15 primitives on a Pentium 450Mhz processor (and less than 35 on an 800 MHz processor).

To visualise the object, we polygonise the iso-surface defined as $F(x, y, z) = 0$, where F is a function evaluated using the constructive tree. Many different algorithms have been proposed for this task. We choose the polygonalisation algorithm based on hyperbolic arcs proposed in [PPP88]. As in the classical Marching Cube algorithm [LC87], an exhaustive enumeration of the 3D grid cells is applied, but instead of using a look-up table to generate the polygons belonging to a given cell, this algorithm uses the trilinear interpolation inside the cell combined with the bilinear interpolation on the cell faces, and resolves topological ambiguities using hyperbolic arcs. The strength of the algorithm is that the polygonal model it generates is topologically correct without unexpected holes. In our implementation, with the use of this algorithm, the polygonal surface is updated in real-time, which leads to an interactive modelling tool.

The major weakness of trivariate B-spline functions is that modelling a complex shape with a lot of details requires a huge number of control coefficients to deal with. Note that this problem also exists when manipulating classical polygonal meshes as well as usual B-spline patches, and it has been widely studied in that context. One ubiquitous solution - initially proposed by Parent in 1977 [Par77] - is to hide the underlying control parameters (that may be mesh vertices, B-spline control coefficients, or whatever) and show only high-level sculpting tools to the user. More precisely, each tool is defined as a filter that is applied to a whole set of control parameters. This paradigm has been applied to zero-set surfaces defined by trivariate scalar functions in [RE99]. The resulting method is very powerful but may encounter problems in some cases. For instance, to model a teapot as shown in Fig. 4.5, the easiest way is to create first a body without cavities, and then to create the cavities by removing the matter. If one considers only the surface of the object, it may be difficult to create this cavity, because one cannot access the inside part of the object.

Contrarily, we choose to keep direct access to individual control coefficients for modelling an object with the trivariate B-spline primitive. To model a shape using a cubic B-spline function, one can use a 3D cursor to select a position in the set of control points (coloured in yellow in Fig. 4.6). As one can see in Fig. 4.6, the polygonised iso-surface lies inside the cube, where its front faces are culled. The back faces are coloured according the function value of the B-spline. A "heat" colour scheme is employed, where the blue gradient ("cold") corresponds to negative function values, and the red gradient ("hot") corresponds to positive values. Green colours mean that the function values are around zero. The aim of this colour scheme is not to provide the exact function value, but rather an approximation to help the user to navigate in the set of control points.

4.5.2 Constructive tree with B-spline primitives

If one uses exclusively the trivariate cubic B-spline primitives to construct a complex object, a comparison with a hierarchical refinement of B-spline patches is needed [FB88]. The latter method consists in building a hierarchical tree of surfaces. Starting from a root surface, sub-surfaces, called overlays, are locally defined. Each overlay can also be refined with new sub-surfaces of finer resolution, to generate a tree of surfaces, where the resolution of each overlay is depends on the depth of the tree. This approach was applied to trivariate B-spline functions

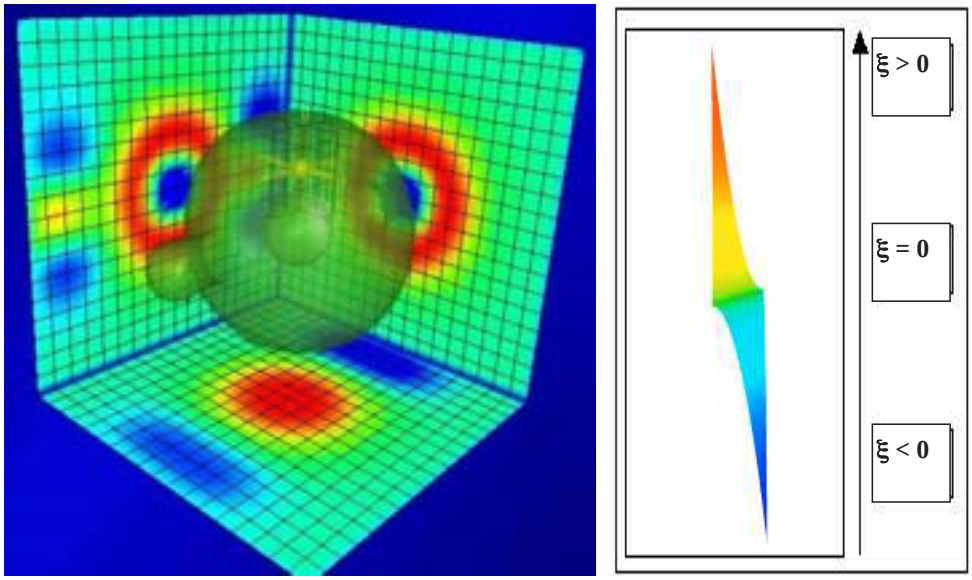


Figure 4.6: Snapshot of the user interface. (Left) The iso-surface is polygonised in real-time while modifying the scalar coefficients of the trivariate cubic B-spline function using the 3D cursor. To help the navigation, each face of the bounding cube is coloured according to a colour scale (Right) and to the function values, depending on the location of the cursor.

in [RE99]. The principle is to use an octree to sculpt a model at different levels of detail. This feature is very useful. For example, to model a teapot, the body can be modelled first, and then, in a finer resolution, the spout, the handle, and the lid. To obtain the whole model, a sum between sub-volumes is applied during the tree traversing process. To preserve C^k continuity, the first k and the last k rows and columns of scalar coefficients are set to zero.

This condition is enough to preserve the continuity of the resulting function during the overlay addition process, but the sum operation may lead to unexpected results. Figure 4.7 shows a simple example in the case of a 1D object. The first step is to model two segments AB and CD with a single B-spline curve (Fig. 4.7*a*). Then, we make a refinement in order to add another segment between two initial segments. This is achieved by adding an overlay (a B-spline curve as well) as shown in Fig. 4.7*b*. As it can be observed, at the extremities of the sub-curve, the continuity is preserved, and it reaches smoothly the main curve. The result of the algebraic sum between those two functions is a 1D object composed of three parts, AB, CD, and EF (Fig. 4.7*c*). Now, suppose that we want to translate the added part EF. As it can be seen in Fig. 4.7*e*, if one applies a simple sum, this part disappears, and one has to change the overlay value and by the way to return to the modelling process to change the scalar coefficients of the control points. On the other hand, one can consider this overlay as a separate function defining the additional segment EF (Fig. 4.7*b*), and thus use it as a simple primitive for a FRep tree. After combining two primitives by a union operation defined below (Fig. 4.7*d*), the result is similar to the sum operation. But now, when the second primitive (segment EF) is translated, the expected result is obtained (Fig. 4.7*f*).

Figure 4.8 illustrates this situation in the 3D case, where a teapot body and its handle are modelled using two B-spline primitives. In Fig. 4.8*a*, the algebraic sum operation is used. An equivalent visual result can be obtained with a union operation. After this modelling step, one wishes to translate the handle towards the body. The difference between the sum and the union operations appears. As it can be seen in Fig. 4.8*b*, unexpected results are obtained for the algebraic sum, whereas for the union operation (Fig. 4.8*c*), the expected result is obtained.

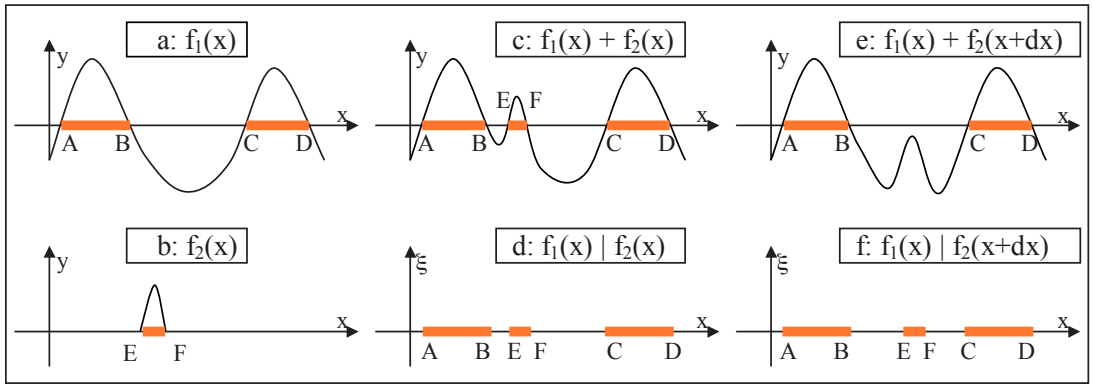


Figure 4.7: Difference between the algebraic sum and the union operation between two functions, in the case of a 1D object.

Figure 4.8: Difference between the algebraic sum and the union operation in 3D case. (a) The entire model is made of two trivariate B-spline primitives. The sum and the union operations give the same result. (b) The handle-like object is translated towards the body part: because of the translation, unexpected results are obtained for the algebraic sum. (c) The sum is replaced with the union operation. The expected result is obtained.

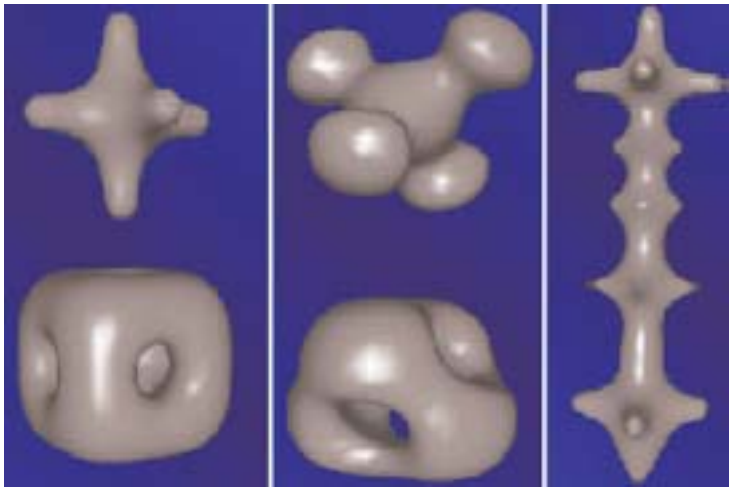


Figure 4.9: Different tools are proposed for modelling using trivariate B-spline functions. (Left and Middle) : Default tools. Note that a tool can be used in two ways: to add (upper image) or to remove (lower image) material. (Right): Example of a user defined tools.

4.5.3 Tools for modelling

Different tools are proposed to modify either a single control point of the B-spline or a set of them. Figure 4.9 shows different available tools. This set can be extended easily as the user can select a set of control points at any stage of the modelling process (similar to the copy and paste operation). However, to support the real-time polygonalisation, the size of the tool is limited to $11 \times 11 \times 11$ control points. When using a tool, one can either modify the scalar coefficients of the current B-spline primitive, or add a new B-spline primitive to the current tree. Figure 4.10 illustrates this functionality. The initial object is a single B-spline primitive which looks at this stage like a sphere, and the selected tool has a cross-like shape. The tool is defined as a set of control points with positive and negative scalar values. On the left side of the initial shape, the tool is directly applied to the control coefficients of the current B-spline primitive. The original shape is deformed according to the scalar values at the control points of the tool. On the right side, those scalar values are not applied directly, but a new cubic B-spline primitive is created, corresponding to the tool, and is added with the union operation to the current constructive tree.

To facilitate the modelling process, with the use of the same interface, some other operations are available, such as combining simple primitives with set-theoretic operations and affine transformations.

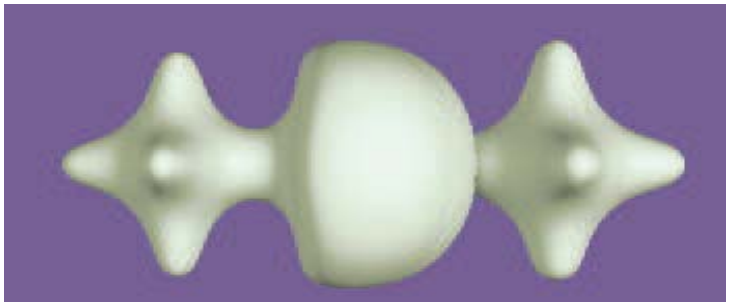


Figure 4.10: Example of the direct use of a tool and addition of another B-spline primitive to the constructive tree. The cross-like shape tool is defined as a set of control points containing both positive and negative scalar values. On the left, the tool is directly applied to the current object and deforms its shape; on the right, a new primitive is added, and the result is a simple union of the initial shape and the tool shape.

4.5.4 Geometry and attributes modelling

The same user interface can be used for modelling attributes of heterogeneous objects. Another example is given in Fig. 4.11, using the opacity as an attribute. The object is a light-bulb where the geometry is defined using only B-spline primitives. Twisting operations where applied to the mouthpiece and to the filament, which are two different B-spline primitives. The space partition for the attributes is simple, as the constructive geometric tree coincides with the attributes one.

In the case of the geometric model other than FRep, the only difference is that a geometric model should be imported first, as Fig. 4.12 shows. In this example, a standard B-rep object (the polygonal "Stanford Bunny") has been loaded, and different space partitions were modelled using the B-spline modeller. In this example, several cubic B-spline primitives were used to create space partitions for photometric attributes (colours and other shading parameters). While modelling, simple colours are used as in Fig. 4.12(*left*) to show which partition a vertex belongs to. One can then export the object to POVRay [Pov] or other formats for rendering. For each vertex defined in the B-rep model, a tree traversing procedure is processed, and depending on the partition the vertex belongs to, a texture index is defined.

Figure 4.13 illustrates the application of the proposed methods to a voxel model. The final result is shown on the right side, and the left side shows the space partition used for texturing the object. The geometric tree is rather simple and is composed of a union of an ellipsoid and a voxel object (the well-known "Siemens Head"). The main task was to create the space partition. In this example, the space partition constructive tree is composed of several primitives and operations. Namely, the mask-shape was defined using the proposed B-spline primitive, as well as the ring around it, combined with a solid noise using an algebraic sum. Several twisting operations were also applied to the torii inside the ring. We considered attributes as a tuple of shading parameters defining the colour and the opacity of the object, but also the ambient, diffuse, specular, and reflective coefficients.

