Formal Specification for Free-Form Curves and Surfaces[1]

Laurent Fuchs, Dominique Bechmann, Yves Bertrand and Jean-François Dufourd

L.S.I.I.T.
URA CNRS 1871,
Université Louis Pasteur,
7, rue René Descartes,
67084 F-Strasbourg
[fuchs,bechmann,bertrand,dufourd]@dpt-info.u-strasbg.fr

Abstract. This paper presents a formal description of curves and surfaces that have evaluation algorithms using control points. The frame of the algebraic specification allows us to unify most points of view by using modularity, genericity and ordered sorts. An experimentation prototype based on a translation of this specification into a typed functional language, namely ML, is currently developed.

Keywords. Free-form curves and surfaces, Bézier curves, Splines, Triangular surfaces, Algebraic specifications, Functional programming.

1 Introduction

The development of unifying theories of curves and surfaces models, such as urn models [1] or simplicial algorithms [5] allows mathematicians and designers to integrate a lot of models of curves, surfaces and even volumes — we will say varieties — into a single system. In order to do so, several authors suggest to build an object-oriented framework [10, 14]. In such an approach, the primary task is to identify the objects classes and the associated operations. This stage in program building process involves crucial decisions which are connected with the final functionalities and the data structures.

In this paper, we propose a new approach to deal with this primary step of system design. From varieties mathematical description, we try to identify and to describe the basic operations needed to build them before to choose any concrete data structures and programming language.

In order to do that, we use a formal specification method which has lead to significant results in computer graphics [12, 13]. More precisely, we use the algebraic specification framework, a well know methodology [15], which has produced good results in geometric modeling, particularly in topology [3].

With the help of this method we describe a modules hierarchy specifying various models of varieties. These modules use ordered sorts and parameterization, so we can work at any dimension with several models theories. Furthermore, a new theory can be easily added to the modules hierarchy. At this abstract level we also describe the basic operations for interactively building and manipulating varieties.

In a second step of system design, concrete data structures have to be chosen. To do that, we can select a programming language, such as a functional or an object-oriented one. But, before developing a real size system founded on this formal method,

[1]This research is supported by the *GDR-PRC de Programmation* and *GDR-PRC Algorithmique, Modèles et Infographie* (French CNRS).

it seems appropriate to well study high level functionalities to hide tedious basic manipulations. Therefore to validate our formal approach and to discover and implement good functionalities, we have realized a prototype system that present succinctly .

Section 2 introduces some notations and the general simplicial algorithm concept. Section 3 presents the basic specifications of curves and surfaces models. Section 4 shows how to work in the specification frame with particular curve or surface models. Section 5 presents our prototype and its functionalities. Section 6 concludes with some remarks and future works.

2 Curves and Surfaces Models

An unique specification is built for both BARRY & GOLDMAN's [1] Urn Models and DAHMEN, MICHELLI & SEIDEL's [5] Simplicial Algorithms. Since the presented specifications can describe any variety which is defined by control points and recursive evaluation algorithm, any theory with these characteristics can be integrated as well. It should be noted that describing polynomial curves and surfaces with control points and recursive evaluation algorithm captures most usual polynomial basis [9] and, in the case of curves, seems to be a very general process [4].

2.1 Notations

In the remain of the text, d refers to the parameter space dimension and p to the dimension of the control points space. Following notations defined in [5], the *barycentric coordinates* $\tau = (\tau_0, \ldots, \tau_d)$ of point $t \in \mathbb{R}^d$, with respect to a set $V = \{v_0, \ldots, v_d\}$ of affinely independent points in $\mathbb{R}^d$, are defined by $\tau_i = d_i(V|t)/d(V)$ where

$$d(V) = det\begin{pmatrix} 1 & \cdots & 1 \\ v_0 & \cdots & v_d \end{pmatrix} \quad \text{and} \quad d_i(V|t) = det\begin{pmatrix} \cdots & 1 & 1 & 1 & \cdots \\ \cdots & v_{i-1} & t & v_{i+1} & \cdots \end{pmatrix}.$$

