Computer-Aided Design and Applications

http://www.cadanda.com

A Generic Rule-Based Formulation of Knot-Insertion Across CAD Applications

Abdulwahed M. Abbas

The University of Balamand, abbas@balamand.edu.lb

ABSTRACT

In software design, a rule-based architecture is usually attempted as an alternative option when no sufficient information is available to devise a constructive algorithm. In this context, this paper starts by providing an account of rule-based systems and proceeds to provide a formulation of the knot insertion algorithm for B-spline curves in these terms. The benefits of that are clarity, flexibility and extendibility. In fact, the formulation not only admits knot insertion for B-spline and NURB surfaces as simple generalizations, but also extends to cover T-splines local refinement as well as similar notions in higher-dimensional structures.

Keywords: algorithm, rule-based systems, knot insertion, NURBS, T-Splines.
DOI: 10.3722/cadaps.2008.831-840

1. INTRODUCTION

In software applications, such as computer-aided design (CAD), a mathematical relation or a formula is acknowledged to be the most direct method of obtaining a result. A formula is a shortcut to the solution of a problem, since it relies on a limited number of operations – representing the essence of available knowledge on the subject.

However, when domain knowledge is not as mature as to possess a formula, the immediate alternative would be to seek an algorithm that can achieve the task. The term *algorithm* is used here in its fundamental theoretical sense; i.e. as a *finite* sequence of *conscious* instructions whose execution *directly* leads to the solution of the problem; i.e. a *constructive* algorithm.

When pursuing this alternative, the outcome is largely dependent on the number of steps needed for the algorithm to complete the task, as well as on the number of basic mathematical operations that are employed at each of those steps.

Unfortunately, available knowledge might again not be so sufficient as to even devise an algorithm. Artificial Intelligence (AI), for instance, along its history, have constantly been faced with such situations, where a solution of a problem needs to be reached at the light of scarce (partial at best) knowledge.

This may be why AI (perhaps more than any other field of research) is pervaded with notions such as: search strategies [15], heuristic methods [13], fuzzy algorithms [4], blackboard architecture [18] and rule-based (or production) systems [6]. Fields such as expert systems [9] and neural networks [8] may also be considered within this category. As the dates of these references suggest, these topics used to be prime targets for research at the time precisely to compensate for the absence of a formula and/or of a complete algorithm.

As in any other field of open research, one does not need to expend too much effort in order to find problems in CAD that fall under this *grey* category. Here, proponents of rule-based reasoning argue that a solution obtained this way is better than no solution at all. The argument goes further to bring out benefits, such as clarity and flexibility, to counter shortcomings such as inefficiency or partiality of the solutions thus obtained.

Given its long history, the introduction of rule-based reasoning to the CAD domain is perhaps not entirely novel. However, this paper advances a perspective especially highlighting the particular category of problems for which rule-based reasoning is most suitable.

In this setting, this paper draws a general framework for performing knot insertion in terms of rule-based systems. This framework admits all known variations of this process as simple instances: curves, B-spline and NURB surfaces and also T-splines. It further opens the way for other potential applications and generalizations.
This paper is structured as follows: section 2 introduces rule-based systems. Section 3 demonstrates, as an illustration, that any subdivision scheme can very well fall under this formulation. Section 4 puts forward an alternative formulation of the knot insertion algorithm as a rule-based system. Section 5 presents knot insertion for B-spline and NURB surfaces as a trivial generalization of this formulation, while section 6 presents local refinement for T-splines as another, though less trivial, generalization. The paper concludes with suggestions for further work.

2. RULE-BASED SYSTEMS: A DEFINITION

A rule-based system generally consists of the following major components (see Fig. 1):

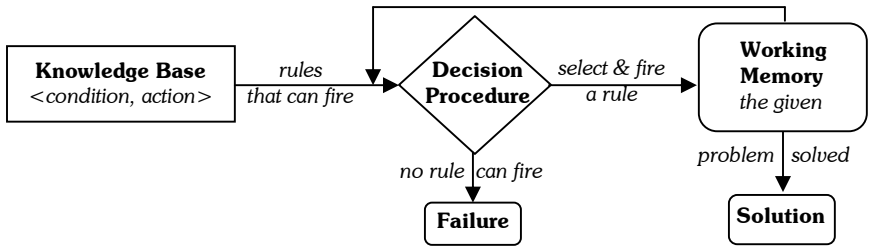


Fig. 1: A Rule-Based System Architecture.