After the modelling process is finished, the object with FRep geometry can be saved as a

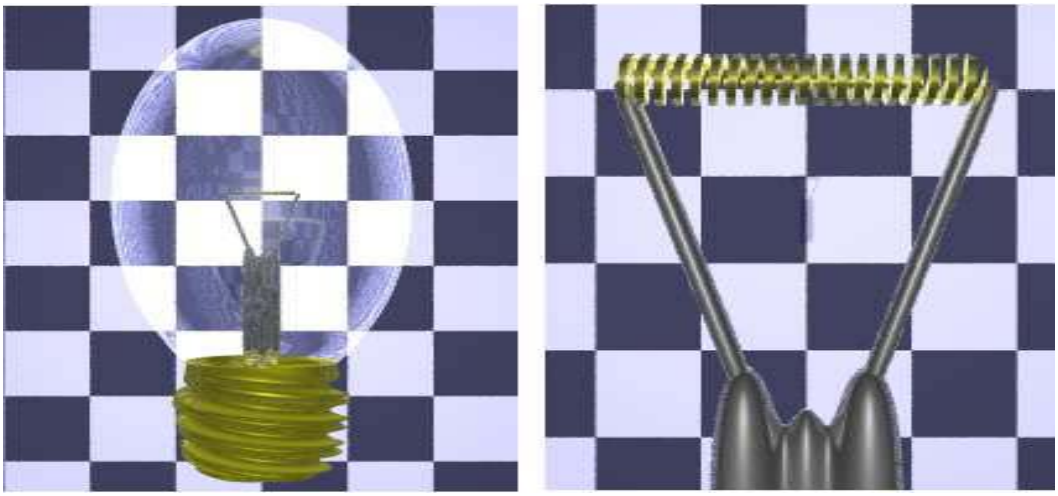


Figure 4.11: Complex heterogeneous object modelled using only trivariate B-spline primitives. (Left) Complete model of a light-bulb; (Right) Zoom on the filament of the light-bulb.

HyperFun script to be used later by other components of the HyperFun software environment [Pro]. HyperFun is a high level language supporting exchange of FRep and hypervolume models. The proposed trivariate B-spline primitive has been included in the FRep library of HyperFun. A HyperFun script can also be used to save both space partitions for attributes and geometric model.

4.6 Conclusion

A hypervolume is a model of a heterogeneous object. It is considered as a multidimensional point set with multiple attributes. Attributes represent different properties of real or abstract objects and processes. In this chapter, the function representation is used as the basic model for representing a point set geometry and its attributes. We presented a modelling approach for hypervolumes that mixes two usual modelling paradigms, namely interactive sculpting and constructive modelling. The sculpting environment allows one to define either geometry of an object, or space partitions for attributes in a similar manner.

To model heterogeneous objects, a new primitive for the FRep tree is proposed on the base of the uniform trivariate B-spline function. To facilitate the modelling process, a multiresolution scheme based on the wavelets transform is applied. To visualise the object, its surface is polygonised and visualized at the interactive rate. The space partition obtained in the modelling process can be applied to define attributes for objects of arbitrary geometry such as BRep or voxel models.

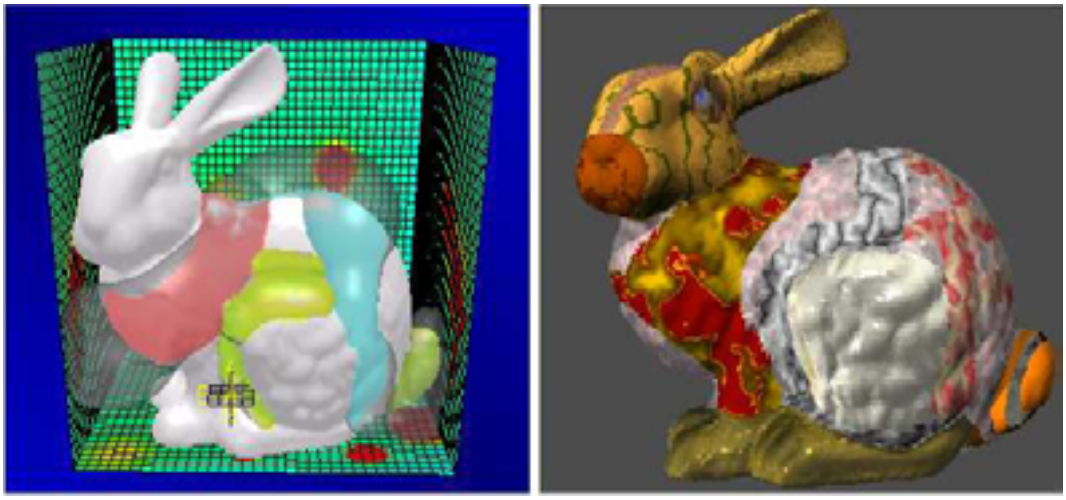


Figure 4.12: Interactive modelling, application to a BRep object. (Left) The GUI used to texture the object. Several B-spline primitives are used to define the space partition. (Right) The corresponding textured BRep object, using a surface ray-tracing engine.

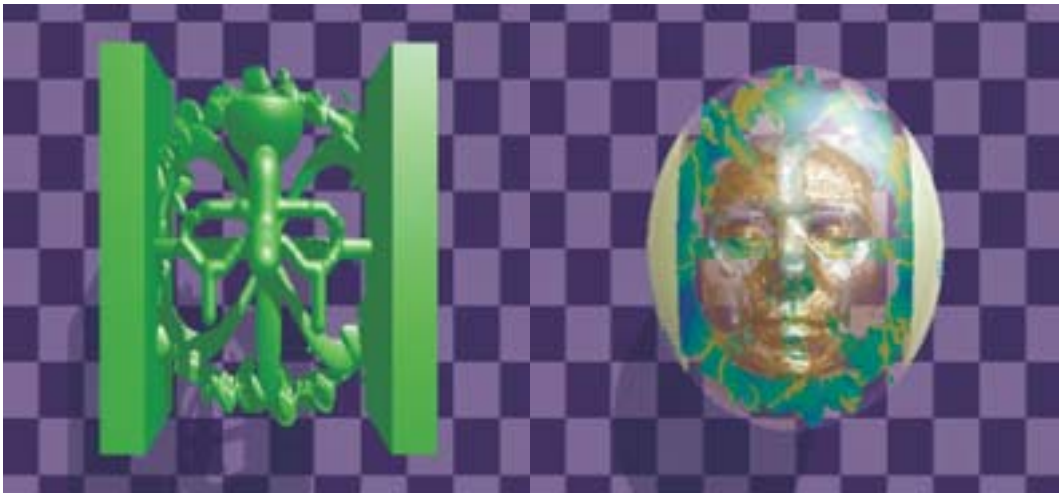


Figure 4.13: Application to a voxel model. (Left) The space partition used to texture the object. (Right) The corresponding voxel model, rendered using a volume rendering engine.

Chapter 5

Deformations in the constructive hypervolume model

5.1 Introduction

In the previous Chapter, we considered the constructive approach to model different objects and proposed a new primitive that mimics the sculpting metaphor, where one can add or remove material to an object. We propose in this Chapter to model objects using another approach that is, starting from an existing object, to deform it until the desired shape is obtained.

One of the first deformation scheme, called *warping*, was proposed in [Par77]. Given a polygonal surface, a vertex is selected, and moved towards the outside of the object. Neighbour vertices of the mesh are then displaced according to a distribution function depending on their distance to the displaced vertex. However, this method can not be used to define global deformations.

Another technique, called FFD [SP86], which stands for *Free-Form Deformation*, embeds the object to be deformed into a rectangular volume defined using a control point lattice. Then, while moving the control points, the embedding volume and the embedded object are deformed. Several extensions have followed, namely EFFD for *Extended Free-From Deformation* [Coq90], where the embedding volume is replaced by some more complex one, or the RFFD (*Rational Free-Form Deformation*), where another degree of freedom is provided while adding a weight factor to the control points of the lattice. One of the last extensions of the FFD model is presented in [MJ96], where a subdivision volume is used to embed the object.

Several other deformation techniques exist. For instance, instead of embedding the object into a volume, an axis can be defined in the object. Then, as one deforms the axis, the object deformation follows. The axis can be a broken line [Par77, LCJ94], a Bézier curve [CR94] or even a Bézier surface [Mik96]. Other approaches to deformation include the "simple constrained deformation" using ellipsoids *Scodef* [BB91, BR94, Bec94], and its extension with generalised metaballs [JLP98], the "implicit free-form deformation" *IFFD* [Cre98], and the "Wires" model [SF98].

Control of 2D image deformation using feature shapes (points, segments) was described in [BN92]. Similar approaches were proposed independently for controlling 3D deformations

[SPKS95, RM95] and 3D metamorphosis of homogeneous volumes [LGL95].

Nevertheless, most of the mentioned above techniques can not be directly used in the constructive approach. Indeed, most of them are applicable only to a discrete structure (e.g., polygonal mesh, control mesh of a spline, etc.), as they are defined using a forward mapping. Even if some deformation tools deform the whole space using a function from $\Re^3 \rightarrow \Re^3$, such as in [Cre98, Bec94], they are also applied to a discrete structure.

In the previous Chapter, we described how to model an object while using a B-spline primitive. The constructive approach and the volume sculpting approach were combined using the unifying FRep model. In this Chapter, we propose to define a new node for the FRep tree for deformations. One important goal is to provide the possibility for the user to model an object without the traditional separation between the constructive approach, the sculpting process, and the deformation steps. Usually, the modelling scheme is as follows: first, one models an object in some way, and then uses deformation tools to obtain the desired shape. As most of the existing tools for deformations are using forward mapping, once the object is deformed, one can hardly return to the modelling step, and combine the existing object with some other.

In order to be able to switch from the modelling step to the deformation step whenever it is needed, one solution is to define the deformation using inverse mapping, and thus to provide a new node for the FRep tree. To calculate the inverse mapping from the forward mapping of the previous work is a very difficult operation, and is even sometimes impossible. A preferable solution is to define a new node from the scratch. Of course, several similarities can be found with the previous works, and we used general ideas to define this new node. But one has to keep in mind that the definition we propose is based on inverse mapping, and thus, even if the visual result of the deformation is close to existing deformations, the underlying idea is different.

Deformation nodes already exist in the FRep tree. One can easily twist, taper or stretch an object along an axis. These deformations are based on the work presented in [Bar84], and known as *axial-based* deformations [1], but the set of available deformations is quite limited. In [SPKS95], deformations were defined using point-controlled space mapping, but it had a too global character due to the interpolation with radial-basis functions. A general framework for deformations in the FRep model has been proposed in [SP98], using the *extended space mapping* introduced in the previous Chapter.

Here, we propose in the following to define new deformation tools to deform an object, either locally or globally. In the first section, we present the underlying idea of the proposed deformation scheme, and then different extensions are defined. To illustrate our approach, we provide several examples. We choose to consider photometric properties (i.e., colour textures) for the attributes of the constructive hypervolume. As it will be shown, one of the main properties of the proposed deformation is that it can occur in several constructive trees, the geometric one and the attribute one.

5.2 Simple deformations using space mapping

In this section, we introduce the underlying idea of the deformation using space mapping. Let us first consider a simple translation. Let f be a defining function of some geometric object,

[1] This name has been already used in [Par77, LCJ94] and other, but the tools are different.

and T a translation vector defined as (dx, dy). Then, given a defining function f of a 2D object (Fig. 5.1*a*), the inverse mapping for this transformation is defined as:

$$T : f(x, y) \rightarrow f(x - dx, y - dy) \tag{5.1}$$

This operation is globally applied to the entire object as dx, dy are constants (Fig. 5.1*b*). Now, let us define the deformation by non-linear space mapping. Consider the same 2D object and the displacement of a point A towards a point A'. To define a local deformation centred in the point A, one has to respect the two following requirements:

- (dx, dy) have maximum values at A'.
- (dx, dy) drop uniformly to zero when (x, y) is far from A'.

To satisfy these requirements, dx and dy have to become two bell-shaped functions. We propose to use the following functions for $dx(x, y)$ and $dy(x, y)$:

$$\begin{cases} if \gamma \leq 1 \\ \qquad dx(x, y) = e^{-\gamma^2} \times (x_{A'} - x_A) \\ \qquad dy(x, y) = e^{-\gamma^2} \times (y_{A'} - y_A) \\ else \\ \qquad dx = 0 \\ \qquad dy = 0 \end{cases} \tag{5.2}$$

where:

$$\gamma^2 = \frac{(x - x_{A'})^2 + (y - y_{A'})^2}{r^2} \tag{5.3}$$

As it can be observed, the displacement is maximum when the considered point is placed at A' (Fig. 5.1(c)). We explicitly set the displacement to zero when γ is greater than one. This insures a localised deformation that depends on the size of "bell" of the curve. Note that the point A is arbitrarily selected, and thus, a space mapping can be defined for any given point in the space. The function $\gamma(x)$ is similar to the Euclidean distance function.

The parameter r is a real value given by the user, and it defines the area of influence of the space mapping, (and thus the shape of the "bell"). Figure 5.2 shows the deformations corresponding to different values of r in the 3D case. As it can be seen, different levels of deformation, local or global, can be obtained. The base shape is an ellipsoid. Each row in the figure represents a different value of the r parameter. The first column represents the effect of the displacement of a single control point inside the object, and the second column represents the displacement of the control point outside the object.

In Fig. 5.3(*left*), the geometry is defined as a single semi-transparent yellow sphere. The additional space partition for the attributes is defined as a union of four smaller spheres. The attributes of these four spheres of the space partition are constructed as successive fully opaque red and green layers. Then, two space mappings are defined in order to deform the object (shown by red arrows). Located along the vertical axis and inside the geometrical sphere, two control points of the non-linear space mapping are moved vertically towards the outside of the object (red arrows). In the case of the upper point, the defined space mapping is applied both

to the geometry and to the space partition for the attributes. As one can see, the two upper spheres are also deformed, and the red and green colour strips follow the deformation. We choose deliberately to apply the deformation only to the two upper spheres. No space mapping was applied to the two lower spheres, and they remain unchanged, whereas the bottom of the geometrical sphere has been deformed.

5.3 Deformations using field functions

In the previous section, we provided a definition of deformations using space mappings. This definition can be extended in various directions. Recalling eq. 5.2, one can reformulate the space mapping as follows:

$$\begin{cases} X' = X - \Delta(X) \\ \Delta(X) = f \circ \gamma(X) \times (A' - A) \end{cases} \tag{5.4}$$

where X is an input point of the Euclidean space, X' is its image after applying the inverse mapping, A and A' are the source and the target points of the deformation.

This definition looks exactly the same as the basic formulation of blobby objects proposed in [Bli82], where f is a potential function, and γ a distance function. Two different directions can be then followed to extend the definition of the space mapping. Either one can change the potential function f or the distance function γ. The next two subsections consider the potential function. The discussion supposes that the distance function γ has been already chosen. The definition given in eq. 5.3 was used to produce the majority of the examples.