We also use the *multinomial indexes* : $\alpha = (\alpha_0, \ldots, \alpha_d)$ with $\alpha \in \Gamma_k$, where $\Gamma_k = \{\alpha \in \mathbb{Z}_+^{d+1}; |\alpha| = \alpha_0 + \cdots + \alpha_d = k\}$. Eventually, let $e_i = (\delta_{i,j})_{j=0}^d$, $i = 0, 1, \ldots, d$, denote the i^{th} element of the canonical basis of $\mathbb{Z}^{d+1}$, where $\delta_{i,j}$ is the Kronecker symbol: $\delta_{i,j} = 1$ if $i = j$, 0 otherwise.

2.2 General simplicial algorithm

Let us consider evaluation algorithms of varieties that can be put in the following general form, called *general simplicial algorithm*:

$$\begin{aligned} P_\alpha^0(t) &= P_\alpha, \quad \text{for} \quad \alpha \in \Gamma_n \\ P_\alpha^r(t) &= \sum_{i=0}^{d} f_{\alpha,i}^r(t) P_{\alpha+e_i}^{r-1}(t), \quad \text{for} \quad r = 1, \ldots, n \text{ and } \alpha \in \Gamma_{n-r} \end{aligned}$$

where P_α are points in $\mathbb{R}^p$ and $t \in \mathbb{R}^d$. This algorithm defines a *segment* of degree n, which can be, depending on the dimension d, a piece of curve, surface or volume.

For a curve ($d = 1$), the structure of the general simplicial algorithm is triangular, for a surface ($d = 2$) it is tetrahedral (see figure 1). In figure 1, for a given r, the points $P_\alpha^r(t)$ correspond to one step of the algorithm. Starting with the points $P_\alpha^0(t)$ the algorithm produces $P_{000}^2(t)$ which is the point of parameter t on the surface. On the schematic representation of a general simplicial algorithm there are $d + 1$ line

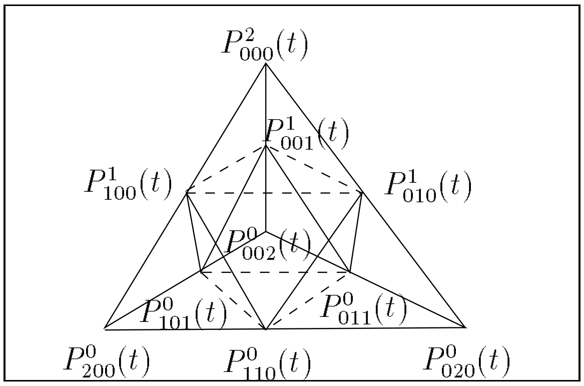

Figure 1: General simplicial algorithm for a surface ($d = 2$) of degre 2 ($n = 2$).

segments under each point $P^r_\alpha(t)$ with $r \geq 1$, defining $d+1$ directions. These line segments symbolize the multiplications by $f^r_{\alpha,i}(t)$.

In the case of the Urn Models Theory, the $f^r_{\alpha,i}(t)$ functions can be written as:

$$f^r_{\alpha,i}(t) = \frac{\tau_i(t) + \mu_{\alpha,i}}{1 + \sum_{j=0}^{d} \mu_{\alpha,j}}$$

where the $\tau_i(t)$ are the barycentric coordinates of t with respect to a set of $d+1$ affinely independent points in $\mathbb{R}^d$, and the $\mu_{\alpha,i}$ are real numbers called *forms parameters*.

In the case of the Simplicial Algorithm Theory, the $f^r_{\alpha,i}(t)$ functions are the barycentric coordinates with respect to a set $X^r_\alpha = \{\mu^r_{\alpha,i}, i = 0, \ldots, d\}$ of affinely independent points. Thus, we have:

$$f^r_{\alpha,i}(t) = d_i(X^r_\alpha|t)/d(X^r_\alpha)$$

Note that numerous distinct $f^r_{\alpha,i}(t)$ functions can fit as long as they are part of a theoretical frame defining the segments accurately.