- Working Memory: initially containing the *given* of the problem.
- Knowledge Base: embodying among other things a list of *<condition, action>* rules. The *condition* part of a rule has to be satisfied, based on the information currently available in the working memory, before the rule can fire. The *action* part of the rule is fired for the side-effects it will have on the working memory.
- Decision procedure: governing the *selection* of which rule is to fire next, until the solution of the problem is reached or until no more rules are there to fire. An argument must be provided as a justification of why this procedure will end up reaching the solution on the basis of the given knowledge base. However, such an argument is usually kept implicit.

In summary, the main control routine of a rule-based system can be cast in the following simple form:

```
WM = the given of the problem;
While ((problem is not solved) and (there are applicable rules)) {
  select an applicable rule and fire it;
  update WM accordingly; }}
If (problem is solved) then declare solution; else declare failure;
```

Several remarks should be made about the above pseudo code. Firstly, the terminology used in the definition above is a slight adaptation of the standard terms that are generally found in the literature. Secondly, although such paradigm is more likely to be encountered in symbol (rather than number) based applications; this paper proposes that some of the benefits of that can be of use to computer-aided geometric design.
Thirdly, the underlying process is *data-driven* or *knowledge-driven*. The knowledge is partly in the associated set of rules (or productions). Moreover, the core control in the five lines of code above resides in the *select* procedure in line 3. The knowledge that one can implant into this procedure could greatly impact its efficiency (and therefore its usability). Finally, since the decision procedure is determined once and for all, the skills that are required in this domain are more of representational than algorithmic nature. That is, when attempting to solve any particular problem, the attention that needs to be paid will mostly be to specifying the contents of the knowledge base.
The following sections are intended to show that well-established algorithms may straightforwardly be implemented as rule-based systems. The modularity of the design brought in by such an exercise can make explicit useful features of the algorithm that may otherwise remain hidden.

3. SUBDIVISION SCHEMES ARE RULE-BASED SYSTEMS

Recursive subdivision [5] is a process through which a smooth surface is obtained out of a rough control polyhedron. That is, a subdivision scheme starts from an initial *control* polyhedron P_0, all components of which are subjected to a

set R of subdivision rules. That is, P_0 starts a sequence of more and more finely subdivided polyhedrons P_1, P_2 ... etc, where each P_i is the result of subdividing the one before it. The limit of this sequence is a smooth subdivision surface. Given what has been said in the previous section, the basic notions of a subdivision scheme should be easy to identify with those of a rule-based system:

- the initial polyhedron of the scheme is the initial working memory of the system.
- the subdivision rules of the scheme are the *<condition, action>* rules of the system. The *condition* part of the rule specifies when and where a subdivision rule is applicable while the *action* part specifies the outcome of its application.
- the subdivision procedure of the scheme is the decision procedure of the system.
- finally, the *raison d'être* of the scheme is a proof (or an argument) that the scheme yields the limit subdivision surface. This can manifest itself in a data structure holding a control polyhedron together with an argument showing how the next polyhedron in the sequence is assembled out of that [1].

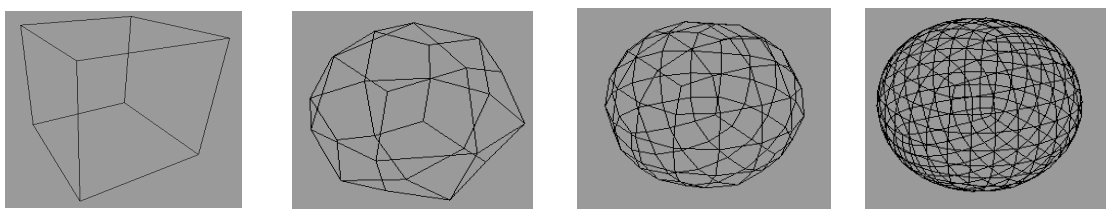
The next section contains a reformulation of the Catmull-Clark subdivision scheme as a rule-based system. It goes without saying that any other scheme can as easily be reformulated.