5.3.1 Potential function

Several authors studied the definition and the behaviour of potential functions [Bli82, NHK$^+$85, Gas93]. These functions can be divided in two main families. The first one has an infinite support [Bli82, KAW91] and thus takes non-zero values everywhere in the space. In the case of deformations, this family of potential function requires some arbitrary conditions to avoid small unwanted deformations (see eq. 5.2 for instance, where we set explicitly the deformation to zero for a given value). The other family of potential function has a finite support, and fits well to the locality requirement of deformation.

In the following, we first recall some results of previous works on potential functions, based on the survey and conclusions proposed in [BS95]. Historically, one of the first potential functions with a finite support was proposed in [NHK$^+$85], and known as *metaballs* based on quadratic polynomials.

$$f(\gamma) = \begin{cases} \frac{4}{3} - 4\gamma^2 & \text{if} \quad 0 \leq d < \frac{1}{3} \\ 2(1-\gamma)^2 & \text{if} \quad \frac{1}{3} \leq d < 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

Other formulations for the locally supported functions were then proposed in [WMW86] and [MI87]. Plots of eq. 5.5 and the two mentioned formulations are shown in Fig.5.4*a*. The set of available shapes are quite limited as the functions only depend on the chosen distance.

Another formulation was proposed first in [Gas93], where the parameter p is introduced to control the behaviour of the potential function (such a parameter is often called *hardness factor*

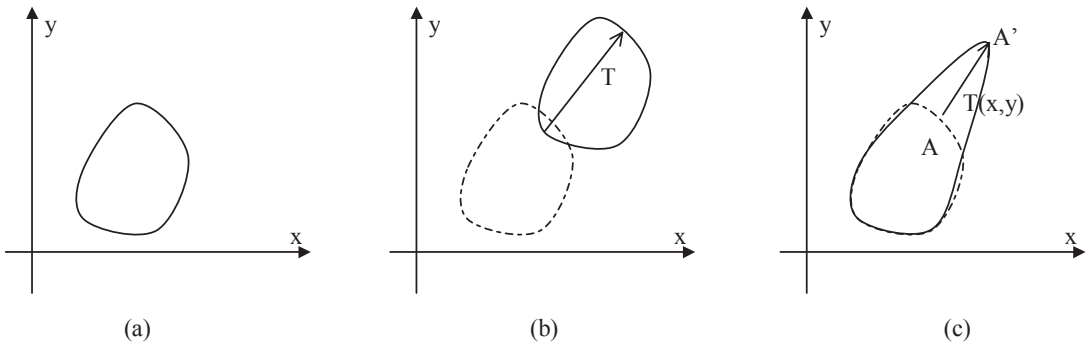


Figure 5.1: Simple example of space mapping. (a) The iso-contour of a 2D object is defined in the xy plane. (b) Deformation of the object using a linear space mapping, i.e., a constant translation vector T. (c) Deformation of the object using a non linear space mapping.

Figure 5.2: Influence of the parameter r in the definition of the space mapping. An ellipsoid is deformed using space mapping with one control point displaced towards the inside of the object (left column) and outside (right column) with three different values of the parameter r.

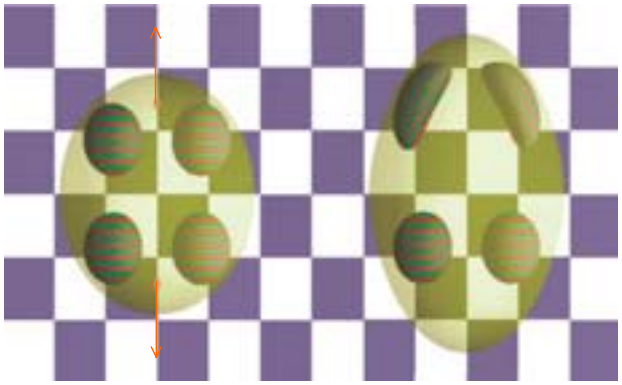


Figure 5.3: Deformation of a constructive hypervolume. The geometry of the object is defined as a semitransparent yellow sphere. Inside, a space partition is defined as a union of four smaller spheres. Non-linear space mappings are defined while moving two points (red dots and arrows). In the upper part, the space partition follows the deformation; in the bottom it is independent.

in the literature):

$$f(\gamma) = \begin{cases} \frac{1}{4}(2+p-2p\gamma) & \text{if} \quad 0 \leq d < \frac{1}{2} \\ (-2+p+8\gamma-2p\gamma)(1-\gamma)^2 & \text{if} \quad \frac{1}{2} \leq d < 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

This potential function offers a good hardness control, but depending on the value of the hardness factor, it can become negative (i.e., when p is too large). Figure 5.4b shows the plot of this function with different values for p.

Another potential function was proposed in [KAW91] with an unbounded hardness factor:

$$f(\gamma) = \begin{cases} \frac{1}{2} + \frac{1}{2}\frac{arctan(p-2p\gamma)}{arctanp} & \text{if} \quad 0 \leq d < \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

A very similar result in the function behaviour can be obtained while using the potential function presented in [BS95], as one can see in Figs. 5.4*b* and *c*, where both potential functions are plotted. The main advantage of the latter one is that it offers a lower computational cost as it is based on a piecewise rational polynomial, and not on a *arctan* function. Furthermore, this function uses a squared distance, and avoids the use of the additional square root:

$$f(\gamma) = \begin{cases} 1 - \frac{(3\gamma^2)^2}{p+(4.5-4p)\gamma^2} & \text{if} \quad 0.25 \leq d < 1 \\ \frac{(1-\gamma^2)^2}{0.75-p+(1.5+4p)\gamma^2} & \text{if} \quad 0 \leq d < 0.25 \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

The use of the hardness factor is intuitive, and the application of this function as a deformation tool is straightforward as Fig. 5.5 shows, where a block is deformed using the above potential function. For a similar displacement along the vertical axis, different hardness factors have been used. The distance function used for this example is similar to the one defined in eq. 5.3. More details on the distance functions can be found in the next section.

5.3.2 B-spline based deformations

The set of deformations generated by the potential function proposed in [BS95] may not offer a sufficient variety. Thus, we propose here to use another distance function based on a parametric curve [NN94, KAW91, BS95]. We choose the cubic B-spline functions for illustrations. Any other interpolation function can also be used. As it was previously mentioned, the main requirement to a potential function is to drop to zero given a certain radius of influence.

Let us consider a B-spline function of one variable $f(t)$, defined by a set of n control points $\{P_i\}$ as follows:

$$f(t) = \sum_{i=0}^{n-1} N_i(t) P_i \tag{5.9}$$

where N_i is an i^{th} B-spline basis function, and t belongs to the parametric space $[0, 1]$. Our interest is turned towards the function value of the spline, and thus we insure that $f^x(t) = t$. Furthermore, we also assume that outside this domain, the B-Spline function remains equal to

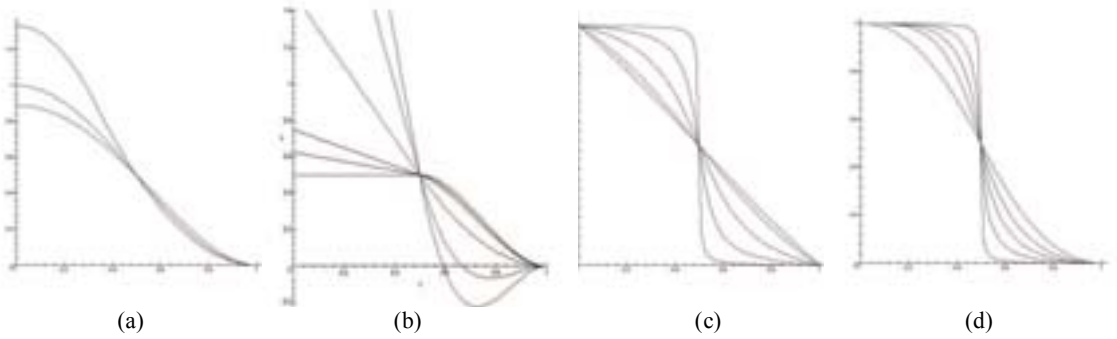


Figure 5.4: Plots of different potential functions. (a) First introduced functions, without hardness factor, corresponding to eqs. 5.5 and later formulations. (b,c,d) Potential functions with different hardness factors, corresponding respectively to eqs. 5.6, 5.7 and 5.8.

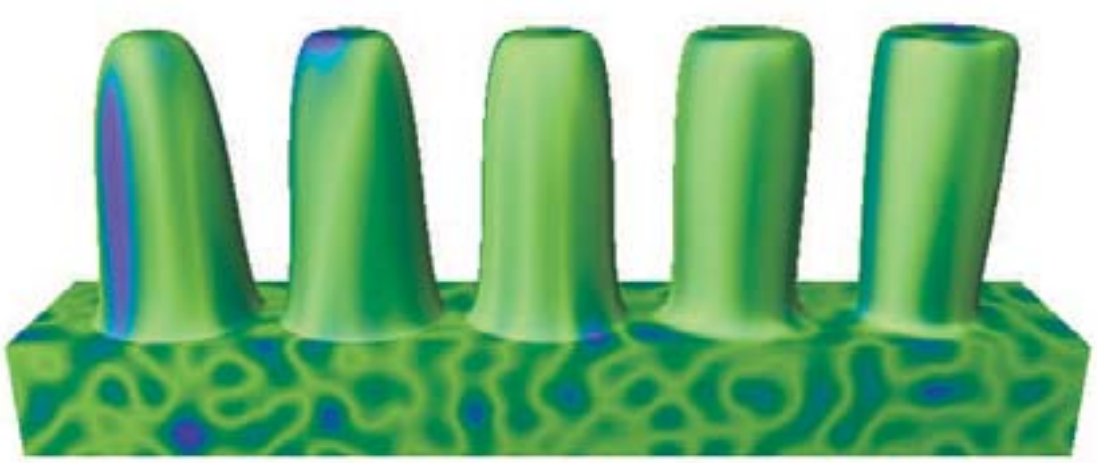


Figure 5.5: Deformation of a block by space mapping using the potential function 5.8. Five different deformations using the same displacement in the vertical direction with five different hardness factors.

zero (see previous chapter for more detailed explanations on B-spline). In the following, the B-spline function will be used as a potential function, and the distance function will define the parameter t, i.e., $t = \gamma^2(X)$.

Let us consider two points A and A', as respectively the source and the target point, and a distance function (eq. 5.3 for instance). Then, for every given point X, one can evaluate the distance between X and A', $\gamma(X)$. When X is placed at A', $\gamma(x)$ is equal to zero, the B-spline function is equal to $f(0)$, i.e., equal to the scalar value of the control point P_0. When X is far from A' and the distance $\gamma(x)$ increases, as it was stated in the definition of the B-spline function, this function will drop to zero when the distance function is greater than 1.

To define deformations using the B-spline potential function is simple. An additional step is required as one needs to model the shape of the B-spline function, i.e., to change the scalar coefficient of the control points . In the following, this shape will be called *profile*. Once this profile is defined, the application of the space mapping is straightforward. Let us consider the example given in Fig. 5.6. The aim is to deform a block object using the proposed method based on B-spline. The first step where the B-spline function is defined is shown in Fig. 5.6*a*. The control points are shown in grey, and the B-spline curve in blue. The maximal displacement corresponds to the first control P_0. To insure that the B-spline function will be equal to zero when t is equal to 1, two control points have been added, shown in green (in a more general context, one can add k control points in order to obtain a C^k continuity). Figures 5.6*b* and *c* show the result of the use of this B-spline function to deform the initial object. Greyscales represent the area of influence centred at the target point. As one can see, the deformation follows the B-spline profile and is symmetric where the target point is the centre of the symmetry.

Previously, we formulated the requirements that the displacement has to be maximal for the target point and to drop uniformly to zero as the given point is far from the target point. This statement is still exact, but may be too restrictive in the case of the deformation based on the B-spline function. Consider, for instance, Fig. 5.7*a*, where a profile has been designed. In this example, the greatest scalar value of the control points in not assigned to P_0, but to some other points. Furthermore, negative scalar values have been assigned to some of them. The result, shown in Figs. 5.6*b* and *c*, is still intuitive and follows the profile of the B-spline function. So we can reformulate the requirements of the deformation in this case as:

- the deformation should be centred along the displacement vector defined by AA', where A is the source point and A' the target point;
- the displacement should reach zero at some given distance with no restriction of the monotone decreasing.

5.3.3 Distance functions

Once the potential function is chosen, (eq. 5.8 is a good candidate), one can use different distance functions. Indeed, as the potential function is a function of one variable, it depends only on the distance value, and even if different values for the hardness factor produce different deformations, the set of available deformations is quite limited. Recalling eq. 5.2, the definition

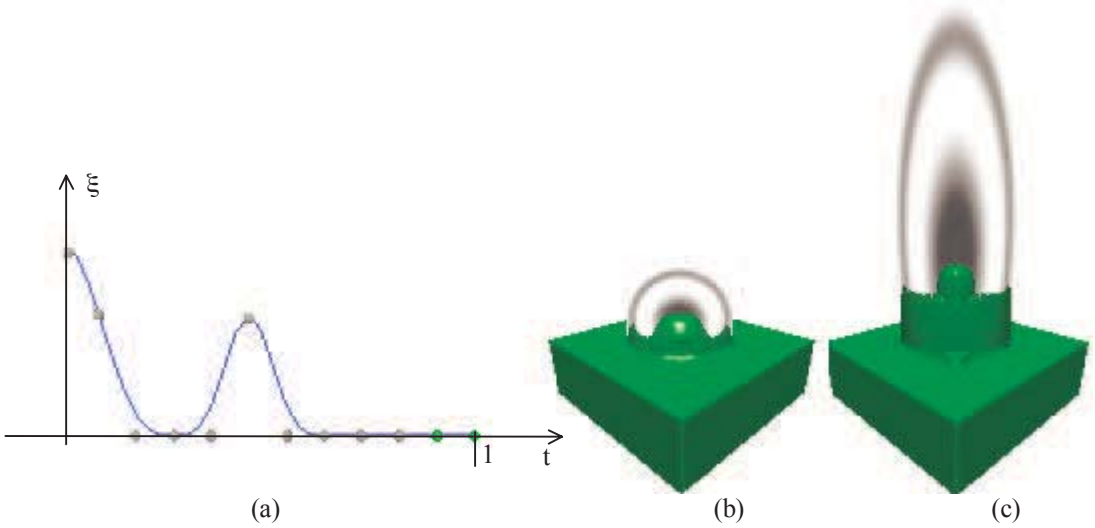


Figure 5.6: Block deformed using a B-spline based potential function. (a) Profile of the B-spline function. (b),(c) The same source point has been used, with two different target points along the y-axis.