3 Basic Specifications

The need of general tools leading to sound computer implementations has induced authors to suggest an approach of curves and surfaces based on hierarchies of classes in object oriented languages [10, 14]. Our point of view is in the same spirit but more formal in the first step. The basic idea of algebraic specifications [15] is to describe data structures just by giving the names of the different sets of data, the names of the basic functions and their characteristic properties in an axiomatic equational theory. Data are then described without taking into account the technical details of the implementation.

3.1 Preliminaries

As the field structure of $\mathbb{R}$ only takes place into a general simplicial algorithm, our description is made with respect to a field $\mathcal{K}$ where addition, subtraction, multiplication, and division are defined. In this way, instead of the floating-point arithmetic we can use other arithmetic such as interval arithmetic and so on. Further, we can use complex numbers, as suggested in [1], to define curves and surfaces over the complex field.

To define the $f^r_{\alpha,i}(t)$ functions at each stage r of a general simplicial algorithm, we need $d+1$ elements μ for an index α. We also need the *control points* P_α which are points in $\mathcal{K}^p$. Eventually, *parameters* t are points in $\mathcal{K}^d$.

From a more practical point of view, we want to build segments in an incremental process, adding step by step segment elements.

3.2 The "kernel"

To write down our specifications, we use an easy to read equational specification language created from the propositions in [2, 7, 8]. In our language we suppose that basic specifications modules are predefined. Parameterization of specification modules is also available in this language. This can be achieved by imposing modules to satisfy a specific *Module-Theory*. Theories can themselves be parameterized. Furthermore, one can import modules into other ones. The declaration <**Protecting** *Name*> points out that importation is made without semantics alteration'[8]. In module SEG (table 1), declaration <**Sort** Seg> gives the definition of a *sort*, i.e a data type). Data of sort *Seg* are built by generators *Vseg*, *Ip*, and *Imu*. Generator *Vseg(n, d)* builds an empty segment, i.e. without the definition of μ and without control points, of degree n and dimension d from its input parameters. Generator *Ip(s, p, a)* inserts a control point p of sort *Point* into a segment s, this control point is numbered by a multinomial index a of sort *Index*. Generator *Imu(s, m, a, i)* inserts a μ denoted m and numbered by a multinomial index a according to a direction i, which is an integer of sort *Nat*.

Table 1: Segments generators

Spec SEG(E:MU, P:POINT) **is**	
Protecting INDEX, NAT	
Sort Seg	(segments sort)
Generated by	
Vseg : Nat Nat $\longrightarrow$ Seg	(empty segment)
Ip : Seg Point Index $\longrightarrow$ Seg	(point insertion)
Imu : Seg Mu Index Nat $\longrightarrow$ Seg	(μ insertion)
(Axioms are not written)	
End	

Example: *A segment of degree* 1 *and dimension* 1 *with control points* p1*,* p2*, and* μ *parameters* μ_1*,* μ_2 *is given by formula:*

$$s = Imu(Imu(Ip(Ip(Vseg(1,1), p_1, i_1), p_2, i_2), \mu_1, i_0, 0), \mu_2, i_0, 1)$$

where i_0 *is the index* 00*,* i_1 *index* 10 *and* i_2 *index* 01*. Whatever the manner in which different elements are introduced, the axiomatic of the preceding operations tells us that they define the same segment.*

Module SEG is parameterized by two modules corresponding to formal parameters E and P. Parameter E satisfies the MU theory which introduces the μ of sort *Mu*, and the parameters t of sort *Param* with respect to Urn Models or Simplicial Algorithms. Parameter P satisfies the POINT theory (see table 2). This theory imposes the existence of a sort *Point*, a point addition, and a scalar multiplication operation. Modules MU and POINT are parameterized by the FIELD theory which defines a sort *Number* and the four elementary operations needed to compute using those scalars.