3.1 The Catmull-Clark Subdivision Scheme

In a single Catmull-Clark (CC) subdivision step [5], a control polyhedron P is subdivided into the next polyhedron P', as follows (see fig. 2): each face F of P gives rise to an F-vertex of P' that is the average of the vertices of the face F. Each inner edge E of P gives rise to an E-vertex of P' that is the average of the vertices of the edge together with the F-vertices of the adjacent faces of E in P. Each inner vertex V of P gives rise to a V-vertex of P' defined by the expression in (1), where:

$$\frac{(n-2)\times V + \frac{(R+S)}{n}}{n} \tag{1}$$

- n is the number of faces adjacent to V.
- $R = \sum_{i=1}^{n} V_i$ and $S = \sum_{i=1}^{n} v_{fi}$, where vv_i is an edge of P and v_{fi} is an F-vertex of a face f_i embodying V.



$Level_0$ $Level_1$ $Level_2$ $Level_3$

Fig. 2: Catmull-Clark Subdivision.

At the end of this process, each F-vertex is connected to the adjacent E-vertices and each E-vertex is connected to the adjacent V-vertices, thus forming the next subdivided polyhedron P'.

3.2 Catmull-Clark Subdivision as a Rule-Based System

Fig. 3 sketches the CC subdivision scheme as a rule-based system. The sketch assumes that the initial control polyhedron is assembled out of a set of related parts: vertices, edges and faces. Each of these parts will have to be appropriately flagged to insure that it is subject to a single application of the appropriate rule.

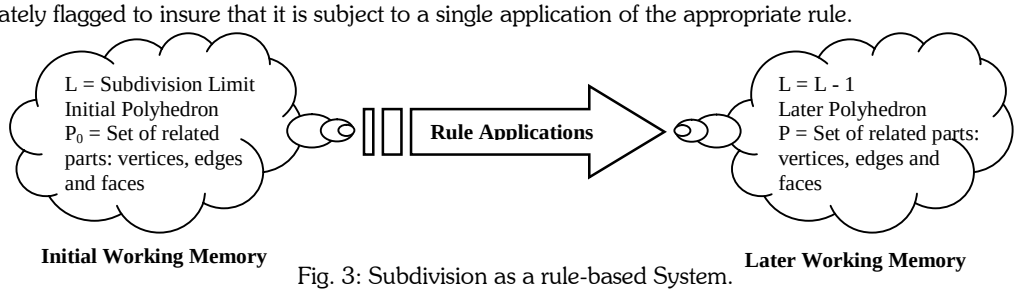


Fig. 3: Subdivision as a rule-based System.

Furthermore, the subdivision loop terminates as soon as the subdivision limit reaches zero. Reference [1] sheds some light on how the newly subdivided parts of the current polyhedron P are gradually assembled to form the next subdivided polyhedron P'. The following list represents the rules that are necessary for the system to achieve the subdivision task:

- if (F is a face (in the current polyhedron P) having n vertices $v_1 \ldots v_n$)
  then {v = F-Vertex(v_1,…, v_n); add v to the next polyhedron P' in WM;}
- if (E is an edge (in the current polyhedron P) common to vertices v_1 and v_2
  and to faces f_1 and f_2, where f_1 and f_2 have the F-vertices w_1 and w_2 respectively)
  then {v = E-Vertex(v_1, v_2, w_1, w_2); add v to the next polyhedron P' in WM;}
- if (V is a vertex (in the current polyhedron P) common to n faces $f_1 \ldots f_n$
  (having F-Vertices $v_1 \ldots v_n$) and to n edges $e_1 \ldots e_n$ (having E-Vertices $w_1 \ldots w_n$))
  then {v = V-Vertex(V, $v_1 \ldots v_n$, $w_1 \ldots w_n$); add v to the next polyhedron P' in WM;}
- if (all components of the current polyhedron P are now subdivided)
  then {Replace P by P' in WM; decrement L by 1;}