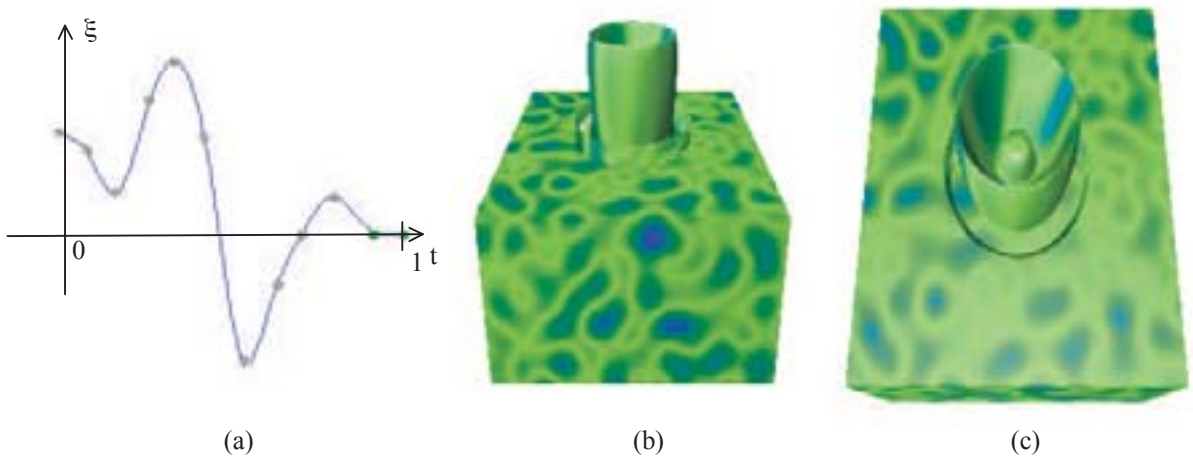


Figure 5.7: Block deformed using a B-spline based potential function. (a) Profile of the B-spline function. (b),(c) Different views of the deformed object.

of the distance function γ was in the 3D case:

$$\gamma = \frac{(x - x_{A'})^2 + (y - y_{A'})^2 + (z - z_{A'})^2}{r^2} \tag{5.10}$$

where A' is the target point, and r is the radius of influence. This distance function is the traditional Euclidean distance function, also known as the *spherical distance*. Figure 5.8 shows in greyscale the area of influence around the target point. Black colour corresponds to $\gamma = 0$, and white colour to $\gamma \geq 1$. Due to the definition of the distance, it is clear that the deformation is symmetric along each axis. Furthermore, the displacement of the source and the target point is limited, depending on the radius of influence r. In Fig. 5.8*a*, the distance between the source and the target point is lower than r; the deformed part remains connected to the object. When the point is moved too far from the source, a disconnected part appears. This case is illustrated in Fig. 5.8*b*.

A simple solution to insure that no disconnected part will appear is to normalise the definition of γ depending on each axis, and thus change its definition from a spherical distance function to an *ellipsoidal distance* function:

$$\gamma = \frac{1}{(r_x)^2}\frac{(x - x_{A'})^2}{1 + (x_{A'} - x_A)^2} + \frac{1}{(r_y)^2}\frac{(y - y_{A'})^2}{1 + (y_{A'} - y_A)^2} + \frac{1}{(r_z)^2}\frac{(z - z_{A'})^2}{1 + (z_{A'} - z_A)^2} \tag{5.11}$$

where r_x, r_y and r_z are three parameters which define the area of influence along each axis, and can be called radii of influence. Figure 5.8*c* shows the result while applying this new definition of gamma. The same displacement of the target point has been used for both Figs. 5.8*b* and *c*, without changing the radii of influence.

For clarity reason, let us introduce the following notations. Given a point X, a source point A, a target point A', and three radii of influence, the corresponding normalization $\tilde{X}$ of X is defined for each coordinate X_i of X as follows:

$$\tilde{X_i} = \sqrt{\frac{(X_i - A'_i)^2}{r_i^2(1 + (A'_i - A_i)^2)}} \tag{5.12}$$

One natural solution extension is to use the generalisation of the Euclidean distance function, known as the D^n distance:

$$\gamma(\tilde{X}) = (|\tilde{x}|^n + |\tilde{y}|^n + |\tilde{z}|^n)^{\frac{1}{n}} \tag{5.13}$$

This extension offers only one additional degree of freedom. Furthermore, if one uses the potential function defined in eq. 5.8, the behaviour of the deformation depending on either the hardness factor or the power n of the distance function is redundant. A more intriguing solution is to use *superquadric distance* functions.

If one considers eqs. 5.10 and 5.11, the analogy with the defining function of respectively a sphere and an ellipsoid is straightforward. Then, it naturally comes that one can use the definition of a superellipsoid for γ. The defining function of a superellipsoid is:

$$F(x, y, z) = 1 - \left(\frac{(x - x_0)^{\frac{2}{n_2}}}{a} + \frac{(y - y_0)^{\frac{2}{n_2}}}{b} \right)^{\frac{n_2}{n_1}} - \frac{(z - z_0)^{\frac{2}{n_1}}}{c} \tag{5.14}$$

where (x_0, y_0, z_0) is the centre of the superellipsoid, and (a, b, c) its half axis. The parameters (n_1, n_2) control the shape. To simplify, one can say that n_1 defines the "sharpness" in z-direction, and n_2 in xy-direction. When (n_1, n_2) are close to 0, a "block" shape is obtained, and when $(n_1, n_2) >> 0$ the result is more similar to a "star" shape.

The use of the definition of the superellipsoid (or supertoroid to provide a C^2 continuity) is straightforward. The target point corresponds to the centre, and the length of the displacements along each axis corresponds to the radii. Then, using the previously introduced notation, the definition of γ becomes:

$$\gamma(X) = \left(\tilde{x}^{\frac{2}{n_2}} + \tilde{y}^{\frac{2}{n_2}} \right)^{\frac{n_2}{n_1}} - \tilde{z}^{\frac{2}{n_1}} \tag{5.15}$$

This definition allows a new set of deformations. Figure 5.9 plots different $\gamma(\tilde{x}, 0, 0)$, where γ is a superellipsoid function, with different values of (n_1, n_2).

The direct application is shown in Fig. 5.10. A forest of superellipsoids deforming an ellipsoid is shown, with different values for $(n1, n2)$. Note that the texture pattern follows the deformations, as the same deformations also occur in the attribute trees.

5.4 Shape driven deformation

In the previous section, we defined an inverse mapping for point to point based deformations, where a source point was moved towards a target point. This case is the simplest, and we propose to use it as a framework for the following extensions. First, we will define a general deformation technique, and then go through examples and case studies exploring this new deformation.

5.4.1 Framework for deformations

The extension of a point to point based deformation to an area based deformation can be decomposed into two basics steps. Given a source point A and a target point A', one can deduce:

1. an area of influence height AA' (such as a cone, a block, an ellipsoid ...).
2. an area where every given points in the space will be projected onto. This area will be called the projection area (such as a line segment, a plane, and other ...).

Figure 5.11 is illustrates the above statements in 2D. The initial state is shown in Fig. 5.11*a*, where a part of a surface is shown, as well as the source and target points. Figure 5.11*b* shows the area of influence and the projection area, a plane in this case, and Fig. 5.11*c* shows the expected result. As one can see, the result is very similar to a point to point based deformation, but as it will be shown in the following, the use of such extension allows one to obtain deformations that can be hardly obtained with only point to point deformations.

Without loss of generality, the area of influence is defined using a FRep defining function, which will be called $Z(X)$ in the following. To insure that the corresponding function value is in the interval $[0, 1]$, we use the following mapping:

$$\tilde{Z}(X) = \frac{1}{2} \left(1 + \frac{Z(X)}{\sqrt{(\vartheta + Z^2(X)}} \right) \tag{5.16}$$

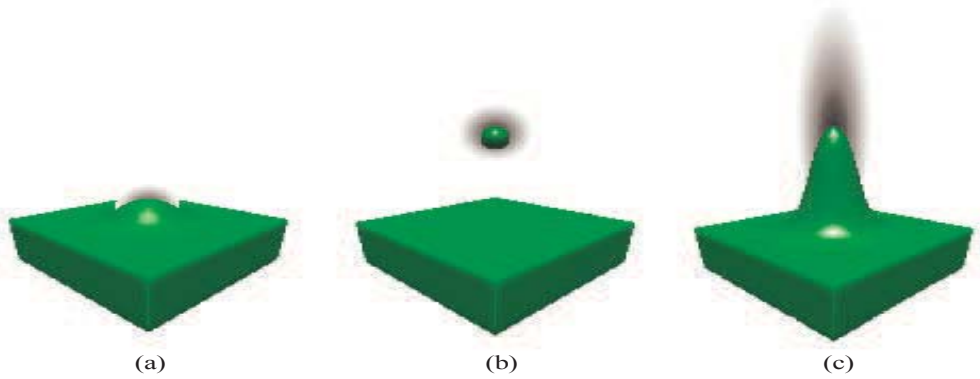


Figure 5.8: Distance function. A point is moved towards the outside of the object, along an axis. The grey area shows the influence around the target point. Black colour corresponds to a distance equal to 0, and white to distances greater or equal to 1. (a) Spherical distance. The displacement is small; the displaced material is still linked to the object. (b) Spherical distance. The target point is moved too far from the object. (c) Ellipsoidal distance. Area of influence now depends on the distance of the given point to the source point and to the target point.

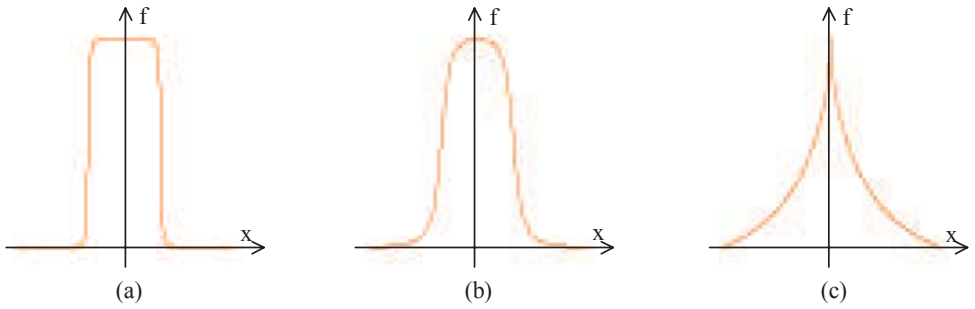


Figure 5.9: Plots of the displacement function based on the superellipsoid defining function $F(x, 0, 0)$, with different values for (n_1, n_2), respectively $(1.1), (5, 1)$ and $(50, 1)$.

Figure 5.10: Ellipsoid deformed using the superellipsoidal distance function. Nine target points have been used.

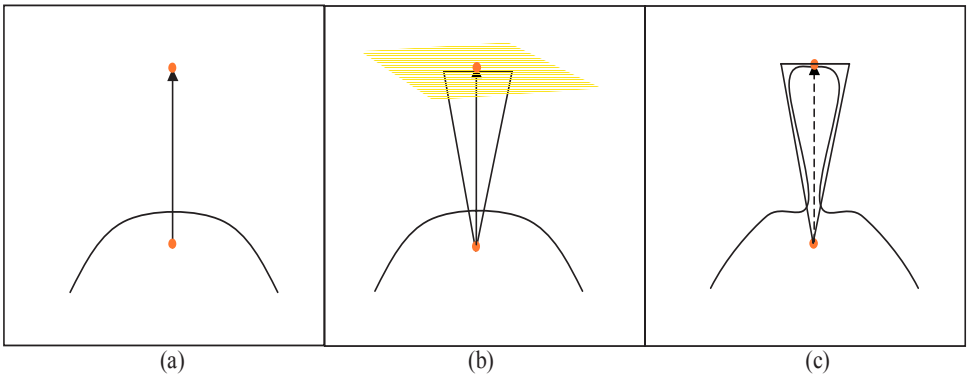


Figure 5.11: Rectangle surface based deformation. Extension steps. Given a source point A and a target point A', one can deduce the area of influence (box) and the plane orthogonal to AA' and containing A'.

The Z function and the mapping function are very important, and have an important influence on the shape of the deformation. Before going further in the study of those functions, we propose first to define the general framework for the deformation and then present two examples. The subsection 5.4.3 is dedicated to them.

We propose to follow the scheme described below to define a general deformation. The following data a regiven:

1. a source point A,
2. a target point A',
3. a point X of the Euclidean space
4. an area of projection, and H the projection of X onto this area,
5. an area of influence Z,
6. a potential function f (eq. 5.8),
7. a distance function γ (eq. 5.11).

The inverse mapping for the point X is defined as:

$$\begin{cases} X' = X - \Delta(X) \\ \Delta(X) = f \circ \gamma(X)\tilde{Z}(X)(H - A) \end{cases} \tag{5.17}$$

If one considers only the point to point based deformations, the analogy is straightforward. Source and target points are clearly identified, the projection area is reduced to A'. The area of influence Z can be omitted in this case.

To illustrate the proposed steps to define an inverse mapping, we propose first to consider a simple example, and then some more complex case studies, where deformations are combined in a FRep tree.

5.4.2 Examples of deformations

In this subsection, we will show how to define an inverse space mapping when the source is a point, and the target area is isomorphe to a rectangle, as illustrated in Fig. 5.11. The next example will consider the case when the target area is a segment.

Given a source point A and a target point A', one can define a cone of height AA', and a plane orthogonal to AA' and containing A'. The radius of the cone specifies the size of the deformation. By analogy with the general framework of the previous subsection, the defining function of the cone corresponds to the function Z. For each given point X, one can calculate its projection H onto the plane. In this example, we choose a projection similar to the perspective one, where the vanishing point is A, and H is defined as the intersection of the line AX and the plane. The distance function is then calculated depending on H (and not A' as previously). The use of the area of influence takes now its full meaning. Indeed, if Z is not included in the definition of the inverse mapping, an infinite deformation occurs. Figure 5.12a shows the result of such deformation. The object to be deformed is an ellipsoid. The source point A coincides with its centre, and the displacement of the target point A' is along the vertical axis, z. The area for the projection is defined as the xy plane, translated along the z axis of AA'. The result is an infinite deformation. It naturally comes that to cut the unwanted part of the deformation, one has to define the area of influence Z. Different results can be obtained depending on the choice of Z. Figure 5.12b illustrates the previous explanations, as we choose a cone with axis AA' for the area of influence, and in Fig. 5.12c, we choose to replace the cone by a block.