Table 2: Points theory

```
Th POINT(K:FIELD) is
    Sort Point
    Operations
      +p : Point Point ⟶ Point                        ( points addition )
      *p : Number Point ⟶ Point                  ( scalar multiplication )
      ( Axioms are not written )
End
```

Now, let us define the *basic selectors* with their axioms (see table 3). These selectors are needed to be able to get information included in a segment. In module SEGOP declaration, <**Extending** SEG(E, P)> indicates importation, without any particular precautions about the semantics alteration, of module SEG and addition of new operations related with sort *Seg*.

Table 3: Basic selectors for segments

```
Spec SEGOP(E:MU, P:POINT) is
    Extending SEG(E, P)
    Operations
      dimSeg : Seg ⟶ Nat                           ( segment dimension )
      deg : Seg ⟶ Nat                                 ( segment degree )
      ap : Seg Index ⟶ Point                  ( control point selection )
      amu : Seg Index Nat ⟶ Mu                           ( μ selection )
    Axioms
      ap(Vseg(n, d), a) = errPoint
      ap(Ip(s, p, a), b) = if eqi(a, b) then p else ap(s, b)
      ap(Imu(s, e, a, i), b) = ap(s, b)
      ( Other axioms left out )
End
```

Selector *dimSeg(s)* returns the dimension of a segment s. Selector *deg(s)* returns the degree of a segment s. Selector *ap(s, a)* selects the control point numbered by a of the segment s and *amu(s, a, i)* selects the μ numbered by a in direction i of the segment s. The first axiom for selector *ap* indicates the impossibility to get a control point from an empty segment and returns an error value. Erroneous values are caught by an ordered sorts mechanism [8] which is not discussed here.

The second axiom specifies *ap* behavior with respect to generator *Ip*. Control point selection is thus allowed. When generator *Imu* is encountered, the third axiom indicates that it is sufficient to examine the remainder segment. Selector *ap* sends back the first encountered control point of index a.

To simplify the segment representation, it is necessary to ensure that there are not two control points sharing the same index. This requires us to create a new sort, the segment sort of uniquely numbered control points and μ.

3.3 A sorts hierarchy

To build this new sort, let us limit the sort *Seg* by an invariance property. Table 4 shows this new sort specification, named *Segs*.

In module SEGS, declaration <**Sort** Segs $\prec$ Seg [issegs]> means that sort *Segs* is a *Seg* sub-sort whose data verify *issegs* invariant which has the true value if and only if a segment has all its control points and μ uniquely numbered. The invariant *issegs* is built with the two predicates *pok* and *muok* which ensure respectively that all control points are uniquely numbered and that all μ in a direction i are uniquely numbered. The keyword **with** in the definition of *issegs* allows us to define local operations which simplify the expression of the main operations. Here the auxiliary operation*forall* applies *muok* in all directions.

Table 4: Sort segs definition

Spec SEGS(E:MU, P:POINT) **is**

  Extending SEGEX(E, P)

  Sort Segs $\prec$ Seg [issegs]

  Predicates
   pok : Seg (all control points uniquely numbered)
   muok : Seg (all μ uniquely numbered)
   isSegs : Seg

  Axioms
   (pok and muok left out)

   issegs(s) = pok(s) $\wedge$ forall(dimSeg(s))
   with forall(0) = muok(s, 0)
     forall(i) = (muok(s, i) $\wedge$ forall(i-1)
   (Used variables declaration left out)

End

Module SEGS extends module SEGEX which itself extends module SEGOP. The SEGEX module defines predicates *isvide*, *exp* and *exmu* which enable us to know if a segment is empty, if it has a control point of index a and if it has a μ of index a in direction i. For given segment degree and segment dimension, nothing enables us to verify if all necessary control points have been inserted. Sub-sort *Segc1* invariant, defined in SEGC1 module, does this verification. All μ must be inserted too, and so SEGC module defines the *Segc* sub-sort of *complete segments*. Due to the lack of space, SEGEX, SEGC and SEGC1 are not presented. The interested reader can refer to [6]. See the figure 2 for a graphical representation of these modules.

3.4 Evaluation

Operation *eval* in SEGEVAL module (see table 5) translates the general simplicial algorithm evaluation mechanism. Operation *fct_eval* represents the computing function $f^r_{\alpha,i}(t)$. Its axioms, which are theory-dependent, are given later in SEGURNE module (for Urn Models) and SEGSIMPL module (for Simplicial Algorithms) which extend and specialize SEGEVAL module. In an object oriented approach, *fct_eval* can be seen as a *virtual functions*. Note that at this stage, one can introduce any new theory which can be expressed using a general simplicial algorithm and a computing function. By giving modules SEGURNE and SEGSIMPL's parameters values, it is now possible to build up 2D segments, "float" 2D segments, and so on.

4 Specializations

At this point we have discussed general segments. However, numerous curve and surface models exist, each one having particular μ's. Even if it is interesting to individually modify the μ's, it is essential to be able to fix them in a single operation. This is the reason why we extend the *Segc1* sort, as depicted in table 6.

In SEGXT module, the new *xt(s, f, i)* generator inserts a *specialization function* (f:Index $\longrightarrow$ Number) for one given direction i. The *Segc1* sort extension implies

Table 5: Complete segments evaluation

```
Spec SEGEVAL(E:MU, P:POINT) is
    Extending SEGC(E, P)
    Operations
      fct_eval : Segc Param Index Nat ⟶ Number
      eval : Segc Param ⟶ Point
    Axioms
      eval(s, t) = evall(s, t, izero(dimSeg(s)),deg(s))
      with evall(s,t, a, 0) = ap(s, a)
           evall(s, t, a, r) = forall(dimSeg(s))
           with forall(0) = fct_eval(s, t, a, 0) *p evall(s, t, iad(a, ei(dimSeg(s), 0)), r-1)
                forall(i) = fct_eval(s, t, a, i) *p evall(s, t, iad(a, ei(dimSeg(s), i)), r-1) +p forall(i-1)
End
```

Table 6: Segc1 sort extension.

```
Spec SEGXT(E:MU, P:POINT) is
    Extending SEGC1(E, P)
    Generated by
      xt : Segc1 (Index ⟶ Number) Nat ⟶ Segc1
    Axioms
      dimSeg(xt(s, f, i)) = dimSeg(s)
      deg(xt(s, f, i)) = deg(s)
      ap(xt(s, f, i), a) = ap(s, a)
      amu(xt(s, f, i), a, j) = if i=j then f(a) else amu(s, a, j)
      ( Axioms of xt specific selectors left out )
End
```

new axioms to be added to the basic selectors in order to describe the behaviour of basic selectors with respect to the generator *xt*. In fact only the μ selection is concerned with this new generator. This selection is made without having to know if it is needed to compute the μ to be selected. So, there is no need to change the previously defined operations, in particular for evaluation.

We are now able to describe specialized segments. Module BEZIERURNE given in table 7 defines a specialization function for the Bézier model in Urn Models theory. This specialization function always returns 0 (the zero of field K). Hence to define Bézier model for any dimension, we have to insert the *fct_bezier* function for all μ directions. This is done by the *gen_bezier* operation.

Note that Bézier model is the simplest case that occurs. For example, specialization for spline model is not as simple as *fct_bezier* is, indeed we have to consider the knot values. This can be done by partial function applications and is not discussed here. Interested reader may refer to [6].

In order to sum up the discussion, figure 2 shows the hierarchy of specification modules. From the top module SEG to the module SEGC, we restrict the classes of represented objects. After module SEGEVAL, we see that a module, e.g. SEGNEW, corresponding to a new theory which can be added. To do such addition, we have to define new axioms for the computing function *fct_eval* in SEGNEW and if necessary new μ's and t in a parameter module. Going down into the module hierarchy we encounter specialized modules, e.g. BEZIERURNE, SPLINESIMPL and so on.

As we have remained so far on a theoretical level, let us explain in practice how we actually use the previously defined concepts in our experimental computer tool.