Instead of considering a plane as the projection area, one can also consider a line segment. In the example shown in Fig. 5.13, the initial configuration for the source and target points is identical to the previous example of Fig. 5.12. The difference comes from the projection area and from the influence area. The projection area is defined as a line segment, and we choose to use a parallel projection to map each point X on it. The area of influence is defined as a convolution triangle. In Fig. 5.13a, one can see the initial state, where the object to be deformed is an ellipsoid, the source and the target points are positioned, as well as the line for the projection, and the triangle for the area of influence. Figure 5.13b shows the projection of three different points of the space X_1, X_2, and X_3. The three new ellipsoids in the figure correspond to the distance function (as usual, the internal part of the ellipsoid represents distances to centre below or equal to 1). The point X_1 is mapped onto the projection line, parallel to AA'. Its image is H_1, but as X_1 is located outside the area of influence, the function $\tilde{Z}(X_1)$ is equal to zero and X_1 will be mapped onto itself. The second point X_2 is mapped onto the line, and its image H_2 corresponds to the centre of the ellipsoidal distance function γ. As $\gamma(X_2)$ is lower than 1, and as X_2 lies inside the area of influence, the point X_2 is mapped to some other location (close to A to be more precise). The third point X_3 emphasises the importance of the choice of the projection. In this case, the projection is parallel. The point X_3 lies inside the area of influence. Its image generates an ellipsoidal distance field, but as one can see, the value $\gamma(X_3)$ is greater than one. Thus, the corresponding potential value $f(\gamma(x_3))$ is equal to zero, and the point X_3 will be finally mapped onto itself. Figure 5.13c shows the resulting deformed object.

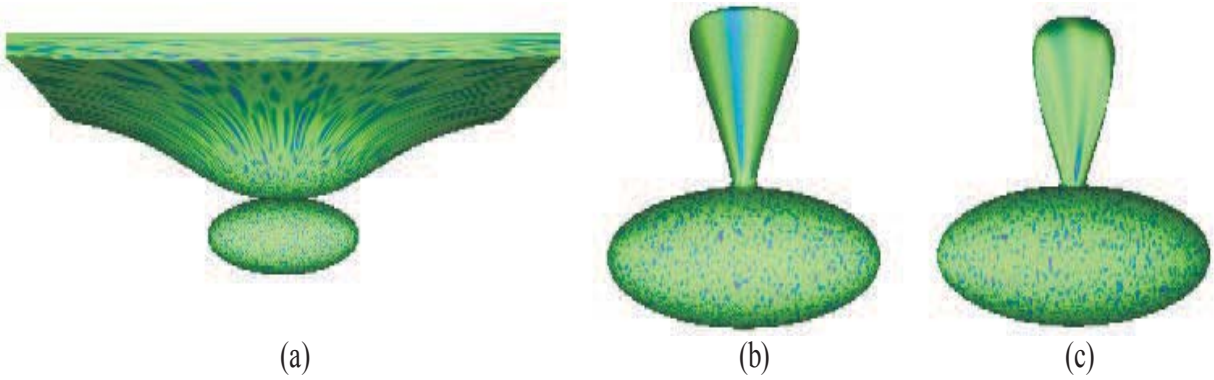


Figure 5.12: Shape driven deformation of an ellipsoid. Effect of applying the area of influence. The area of projection is a plane. (a) No area of influence is defined, resulting in an infinite deformation. (b),(c) The area of influence is defined respectively using a cone and a block function.

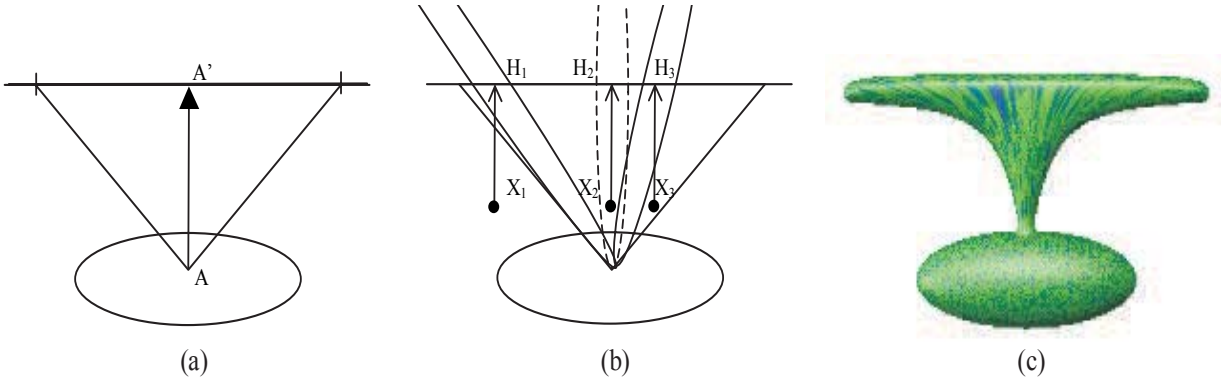


Figure 5.13: Shape driven deformation of an ellipsoid. Influence of the projection. The area of projection is a straight line. (a) Initial configuration. (b) Parallel projection of some points. (c) Resulting deformation.

5.4.3 Using FRep tree for deformations

In this subsection, our interest is turned towards the function Z defining the area of influence, and its corresponding mapping, recalled here:

$$\tilde{Z}(X) = \frac{1}{2}\left(1 + \frac{Z(X)}{\sqrt{(}\vartheta + Z^2(X)}\right) \tag{5.18}$$

The variable ϑ is an important feature in the behaviour of the deformation. In some sense, it can be compared to the hardness factor of the potential functions (eq. 5.8). Figure 5.14 shows a deformation of an ellipsoid with different values of ϑ. The Z function is defined as a simple block, where the vertical axis is the displacement vector. As one can see, as ϑ is small, as the resulting deformation is close to the area of influence with less blend, and as it gets bigger, as the deformation is smoother and blends more with the initial object.

Until now, the function Z defined a simple block, or a cone, and in the proposed examples, the area of influence was always aligned along an axis. Extension to arbitrary orientations is straightforward. Considering the displacement vector AA', one can apply a set of two inverse rotations and a translation to obtain the desired orientation and location. Furthermore, as the area of influence is defined as a FRep object, all the set of available operations and primitives can be combined to define it. These possibilities extend considerably the set of possible deformations. In the following, several examples are given to illustrate some of the offered possibilities.

Figure 5.15 shows a deformed ellipsoid, but in this case, the FRep constructive tree for Z is defined as an intersection of the block and a torus. Therefore, the deformation that can be achieved looks like a torus. To emphasise the result of the parameter ϑ, three different deformations are shown, with different values for ϑ. One important feature is that one can change the topology of the initial object using the proposed deformation method. None of the existing method based on forward mapping can handle this problem.

In the next example, Fig. 5.16, the area of influence is defined as a block composed with a tapering operation. From left to right, one can see the initial deformation, the deformation tapered along one axis, and then along both axes.

Despite of the topology change, the result is also interesting if one considers the attributes of the object, i.e., the texture in this case. In the torus example and for a given ϑ, the result is similar to a union of the initial object and a torus. However, one can notice that the texture is stretched along the deformation.

5.5 Using space mapping node in a FRep tree

One of the major advantages of defining a deformation as an inverse mapping is that it can be used as a new node in a FRep tree. For any given object, one can deform it using several deformations, and then use the resulting object as a new primitive for another tree. When modelling an object, the deformation node can be used at any level of the modelling stage. Several deformation nodes can be combined together. Figure 5.17 is given as a forestalling example. An ellipsoid is deformed using 4 deformations and is then combined with a cylinder and another ellipsoid using the set-theoretic intersection. A straightforward application is to deform an object along a curve while applying successive deformations.

Figure 5.14: Influence of the ϑ parameter in the $\tilde{Z}$ function. From left to right, $\vartheta < 1$, $\vartheta = 1$ and $\vartheta > 1$.

Figure 5.15: Changing the area of influence and ϑ. The Z function is defined as a FRep tree composed of an intersection of a block and a torus. The parameter varies in a similar way as in eq. 5.14.

Figure 5.16: Changing the area of influence. The Z function is defined as a FRep tree composed of a tapering operation and a block primitive. From left to right, the first figure shows the initial deformation, then a tapering operation along one axis, and then along two axes.

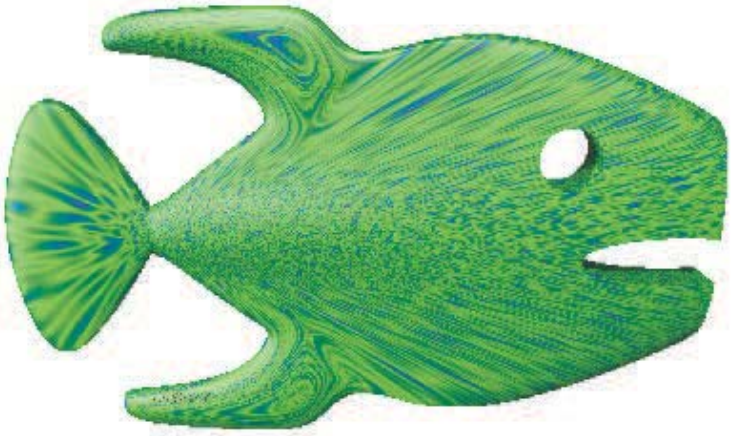


Figure 5.17: Space mapping as a new node in a FRep tree. Several deformations are applied to an ellipsoid and are then combined within a complex object using a FRep tree (intersection operation with a cylinder and another ellipsoid).

5.5.1 Deformations along a curve

There are different possible solutions to define a deformation along an arbitrary curve. In the following, we will consider that the curve is sampled, resulting into a set of connected segments, i.e., a broken line. One solution is to consider the set of segments as primitives for a convolution surface which can be then directly used in the definition of the area of influence Z. Each convolution segment is combined with another using the set-theoretic union operation. A bounding box can be deduced from this set of segments, as well as a displacement vector. Figure 5.18 is given as an example, where an ellipsoid is deformed along a broken line. Figure 5.18a shows the initial configuration, and the result when one uses convolution surface to deform the object is shown in Fig. 5.18b. This solution suffers from two main drawbacks. The first one is the behaviour of the attributes. Indeed, the texture does not follow the deformation, but is rather stretched along the displacement vector. In some cases, one may prefer to obtain a texture deformation that follows the broken line. Another limitation of the use of convolution surface is its computational cost. Indeed, if the input curve for the deformation is smoother than the one proposed in the example, i.e., based on a B-spline, to sample it sufficiently and to calculate all the corresponding convolution surfaces is too computationally expensive. One preferable solution is to apply a stack of deformations along the samples of the curve, i.e., a set of successive deformation nodes in the tree. They are defined using only a potential function and a distance function; the area influence is not used in this case. In the example of Fig. 5.18c, the result is deformation that follows smoothly the input curve, and as one can see, the texture pattern follows the path of the deformation.

Another reason to sample a curve rather than using a convolution surface is that it can handle the case where the curve intersects several times the object to be deformed. Consider for instance that one wants to deform an ellipsoid using a spiral curve as shown in Fig. 5.19. As one can see, the spiral intersects several time the ellipsoid. Figure 5.19a shows a vertical cut of the ellipsoid, where the spiral is shown in red, and Fig. 5.19b shows the whole object. The

Figure 5.18: Deformation along a broken line. (a) The object to be deformed and the input broken line. (b) Deformation using the area of influence, where the broken line is used to define a convolution line. Texture is stretched along the displacement vector AA. (c) Set of deformations. Each segment of the broken line is considered as a single deformation. The texture is stretched along each deformation, and thus follows the broken line.

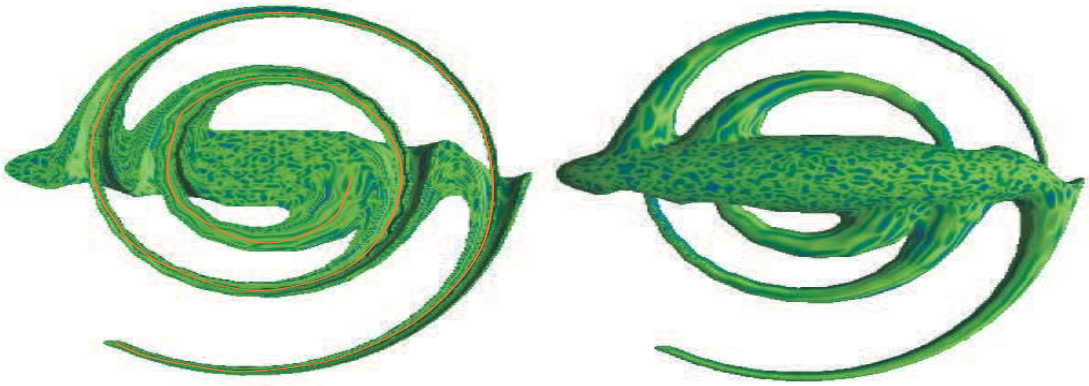


Figure 5.19: Self intersecting deformations. An ellipsoid is deformed using a spiral curve. Several intersections between the original shape and the deformation occur. Right picture shows the inside of the deformed object.

result is natural as the object follows the deformation. Here and there of the spiral, holes are created as the neighbouring points are also displaced. The resulting shape is similar to a shell.

5.5.2 Complex examples

Figure 5.20 provides an example where a single sphere is transformed using several different space mappings. Some parts are deformed using single control points, such as the nose, the eyes, the bumps on the top of the head, and the mouth. The straight horns at the bottom of the head are modelled using two deformations. The horns at the top of the head are modelled by sampling two B-spline curves. As one can see, deformations in arbitrary directions can be obtained.

Figure 5.21 shows the combination of a deformed object with another primitive. The central object is a sphere deformed by several space mappings. It is also deformed by a twisting

Figure 5.20: Example of the deformation using space mapping. Eyes, nose, mouth, bumps and bottom horns result from point to point based deformations, and top horns are obtained as sets of deformations along a sampled curve.

operation. As one can see, the shape driven deformations also follow the twist. A torus is added to this object using the union operation. We choose deliberately to apply only one deformation to the torus, identical to one of the deformations applied to the ellipsoid. The corresponding FRep tree is shown in Fig. 5.21(*left*).