Table 7: Bézier segments specification.

Spec BEZIERURNE(K:FIELD, P:POINT) **is**

Extending SEGURNE(K, P)

Sort Bezier $\prec$ Segc

Generated by
gen_bezier : Segc1 $\longrightarrow$ Bezier

Operations
fct_bezier : Index $\longrightarrow$ Number

Axioms
fct_bezier(a) = 0

gen_bezier(s) = forall(dimSeg(s))
with forall(0) = xt(s, fct_bezier, 0)
forall(i) = xt(forall(i-1), fct_bezier, i)

End

5 Tools and Application

Several algebraic specification systems offer rigorous frames to develop specifications [15]. However, we want to build a prototype with a high degree of interaction and with a graphical user interface. Our prototype can be seen as a segment editor or modeler with a lot of curve, surface and volume models.

We have chosen to stress the interaction on the segment model parameters, that is to say the μ's, in order to be able to explore new segment models potentially included in theories (such as Urn models or Simplicial Algorithms).

To do that, our implementation is based on a specification translation into Caml-Light [11], an efficient functional language of the ML family. Moreover, this language can be interfaced with the C language and so we can use a graphical library. This typed functional language allows us to develop a prototype under good conditions.

The specifications of segments are written in a functional programming style. Thus writing Caml-Light functions corresponding to them remains close to the chosen formalism. Furthermore, the formal level (the specifications) and prototype (in Caml) evolve accordingly.

Let us now describe some functionalities of the prototype. Given 3D points, we can interactively build segments of dimension one, two and three, as shown on figure 4. We insist on the fact that there is no difference to work with curves, surfaces or volumes.

In figure 3 we illustrate the edition of a curve segment. Here we are in the case of a curve in the Urn Models theory, the μ's are real numbers and, on the left side of figure 3, they are edited on the keyboard. The entries positions refer to the schematic representation of the corresponding general simplicial algorithm.

The possibility to edit the μ's with a button box is shown on the right side of figure 3. In a first step the user chooses which μ are associated to which potentiometer. On the left down corner of the interaction window potentiometers are symbolized by radio button, the user select a potentiometer by clicking on it. Note that, after he associates the μ with the selected potentiometer also by clicking, he can associate more than one μ to one potentiometer. In a second step, the user turns the potentiometers and the corresponding μ are changed. Effects of these modifications are observable in real time. In the case of Urn Models, following Barry & Goldman indications [1] we can interactively add tension and bias to segments.

Finally to illustrate the possibilities of having several segment models, figure 4 displays some classical ones. These specializations are also available in dimension two, to create surfaces and three for volumes.