Figure 5.22 is given to show that the deformation scheme we propose can be also used to carve object, and even if most of the examples show major deformation of an object, subtle details can be also defined. On the left part, an object has been carved (according to two parametric functions of Lissajous). On the right, an additional deformation has been applied. As one can see, the carved details also follow the deformation.

The last example is a vase, shown in Fig. 5.23. To model this object, we combined the constructive approach with the sculpting and deforming steps. First, the body was created using a B-spline object, and then its top was deformed. The next step was to combine it with an ellipsoid. Once both parts were combined, other deformations are achieved along two curves, such as they get close to the body. The final step was to create the handles of the vase. If the deformation step was the last one as it is usually the case, it may be difficult to find the correct location for the handles, but as we built the tree node by node, regardless of the nature of the operation, this task was easy.

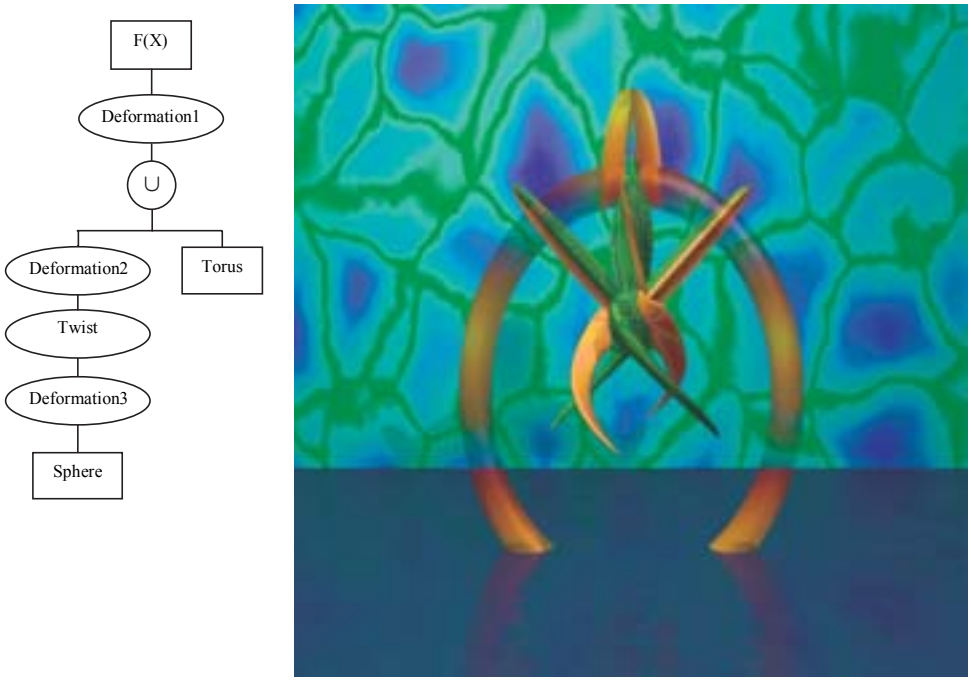


Figure 5.21: Example of the deformation using space mapping. A sphere is deformed, and combined with a torus. The tree shows the successive deformations, where the first one is applied to both sphere and torus primitives. A twisting operation is applied to the deformed sphere.

Figure 5.22: Deformation as a carving tool. (Left) A block is carved according to a path defined along a curve. (Right) An additional global deformation is then added, and the carving follows it.

Figure 5.23: Example of the shape-driven deformation. Constructive modelling, volume sculpting and deformations steps were performed in arbitrary order while designing the object.

5.6 Conclusion

In this chapter, we proposed different techniques for deforming constructive hypervolumes on the base of the FRep model. A first set of deformations is point based, as arbitrary points in the space are moved to arbitrary positions. Traditional field functions are then used to define the deformation. The second technique consists in moving arbitrary points too, but an area of influence for the deformation is defined, as well as a target area. A large number of new deformations can be obtained with this method, and it becomes possible to change easily the topology of the initial shape.

These techniques can be easily applied to traditional implicit objects. Nevertheless, the most intriguing results are achieved while using the constructive hypervolume model. Indeed, if the proposed deformations are applied only to the geometry of the object, similar visual results can be obtained with traditional set-theoretic and blending operations. The justification of the proposed work takes its full meaning when both attributes and geometry are considered. If the deformation node is included in the geometric and attribute trees, the deformation of geometry is followed by the corresponding deformation of attributes. The complex examples that have been given show that the texture intuitively follows the deformation of the objects geometry.

We choose to define the deformations as a new node in the FRep model. This brings the important feature that it becomes possible to combine several modelling approaches to define

objects. The first one was the constructive approach, inherent property of the FRep model, the second one was volume sculpting while using a node based on trivariate B-spline and Bézier functions. In this chapter, we proposed a different technique of deforming existing object with the use of different control shapes (points, segments, curves, surfaces, and solids), and corresponding field functions.

Conclusion

In this document, we introduced a general hypervolume model as a framework. The model includes the following components: hypervolume objects, operations, and relations. A hypervolume object is considered as a multidimensional point set with multiple attributes. Attributes represent different properties of real objects and processes such as material, photometric properties (colour, transparency, diffuse and specular reflections, etc.), physical properties (density, temperature, etc.), and other properties of an arbitrary nature. The function representation (FRep) is used as the basic model for point set geometry, and attributes are modelled independently using real-valued scalar functions of several variables. Geometry and attributes are modelled in a step-by-step manner using elementary primitives, operations and relations. This is reflected in the underlying representation in the form of the constructive tree. Each real-valued function defining geometry or an attribute is evaluated at the given point by a procedure traversing the constructive tree structure with primitives in the leaves, and operations and relations 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 modelling of space partitions for texturing and modelling of multidimensional textured objects. We discussed some operations specific for constructive solid texturing. The proposed approach allows for modelling, texturing and visualization of 3D solids, time-dependent and multidimensional objects in a completely uniform manner. The concept of constructive hypervolume textures is independent of the geometry representation. We provided examples of textured FRep and BRep objects as illustrations.

To model heterogeneous objects, we propose a new primitive and a new node for the FRep tree. Close to the volume sculpting metaphor, the proposed primitive, based on a trivariate B-Spline function, mimics the process of clay-sculpting. While modelling an object, one can freely add or remove material at the desired location. A multiresolution scheme based on a wavelet transform is also proposed to facilitate the modelling process. To provide sculpting at any level of the constructive tree, a general deformation node based on space mapping is proposed, where one can move arbitrary points to arbitrary positions to define the deformation.

A first set of deformations is point based, as arbitrary points in the space are moved to arbitrary positions. Traditional field functions are then used to define the deformation. The second technique consists in moving arbitrary points too, but an area of influence for the deformation is defined, as well as a target area. A large number of new deformations can be obtained with this method, and it becomes possible to change easily the topology of the initial shape.

The constructive hypervolume model can be extended in various directions. Similarly to the hybrid volume model, constructive hypervolumes 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 multidimensional voxel arrays in this new framework, applications of volume rendering as well as multiple-material rapid prototyping of modelled objects will be the subjects of our future research.

Another important research direction is the multidimensionality. A constructive hypervolume covers heterogeneous multidimensional objects. In this document, heterogeneous is in the sense of material and attributes. An interesting approach would be to consider an approach to modelling heterogeneous objects as multidimensional point sets with multiple attributes, i.e., consider real or abstract heterogeneous objects that have internal structure with non-uniform distribution of material and other attributes of an arbitrary nature and elements of different dimension. Multidimensional point sets with a fixed dimensionality and with multiple attributes can be quite effectively modelled by the constructive hypervolume model based on real-valued functions. The requirement of dimensional heterogeneity naturally brings the idea of adding a kind of cellular representation to the model. Moreover, different applications such as CAD or finite-element analysis require an explicit representation of mixed-dimensional objects along with the functional one. On the base of the constructive hypervolume model, another model, called *new hybrid cellular-functional model* has been proposed in [AKK$^+$02], and introduces the first step towards a general heterogeneous model. Our future work will try to extend those models.

Appendix A
Examples of HyperFun models

Example 1: A textured sphere

In this appendix, we provide some HyperFun examples. While the first example is defined using the previous version of the language, the others use the extended version of HyperFun supporting the constructive hypervolume model. Let us first consider a simple example of the HyperFun language that defines a sphere.

```
my_model(x[3],a[1]){
array center[3];
center=[0,0,0];
my_model=hfSphere(x,center,5);
}
```

The first line defines the name of the defining function, *my_model*, and is followed by two arrays. The first one, x, corresponds to the point coordinate, and the second one is used to pass additional parameters to the function (for multidimensional objects for instance). The second line declares an array called *center*, the third line initialises it, and the fourth line calls a built-in function defined in the HyperFun library, *hfSphere*, which corresponds to the defining function of a sphere. The conversion of the sphere example to the extended version of HyperFun leads to the HyperFun code given in Fig. 5.24.

In this simple example, an additional array s is added in the declaration of the function. It contains the attributes. In a rendering context, we use 13 different attributes. The following table shows the correspondence between the components of this array and the shading parameters. Note that the two coefficients *s[4]* and *s[5]* define the opacity. While the first one defines the amount of light filtered by the object, the second defines the amount of non-filtered light transmitted in the object.

The visual result of this example is shown in Fig. 5.24. The geometry of the object is a sphere. The texture is defined everywhere in the space using trigonometric functions. Any other functions can be easily used using the HyperFun language (conditional selection and iterative structures are available in the HyperFun language).

```
my_model(x[3],a[1],s[3]){
array center[3];
center=[0,0,0];

s = [0.0, 0.00, 0.0,0.0,0.0,0.25,0.25,0.25,0.8,0.8,0,0,0];

s[1]=(1+sin(x[1]))/2;
s[2]=(1+cos(x[2]))/2;
s[3]=0.5;

my_model=hfSphere(x,center,5);
}
```

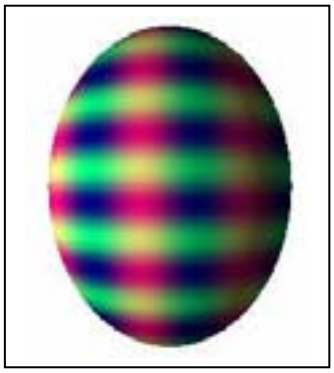


Figure 5.24: A simple sphere textured using trigonometric functions.

s[1]	Red component of an RGB colour
s[2]	Green component of an RGB colour
s[3]	Blue component of an RGB colour
s[4]	Filter component
s[5]	Transmittance
s[6]	Red component of the ambient coefficient
s[7]	Green component of the ambient coefficient
s[8]	Blue component of the ambient coefficient
s[9]	Diffuse coefficient
s[10]	Specular coefficient
s[11]	Red component of the reflectance coefficient
s[12]	Green component of the reflectance coefficient
s[13]	Blue component of the reflectance coefficient

Example 2: Space Partitions

The next examples show how to define a space partition of the object space. The geometry is defined using the following polynomial:

$$3 + 8(x^4 + y^4 + z^4) = 8(x^2 + y^2 + z^2) \tag{5.19}$$

The corresponding object is an algebraic surface, called *chmutov surface* in the literature, based on Chebyshevs polynomials of degree six, and is shown in Fig.5.25*a*. Let us define a simple space partition of the object space, with the use of three spheres. Figure 5.25*b* shows the initial object placed inside the space partition, and Fig. 5.25*c* shows the resulting object. To produce this example, the HyperFun code of Fig. 5.25 was used.

The next example shown in Fig. 5.26 is the famous washing-rag model, with a complex space partition. One can notice the use of the function *hfA_Union*, which is used to determine the colour of a given point when it belongs to more than one partition. The semantic of the function is :

```
tmp=hfA_Union(f1,f2,out,in1,in2,index);
```

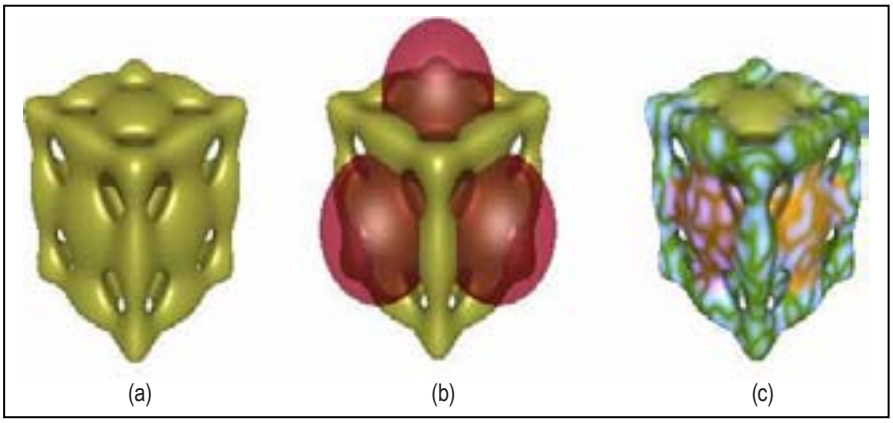
where:

- (*f1,f2*) are the function values of two partitions for a given point,
- (*out*) is the output array containing the resulting attribute values,
- (*in1,in2*) are the input arrays containing the attributes of the two space partitions,
- (*index*) is an index used to choose between predefined union operations, such as *min/max* functions, *sum/difference* operations and others.

Example 3: Deformations

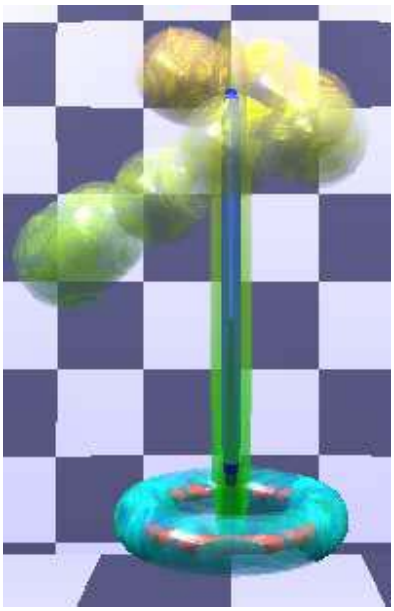
This last set of examples is given to show how the proposed deformation can be defined in the HyperFun language. Figures 5.27 and 5.28 show a point based deformation, and its corresponding HyperFun model. The sample source and target points are used. In the first Fig. 5.27, a simple deformation using an exponential potential function is used, combined with an ellipsoidal distance function. Two different results are shown, one where the texture pattern follows the deformation, and another independent of the deformation. The texture pattern is based on Perlins noise function. Then, if one wants to see a texture that follows the deformation, the call to this function uses the point coordinate that has been modified; otherwise, the original point coordinate is used.

The example given in Fig. 5.28 shows how to change the orientation of the deformation. The sample displacement vector has been used as in Fig. 5.27, the distance function is now superellipsoidal based. We choose to map the displacement vector on the z axis, in order to obtain a deformation oriented arbitrary (compare the result of Fig. 5.27). Then, after calculating different angles, a set of rotations is sequentially applied to obtain the desired orientation. As one can see, the HyperFun language is rich enough to define complex functions.



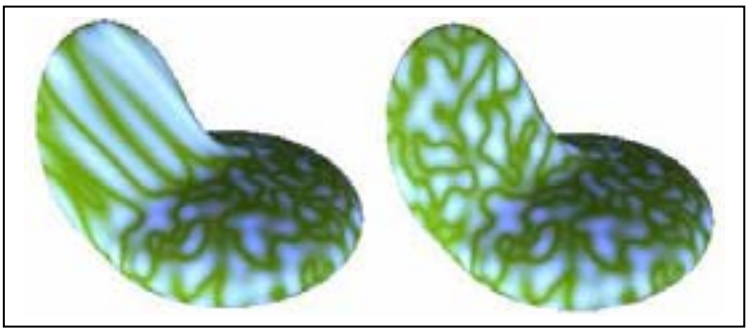

```
my_model(x[3], a[1],s[13]){
array xt[3], center[3], c3[3];
--definition of chmutov
--polynomial
x2=x[1]*x[1];
y2=x[2]*x[2];
 z2=x[3]*x[3];

xa=(3-4*x2)*(3-4*x2)*x2 ;
ya=(3-4*y2)*(3-4*y2)*y2;
 za=(3-4*z2)*(3-4*z2)*z2;

--chebichev surface
model=3-2*(xa+ya+za);

if(model>=0) then
        --default attribute values
        s=[0.0,1.0,1.0, 0.0, 0.0, 0.35,0.35,0.35,  0.7,   0.01, 0.0,0.0,0.0];
        --Noise for texture, based on Perlin's noise function
        noise=hfA_NoiseP(xt,1);

        --definition of the RGb colour vector
        s[1]=0.4+noise;       s[2]=0.6+noise;       s[3]=4*noise;

        --definition of the space partition : 3 disjoint spheres
        radius = 0.75;          center=[1,0,0];
        sphere1=hfSphere(x,center,radius);
        --the colour inside this sphere is a permutation of the default colour
        if(sphere1>=0) then
                s[2]=0.4+noise;       s[1]=0.6+noise;       s[3]=4*noise;
        endif;

        center=[0,0,1];
        sphere3=hfSphere(x,center,radius);
        --The colour inside the second sphere is the max of some given value with the
        --default colour values (based on a noisy function)
        if(sphere2>=0) then
                s[1]=max(0.89,s[1]);              s[2]=max(0.18,s[2]);            s[3]=max(0.1,s[3]);
        endif;

         center=[0,1,0];
         sphere3=hfSphere(x,center,radius);
        --The colour inside the sphere is a blend of a constant yellow colour with
        --the default colour.
        --yellow colour
        c3[1]=0.78;           c3[2]=0.78;         c3[3]=0.3;
        --mapping of the function value of the sphere in the interval [0,1]
        sp3=0.5*(1+sphere3/sqrt(0.15+sphere3*sphere3));

        --linear interpolation
         s[1]=s[1]*(1-sp2)+c2[1]*sp2;
        s[2]=s[2]*(1-sp2)+c2[2]*sp2;
        s[3]=s[3]*(1-sp2)+c2[3]*sp2;
endif;
my_model = model;
}
```

Figure 5.25: Chebyshev algebraic surface of sixth order defined using a HyperFun model. (a) The object geometry. (b) Initial object placed in the space partition defined of three spheres. (c) Resulting textured object.



```
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 af 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;
        --Colour 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 colour 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 5.26: Complex space partitioning for texturing defined with a HyperFun model.



```
my_model(x[3], a[1],s[13]){

array xt[3];
array center[3];
array pt_from[3];
array pt_to[3];

--source and target points
pt_from=[0,0,0];
pt_to=[6,6,0];

--Radii of influence
ra=1;
rb=1;
rc=2;
--Ellipsoidal distance function
gamma=  (x[1]-pt_to[1])*(x[1]-pt_to[1])/(ra*(1+(pt_to[1]-pt_from[1])^2))+
      (x[2]-pt_to[2])*(x[2]-pt_to[2])/(rb*(1+(pt_to[2]-pt_from[2])^2))+
      (x[3]-pt_to[3])*(x[3]-pt_to[3])/(rc*(1+(pt_to[3]-pt_from[3])^2));
--Potential function
b=2;
pot=exp(-b*gamma);
--Displacement
dx=pot*(pt_to[1]-pt_from[1]);
dy=pot*(pt_to[2]-pt_from[2]);
dz=pot*(pt_to[3]-pt_from[3]);
--inverse mapping
xt[1]=x[1]-dx;
xt[2]=x[2]-dy;
xt[3]=x[3]-dz;

--Original shape
center=[0,0,0];
ell=hfEllipsoid(xt,center,7.0,2,7);

      s=[0.0,1.0,1.0, 0.0, 0.0, 0.35,0.35,0.35,  0.7,   0.01, 0.0,0.0,0.0];
--Texture pattern
            --The deformation influence the pattern (fig left)
            noise=hfA_NoiseP(xt,1);
            --The deformation do not influence the pattern (fig right)
            --noise=hfA_NoiseP(x,1);

s[1]=0.4+noise;
s[2]=0.6+noise;
s[3]=4*noise;
my_model = ell;
}
```

Figure 5.27: Simple point based deformation, using an ellipsoidal distance function and an exponential potential function. A simple parameter decides whether the texture pattern follows the deformation or not.

```
my_model(x[3], a[1],s[13])
{
array xt[3], center[3], v[3], r[3],pt_from[3], pt_to[3];
--Source and target points
pt_from=[0,0,0];        pt_to=[6,6,0];

--Size of the displacement
norm = sqrt((pt_to[1]-pt_from[1])*(pt_to[1]-pt_from[1])+
            (pt_to[2]-pt_from[2])*(pt_to[2]-pt_from[2])+
            (pt_to[3]-pt_from[3])*(pt_to[3]-pt_from[3]));
r[1]=1;      r[2]=1;     r[3]=norm;

--temporary point coordinate
xt[1]=x[1];xt[2]=x[2];xt[3]=x[3];

--Displacement Vector
V[1]=(pt_to[1]-pt_from[1]);        V[2]=(pt_to[2]-pt_from[2]);        V[3]=(pt_to[3]-pt_from[3]);

--rotation : Mapping of V[3] onto x[3]
theta = atan2(V[2],V[1]);
norm=sqrt(V[1]*V[1]+V[2]*V[2]);
phy=atan2(norm,V[3]);

--rotation of the temporary point coordinate xt

tmp=hfRotate3DZ(xt,theta);
tmp=hfRotate3DY(xt,phy);

--Superellipsoidal distance function. Rotation of the corresponding superellipsoid
center[1]=pt_to[1];    center[2]=pt_to[2];    center[3]=pt_to[3];

tmp=hfRotate3DZ(center,theta);
tmp=hfRotate3DY(center,phy);

n1=0.4;    n2=2;
dist=1-hfSuperell(xt,center,r[1],r[2],r[3],n1,n2);
d2=sqrt(dist*dist);

--Potential function
p=0.1;      b=2;
pot=exp(-b*d2);

--Displacement
dx=pot*(pt_to[1]-pt_from[1]);        dy=pot*(pt_to[2]-pt_from[2]);        dz=pot*(pt_to[3]-pt_from[3]);

xt[1]=x[1]-dx;                       xt[2]=x[2]-dy;                       xt[3]=x[3]-dz;

--Original shape
center=[0,0,0];
ell=hfEllipsoid(xt,center,7.0,4,7);

--Attributes
 s=[0.0,1.0,1.0, 0.0, 0.0, 0.35,0.35,0.35,  0.7,   0.01, 0.0,0.0,0.0];
noise=hfA_NoiseP(xt,1);
s[1]=0.4+noise;        s[2]=0.6+noise;        s[3]=4*noise;