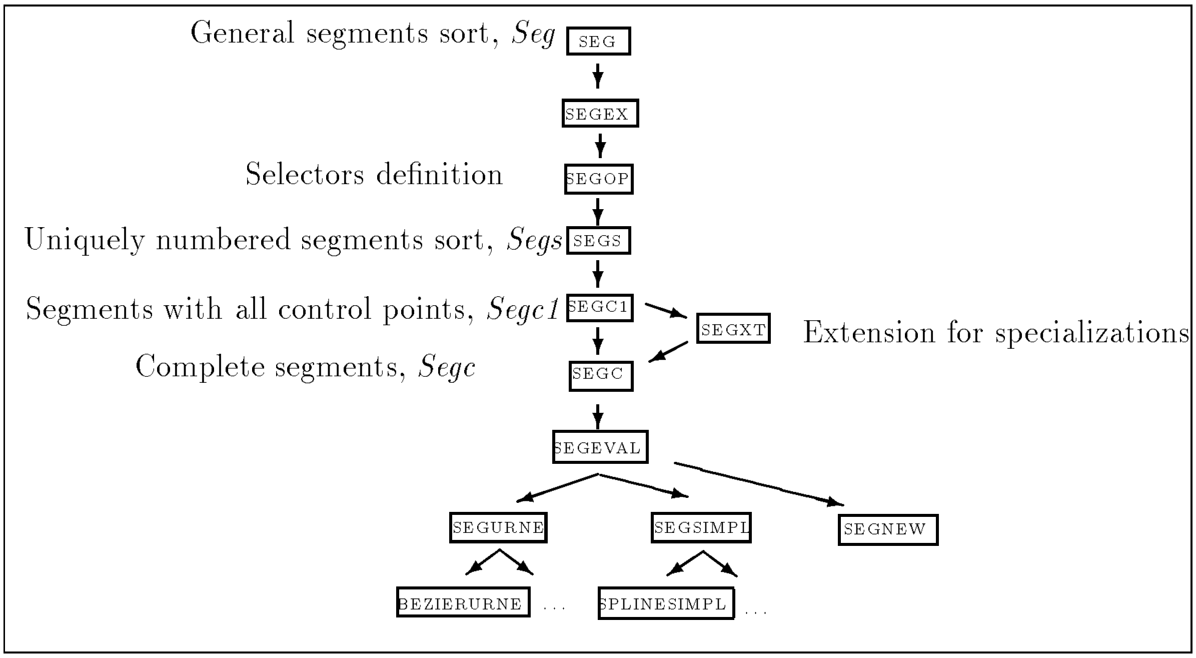

Figure 2: A main part of the hierarchy of specification modules.

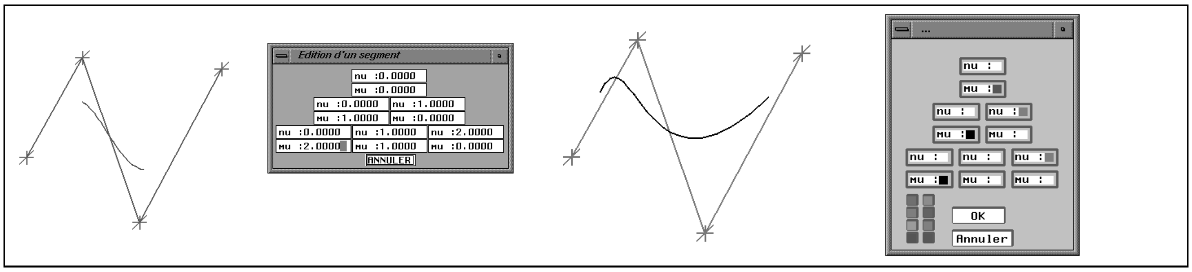

Figure 3: Segment edition on keyboard (left), with a button box (right).

6 Conclusion

Urn Models and Simplicial Algorithms explain known phenomena appearing in tailor-made curve and surface models used in modeling and image synthesis. But facts related with building and computer representation of curves and surfaces are often overshadowed. Our work shows that formal specification methods are able to consider those aspects in a manner which is enough abstract to catch several curves and surfaces theories and their implementations.

This high level of abstraction allow us to use, in a second step, various programming paradigms and languages. We use a functional language to validate the operational meaning of our design methodology. With this methodology and from our specifications we can also develop an object-oriented program joining the proposition of [10, 14].

Our prototype is actually usable to experimentation. This one is working undifferently with curves, surfaces and volumes. Editing segments allows us to explore models inside several theories. This part of experimentation seems to be very fruitful and we hope to discover interesting particularities which lead to segment operators usable by final users.

Further works are needed to extend our specifications and therefore our prototype, some of them being under current development. Compound objects made by several general simplicial algorithms will be considered with a topological structure. A tensor-