my_model = ell;
}
```

Figure 5.28: Point based deformation. A superellipsoidal distance function is used. To obtain an arbitrary oriented deformation, a set of rotations is applied.

Bibliography

[ACF+99] V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, and V. Savchenko. Hyperfun project: a framework for collaborative multidimensional f-rep modelling. *Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop, J. Hughes and C. Schlick (Eds.)*, pages 55–69, September 1999.

[AKK+02] V. Adzhiev, E. Kartasheva, T. Kunii, A. Pasko, and B. Schmitt. Cellular-functionnal modelling of heterogeneous objects. *Solid Modeling and Applications'02, K. Lee and M. Patrikalakis (Eds.)*, pages 192–203, September 2002.

[AKPS00] V. Adzhiev, M. Kazakov, A. Pasko, and V. Savchenko. Hybrid system architecture for volume modelling. *Computers And Graphics, Pergamon Press*, 24(1):67–78, 2000.

[AS96] R. Avila and L. Sobierajski. A haptic interaction method for volume visualization. *IEEE Visualization '96, Yagel R. and Nielson G. (Eds.), IEEE Computer Society Press*, pages 197–204, 1996.

[ATTY99] H. Arata, Y. Takai, N. Takai, and T. Yamamoto. Free-form shape modelling by 3d cellular automata. *Modelling International '99, IEEE Computer Society Press*, pages 242–247, 1999.

[Bae98] J. Baerentzen. Octree-based volume sculpting. *IEEE Visualization '98, Late breaking hot topics proceedings*, page 1998, 1998.

[Bar81] A.H. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981.

[Bar84] A.H. Barr. Global and local deformations of solid primitives. *Proceedings of SIGGRAPH '84, Computer Graphics*, 18(3):21–30, 1984.

[BB91] P. Borrel and D. Bechmann. Deformations of n-dimensional objects. *Internat. J. Comput. Geom. App.*, 1(4):427–453, 1991.

[Bec94] D. Bechmann. Space deformation models survey. *Computer and Graphics*, 18(4):571–586, 1994.

[BGS94] C. Blanc, P. Guitton, and C. Schlick. A methodology for description of geometrical deformations. *Pacific Graphics'94 Proceedings, Beijing, China*, pages XX–XX, August 1994.

[BL95] J.R. Bill and S. Lodha. Sculpting polygonal models using virtual tools. *Graphics Interface'95, Morgan Kaufmann Publihsers*, pages 272–278, 1995.

[Bli78] J. Blinn. Simulation of wrinkled surfaces. *Computer graphics, SIGGRAPH'78*, 12(3):286–292, 1978.

[Bli82] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[BN76] J. Blinn and M.E. Newell. Texture and reflection in computer generated images. *Communications of ACM*, 10(19):542–547, 1976.

[BN92] T. Beier and S. Neely. Feature-based image metamorphosis. *SIGGRAPH'92 Proceedings, ACM Press*, pages 35–42, 1992.

[Bow95] A. Bowyer. Svlis: Introduction and user manual. *Information Geometers, UK, 128 p.*, 1995.

[BPRS98] C. Bajaj, V. Pascucci, G. Rabbiolo, and D. Schikore. Hypervolume visualization: a challenge in simplicity. *IEEE Symposium on Volume Visualization, ACM SIGGRAPH*, pages 95–102, 1998.

[BR94] P. Borrel and A. Rappoport. Simple constrained deformations for geometric modeling and interactive design. *ACM Transactions on Graphics*, 13(2):137–155, 1994.

[BS91] J. Bloomenthal and K. Shoemake. Convolution surfaces. *SIGGRAPH'91 Proceedings*, 25(4):251–256, 1991.

[BS95] C. Blanc and C. Schlick. Extended field functions ofr soft objects. *Implicit Surfaces '95, Eurographics/ACM SIGGRAPH Workshop*, pages 21–32, 1995.

[BS96] C. Blanc and C. Schlick. Ratioquadrics: an alternative model for superquadrics. *The visual computer*, 12, 1996.

[Cat74] E. Catmull. A subdivision algorithm for computer display of curved surfaces. *PhD thesis, Department of Computer Science, University of Utah*, December 1974.

[Coo84] R. Cook. Shade trees. *Computer Graphics, (SIGGRAPH'84 Proceedings)*, 18(3):223–232, 1984.

[Coq90] S. Coquillart. Extended free-form deformation : A sculpting tool for 3d geometric modelling. *Computer Graphics, (SIGGRAPH'90 Proceedings*, 24(4):187–193, 1990.

[CR94] Y.K. Chang and A.P. Rockwood. A generalised de casteljau approach to 3d free-form deformation. *SIGGRAPH'94 proceedings*, pages 257–260, 1994.

[Cre98] B. Crespin. Modélisation et déformation de forme libre à base de surfaces splines équipotentielles. *PhD thesis, Bordeaux University I*, 1998.

[CT98] M. Chen and J. Tucker. Constructive volume geometry. *Technical Report CS-TR-98-19, University of Wales Swansea, UK*, page 36, 1998.

[CT00] M. Chen and J. Tucker. Constructive volume geometry. *Computer Graphics Forum*, 19(4):281–293, 2000.

[DSS95] T.D. DeRose, E.J. Stollnitz, and D.H. Salesin. Wavelets for computer graphics : A primer, part 2. *Computer Graphics and Applications*, pages 75–85, Jully 1995.

[Ea98] D. Ebert and al. Texturing and modelling: a procedural approach. *AP Professional, San Diego*, 1998.

[EG01] G. Elbert and C. Gostman. *Multiresolution Control for Nonuniform B-Spline Curve editing.* CRC-HP Israel Science Center, 2001.

[EKL+91] J. Ellis, G. Kedem, T. Lyerly, D. Thielman, R. Marisa, J. Menon, and H. Voelckerand. Tehray casting engine and ray representations. *Proceedings of Symposium on solid modeling foundations and CAD/CAM applications, ACM press*, pages 255–267, 1991.

[Far90] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Second Edition, Academic Press, 1990.

[FB88] R.D. Forsey and R.H. Bartels. Hierarchical bspline refinement. *Computer Graphics*, 22(4):205–211, 1988.

[FCG00] E. Ferley, M.-P. Cani, and J.-D. Gascuel. Practical volumetric sculpting. *Visual Computer*, 16(8):469–480, 2000.

[FDFH95] J.D. Foly, A. Van Dam, S.K. Feiner, and J.F. Hugues. *Computer Graphics: Principles and Practices.* Second Edition, ADDISON-WESLY, ISBN: 0-201-84840-6, 1995.

[Fer02] E. Ferley. Sculpture virtuelle. *PhD Thesis, iMAGIS-GRAVIR laboratory, Grenoble, France*, 2002.

[FK97] A.T. Fomenko and T.L. Kunii. Topological modeling for visualization. *Springer-Verlag, Tokyo and Heidelbrg*, 1997.

[Gas93] M.P. Gascuel. An implicit formulation for precise contact modeling between flexible solids. *Computer Graphics*, 27(4):313–320, 1993.

[GH91] T. Galyean and J. Hughes. Sculpting: an interactive volumetric modelling technique. *SIGGRAPH '91, Computer Graphics Proceedings*, 25(4):267–274, 1991.

[Gla95] A.S. Glassner. *Principles of Digital Image Synthesis.* Morgan Kauffmann, USA, 1995.

[GMR99] A. Gomes, A. Middleditch, and C. Reade. A mathematical model for boundary representations of n-dimensional geometric objects. *Fifth Symposium on Solid Modelling and Applications, W. Bronsvoort and D. Anderson (Eds.), ACM Press*, pages 270–277, 1999.

[GOP99] C. Gonzales-Ochoa and J. Peters. Localized-hierarchy surface spline (less). *ACM Symposium on Interactive 3D Graphics, ISBN 1-584-13-0821*, pages 7–16, April 1999.

[Gri99] L. Grisoni. Elements de multiresolution en modelisation geometrique (elements of multiresolution in geometric modeling). *PhD thesis, LaBRI, Bordeaux I University*, December 1999.

[HB84] S. Haruyama and B.A. Barsky. Using stochastic modelling for texture generation. *IEEE Computer Graphics and Applications*, 4(3):7–19, 1984.

[HFP90] K.H. Hohne, H. Fuchs, and S. Pizer. 3d imaging in medicine: Algorithms, systems, applications. *NATO Advanced Science Institutes Series, Series F, Computer and Systems Science*, 60, 1990.

[Hou94] S. Houlding. 3d geoscience modelling - computer techniques for geological characterization. *SIGGRAPH '91, Computer Graphics Proceedings*, 1994.

[Hug92] J. Hughes. Scheduled fourier volume morphing. *SIGGRAPH '92, Computer Graphics*, 1992.

[JC94] M. Jones and M. Chen. A new approach to the construction of surfaces from contour data. *Computer Graphics Forum*, 13(3), 1994.

[JLP98] X. Jin, Y.F. Li, and Q. Peng. General constrainde deformations based on generalized metaballs. *Proceedings of Pacific Graphics'98*, pages 115–124, 1998.

[Jon96] M. Jones. The production of volume data from triangular meshes using voxelization. *Computer Graphics Forum*, 15(5), 1996.

[JW88] D. Jevans and B. Wyvill. Ray tracing implicit surfaces. *Technical report 88/292/04, University of Calgary*, 1988.

[Kar99] E. Kartasheva. Reduction of h-genus polyhedrons topology. *International Journal of Shape Modeling*, 2(5), 1999.

[KAW91] Z. Kasic-Alesic and B. Wyvill. Controlled blending of procedural implicit surfaces. *Graphics Interface'91*, pages 236–245, 1991.

[KBDH99] V. Kumar, D. Burns, D. Dutta, and C. Hoffmann. A framework for object modeling. *Computer-Aided Design*, 31(9):541–546, 1999.

[KCY93] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *IEEE Computer*, 26(7), 1993.

[KD97] V. Kumar and D. Dutta. An approach to modeling multi-material objects. *Fourth Symposium on Solid Modeling and Applications, ACM SIGGRAPH*, pages 336–345, 1997.

[LC87] W.E. Lorensen and H.E. Cline. Marching cubes : A high resolution 3d surface construction algorithm. *Computer Graphics, Siggraph*, 21(4):163–196, july 1987.

[LCJ94] F. Lazarus, S. Coquillart, and P. Jancène. Axial deformations: an intuitive deformation technique. *Computer Aided Design*, 26(8):607–613, 1994.

[LGL95] A. Lerios, C.D. Garfinkle, and M. Levoy. Feature-based volume metamorphosis. *SIGGRAPH'95, Computer Graphics Proceedings*, pages 449–456, 1995.

[Loh97] R. Lohner. Automatic unstructured grid generators. *Finite Elements in Analysis and Design*, 25:11–134, 1997.

[Man88] M. Mantyla. An introduction to solid modeling. *Computer Sciences Press, College Park, MD*, 1988.

[May] Maya. *Alias-WaveFont*. http:// www.aliaswavefront.com/.

[MI87] S. Murakami and H. Ichihara. On a 3d display method by metaball technique. *Electronics communications*, v70D(8):1607–1615, 1987.

[Mik96] M. Mikita. 3d free-form deformation: Basic and extended algorithms. 12^{th} *Spring Conference on Computer Graphics, W. Purgathofer editors, Comenius University, Bratisalava*, pages 183–191, 1996.

[MJ96] R. MacCracken and K.I. Joy. Free-form deformation with lattices of arbitrary topology. *SIGGRAPH'96 Proceedings*, pages 181–188, 1996.

[MK85] A. Middleditch and Sears K. Blend surfaces for set-theoretic volume modeling systems. *Computer Graphics, SIGGRAPH'95*, 19(3):161–170, 1985.

[MMZ94] J. P. Menon, R. Marisa, and J. Zagajac. More powerful solid modelling through ray representations. *IEEE Computer Graphics and Applications*, 14(3):22–35, 1994.

[MPS96] K. Miura, A. Pasko, and V. Savchenko. Parametric patches and volumes in the functional representation of geometric solids, set-theoretic solid modeling: Techniques and applications. *Proceedings CSG 96 (Winchester, UK, 17-19 April 1996), Information Geometers, UK*, pages 217–231, 1996.

[MS98] J. McCormack and A. Sherstyuk. Creating and rendering convolution surfaces. *Computer Graphics Forum*, 17(2):113–120, 1998.

[NC99] F. Neyret and M.P. Cani. Pattern-based texturing revisited. In *SIGGRAPH 99 Conference Proceedings*, pages 235–242. ACM SIGGRAPH, Addison Wesley, August 1999.

[NF91] D. Ney and E. Fishman. Editing tools for 3d medical imaging. *IEEE Computer Graphics and Applications*, 11(6):63–71, 1991.

[NHK+85] H. Nishimura., M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modelling by distributed function and a method of image generation. *Transactions of IECE of Japan*, J68-D(4):718–728, 1985. (in Japanese).

[Nie93] G. Nielson. Scattered data modelling. *IEEE Computer Graphics and Applications*, 13(1):60–70, 1993.

[Nie00] G. Nielson. Volume modelling. *Volume Graphics, M. Chen, A. Kaufman, R. Yagel (Eds.), Springer-Verlag*, pages 29–48, 2000.

[NN94] T. Nishita and E. Nakamae. A method for displaying metaballs by using bezier clipping. *Computer Graphics Forum*, 13(3):271–280, 1994.

[OF00] N. Ozawa and I. Fujishiro. A morphological approach to volume synthesis of weathered stones. *Volume Graphics, M. Chen, A. Kaufman, R. Yagel (Eds.), Springer-Verlag*, pages 367–378, 2000.

[Par77] R. Parent. A system for sculpting 3d data. *Computer Graphics*, 11(8):138–147, 1977.

[Pas88] A. Pasko. Conceptual and instrumental tools for direct method of multidimensional geometric problems solving with a computer. *PhD thesis, Moscow Engineering Physics Institute, Moscow*, 1988.

[PASS95] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modelling: concept, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995.

[PASS02] A. Pasko, V. Adzhiev, B. Schmitt, and C. Schlick. Constructive hypervolume modelling. *Graphical Models, Special issue on volume modeling*, 63:413–442, 2002.

[Pea85] D.R. Peachey. Solid texturing of complex surfaces. *SIGGRAPH '85, Computer Graphics, USA, ACM Press*, 19(3):279–286, 1985.

[Ped85] H. Pedersen. Decorating implicit surfaces. *SIGGRAPH'95, , Annual Conference Series, ACM Press, USA*, pages 291–300, 1985.

[Per85] K. Perlin. An image synthesizer. *Computer Graphics, ACM Press, USA*, 19(3):287–296, 1985.

[Pov] PovRay. *the Persistance of Vision*. http:// www.povray.org/.

[PPIK02] G. Pasko, A. Pasko, M. Ikeda, and T. Kunii. Bounded blending operations. *Shape modeling international, IEEE Computer Society, Banff, Canada*, pages 95–103, May 2002.

[PPP88] A. Pasko, V.V. Pilyugin, and V.V. Pokrovskiy. Geometric modelling in the analysis of trivariate functions. *Computers and Graphics*, 12(3/4):457–465, 1988.

[Pro] HyperFun Project. Language and Software for FRep Modelling, http::/www.hyperfun.org.

[PS94] A. Pasko and V. Savchenko. *Blending operations for the functionally based constructive geometry, Set-theoretic solid modeling: technique and applications*. CSG'94 conference proceedings, Information geometers, Winchester, UK, 1994.

[PSAS93] A. Pasko, V. Savchenko, V. Adzhiev, and A. Sourin. *Multidimensional geometric modelling and visualization based on the function representation of objects*. Technical Report 93-1-008, 1993.

[PT92] B. Payne and A. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, 12(1):65–71, 1992.

[RE99] A. Raviv and G. Elber. *Three dimensional freeform sculpting via zero sets of scalar trivariate functions.* Technical Report CIS9903, 1999.

[Req80] A. Requicha. Representations for rigid solids: theory, methods, and systems. *ACM Computing Surveys*, 12(4):437–464, 1980.

[Ric73a] A. Ricci. A constructive geometry for computer graphics. *The computer journal*, 16(2):157–160, 1973.

[Ric73b] A. Ricci. A constructive geometry for computer graphics. *The Computer Journal*, 16(2):157–160, 1973.

[RM95] D. Ruprecht and H. Mueller. Spatial free form deformation with scattered data ¿ interpolation methods. *Computers and Graphics*, 19(1):63–71, 1995.

[Rva63] V.L. Rvachev. On the analytical description of some geometric objects. *Reports of Ukrainian Academy of Sciences*, 153(4):765–767, 1963.

[SF98] K. Singh and E. Fiume. Wires: A geometric deformation technique. *SIGGRAPH'98*, pages 405–414, 1998.

[Sha88] V. Shapiro. Theory of r-funtions and applications: a primer. *TR CPA88-3, Cornell University*, 1988.

[Sha94] V. Shapiro. Real functions for representation of rigid solids. *Computer Aided Geometric design*, 11(2):153–175, 1994.

[Sha01] V. Shapiro. Solid modeling. *Handbook of Computer Aided Geometric Design, Elsevier Science Publishers (to be published*, 2001.

[SKPS00] B. Schmitt, M. Kazakov, A. Pasko, and V. Savchenko. Volume sculpting with 4d spline volumes. *CISST'2000*, 2:475–483, September 2000.

[Sny92] J. Snyder. *Generative Modelling for Computer Graphics and CAD*. Academic Press, 1992.

[SP86] T.W. Sederberg and S.R. Parry. Free-form deformations of solid geometric models. *Computer Graphics (SIGGRAPH'86 proceedings)*, 20(4):151–160, 1986.

[SP98] V. Savchenko and A. Pasko. Transformation of functionnaly defined shapes by exented space mapping. *The Visual Computer*, 14(5/6):257–270, 1998.

[SPAS01] B. Schmitt, A. Pasko, V. Adzhiev, and C. Schlick. Constructive texturing based on hypervolume modeling. *The Journal of Visualization and Computer Animation*, 12:297–310, 2001.

[SPKS95] V. Savchenko, A. Pasko, T. Kunii, and A. Savchenko. Feature based sculpting of functionally defined 3d geometric objects. *Multimedia Modeling. Towards Information Superhighway, T.S.Chua, H.K.Pung and T.L.Kunii (Eds.), World Scientific, Singapore*, pages 341–348, 1995.

[SPS99] B. Schmitt, A. Pasko, and V. Savchenko. Extended space mapping with bézier patches and volumes. *Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop, J. Hughes and C. Schlick (Eds.)*, pages 25–31, September 1999.

[SPS01] B. Schmitt, A. Pasko, and C. Schlick. Constructive modelling of frep solids using spline volumes. *Sixth ACM Symposium on Solid Modeling and Applications, D. Anderson, K. Lee (Eds.), ACM Press*, pages 321–322, 2001.

[SS96] J.P. Smets-Solanes. Vector field based texture mapping of animated implicit objects. *Computer Graphics Forum*, 15(3):289–300, 1996.

[TW99] M. Tigges and B. Wyvill. A field interpolated texture mapping algorithm for skeletal implicit surfaces. *Computer Graphics International, CGI'99, IEEE Computer Society*, pages 25–33, 1999.

[UO91] K. Udupa and D. Odhner. Fast visualization, manipulation, and analysis of binary volumetric objects. *IEEE Computer Graphics and Applications*, 11(6):53–62, 1991.

[Ups90] S. Upstill. *The Renderman Companion*. Addison-Wesley, 1990.

[WB96] 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: Techniques and Applications, Information Geometers, UK*, pages 359–376, 1996.

[Wel97] H.G. Wells. *The Invisible Man*. original ISBN 1-58734-078-X, 1897.

[WGG99] B. Wyvill, E. Galin, and A. Guy. The blobtree. warping, blending and boolean operations in an implicit surface modelling system. *Computer Graphics Forum*, 18(2):149–158, 1999.

[Win] A.S. Winter. *The vlib Web Site*. http://vg.swan.ac.uk/vlib/.

[WK93] S. Wang and A. Kaufman. Volume sampled voxelization of geometric primitives. *IEEE Symposium on Volume Visualization, Los Alamos*, pages 78–84, 1993.

[WK95] S. Wang and A. Kaufman. Volume sculpting. *Symposium on Interactive 3D Graphics, ACM Press*, pages 151–156, 1995.

[WMW86] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

[WT90] G. Wyvill and A. Trotman. Ray tracing soft objects. *New trends in computer graphics, Proceedings of CG International'90*, pages 467–476, 1990.

[ZGVdF97] R. Zonenschein, J. Gomes, L. Velho, and L. Henrique de Figueiredo. Texturing implicit surfaces with particle systems. *Visual Proceedingsn SIGGRAPH'97*, pages 172–184, 1997.