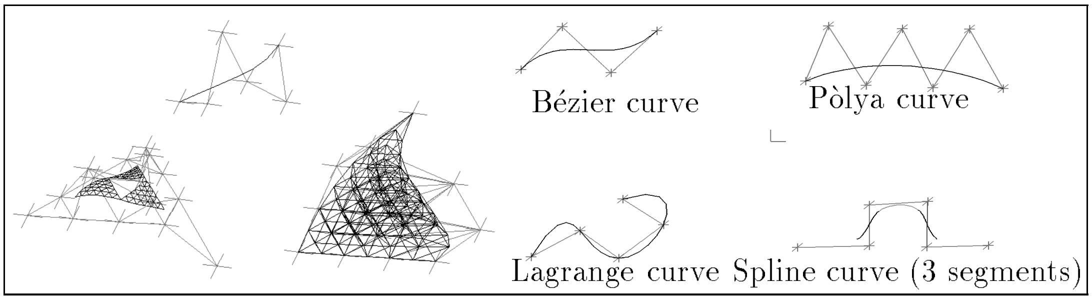

Figure 4: Left, our prototype works equally with curves, surfaces and volumes. Right, some specializations.

product approach of varieties, currently under investigation, proposes a general frame with a good topological underground. We are also studying "High level" operations, such as derivation, interpolation, or putting bias and tension.

Finally, from our prototype future works would lead to create new computer tools to easily create and manipulate curves, surfaces and volumes in CAD and image synthesis.

References

[1] P. J. Barry and R. N. Goldman. Recursive polynomial curve schemes and computer-aided geometric design. *Constructive Approximation*, 6:65–96, 1990.

[2] Y. Bertrand. *Spécification algébrique et réalisation d'un modeleur interactif d'objets géométriques volumiques.* PhD thesis, Université Louis Pasteur, Strasbourg, France, October 1992.

[3] Y. Bertrand and J-F. Dufourd. Algebraic specification of a 3D-Modeler based on hypermaps. *CVGIP: Graphical Models and Image Processing*, 56(1):29–60, January 1994.

[4] C. Brezinski and G. Walz. Sequences of transformations and triangular recursion schemes, with applications in numerical analysis. *Journal of Computational and Applied Mathematics*, 34:361–383, 1991.

[5] W. Dahmen, C. A. Micchelli, and H-P. Seidel. Blossoming begets B-spline bases built better by B-patches. *Mathematics of computation*, 59(199):97–115, July 1992.

[6] L. Fuchs, D. Bechmann, Y. Bertrand, and J-F. Dufourd. Spécification algébrique de modèles de courbes. Technical Report 95-15, LSIIT, Université Louis Pasteur, 1995.

[7] M.-C. Gaudel. Structuring and modularizing algebraic specifications : the PLUSS specification language, evolutions and perspectives. In *Proc. of the 9th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 3–23. Springer-Verlag L.N.C.S. 577, 1992.

[8] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J-P. Jouannaud. Introducing OBJ. In *Applications of Algebraic Specifications using OBJ.* Cambridge, 1992.

[9] N. Goldman, R. and J. Barry, P. Wonderful triangle: A simple, unified, algorithmic approach to change of basis procedure in computer aided geometric design. In T. Lyche and L. Schumaker, L., editors, *Mathematical Methods in Computer Aided Geometric Design II*, pages 297–320. Academic Press, 1992.

[10] G. Greiner, A. Kolb, R. Pfeifle, H-P. Seidel, P. Slusallek, M. Encarnação, and R. Klein. A platform for visualizing curves and surfaces. *Computer-Aided Design*, 27(7):559–566, 1995.

[11] X. Leroy and P. Weis. *Manuel de référence du langage Caml.* InterEditions, Paris, 1993.

[12] W. R. Mallgren. Formal specification of interactive graphic programming languages. ACM dist. dissertation, MIT Press, USA, 1982.

[13] Ph. N. Nehlig and D. A. Duce. GKS–9x: The Design Output Primitive, an Approach to a Specification. In M. Daehlen and L. Kjelldahl, editors, *Proceedings of Eurographics '94.* NCC/Blackwell, 1994.

[14] A. H. Vermeulen and R. H. Bartels. C++ splines classes for prototyping. In *SPIE Vol. 1610: Curves and Surfaces in Computer Vision and Graphics II*, pages 121–131, 1991.

[15] M. Wirsing. Algebraic Specification. In J. van Leeuwen, editor, *Handbook of theoretical computer science*, pages 675–788. Elsevier Science, 1990.