The following features in the formulation of the above rules are worth pointing out:

- the type and interrelationships of information allowed in the working memory
- the varieties of control modes allowed in the decision procedure
- the complexity of the conditions and the amount of procedural information that may be embodied in the action part of a rule
- the amount of modularity introduced by the design implies a comparable amount of simplicity to a rather complex control structure

The formulation of the rules implies some sequencing on their applications. However, this implicit sequencing is still less than the level of sequencing required by an actual algorithm, which clearly influences the resulting flexibility of the system design. Furthermore, on the application side, the clean formulation of the subdivision rules lays the grounds for experimenting with concepts such as:

- Parallel subdivision: applying instances of the rules concurrently to isolable components of the polyhedron.
- Hierarchical subdivision: stopping certain rules from being applied while applying others more than once during the current subdivision iteration.

4. THE KNOT INSERTION ALGORITHM AS A RULE-BASED SYSTEM

Knot insertion is a fundamental algorithm in curve and surface design. Its goal is to add one or more knots to the associated knot vector without changing the shape of the resulting curve. This section starts with some background information of knot insertion intended to make the material that follows more self-contained (see [7], for further elaboration on that).

4.1 B-spline Basis Functions

Given a positive integer k and a sequence (τ) of knots t_0, t_1, …, t_{n+k}, such $t_i \le t_{i+1}$, for all i such that $0 \le i < n+k$, a B-spline basis function $N_{i,k}$ is recursively defined as follows:

- $N_{i,1}(t) = 1$ if $t_i \le t < t_{i+1}$ and 0 otherwise.
- $N_{i,k}(t)$ is defined by the following recursive expression:
-

$$\frac{t-t_i}{t_{i+k-1}-t_i} N_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} N_{i+1,k-1}(t) \tag{2}$$

B-spline basis functions are nonnegative: $N_{i,k} \ge 0$; we in fact have $0 \le N_{i,k} \le 1$. Moreover, they have *partition of unity* property:

$$\sum_{i=0}^{n} N_{i,k}(t) = 1 \tag{3}$$

4.2 B-spline Curves and Surfaces

Given a *knot vector* (τ) and a sequence (π) of control points P_0, P_1 … P_n, a B-spline curve of order k (degree k-1) is defined by the following equation:

$$P(t)=\sum_{i=0}^{n} N_{i,k}(t)P_i, t_{\min} \le t < t_{\max} \tag{4}$$

Since $N_{i,k} = 0$ only when $t < t_i$ or $t \ge t_{i+k}$, a control point P_i influences the curve only for $t_i \le t < t_{i+k}$. This means that at most k consecutive control points influence the curve in any knot span. Equation (4) directly generalizes to B-spline surfaces by taking a tensor product:

$$P(s,t) = \sum_{i=0}^{p}\sum_{j=0}^{q} N_{i,k}(s)N_{j,l}(t)P_{ij}, s_{\min} \le s < s_{\max}, t_{\min} \le t < t_{\max} \tag{5}$$

Similarly, NURBS surfaces are generalizations of B-spline surfaces; the primary difference being the weighting of the control points which makes NURBS curves *rational* [12, 14].

4.3 The Knot Insertion Routine

The *knot insertion* routine consists of adding a new knot to an existing knot vector *without causing any change to the shape* of the curve. The number of knots, the number of control points, and the order of the B-spline curve (respectively m+1, n+1, and k) are bound by the *fundamental identity*: m = n + k.

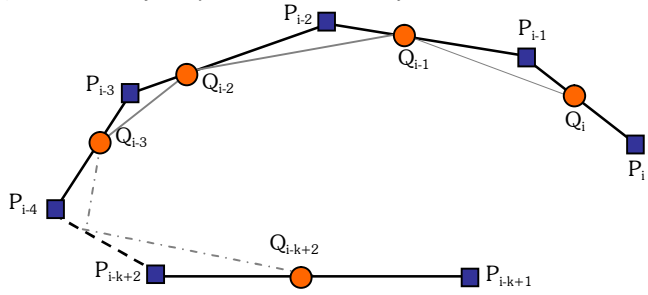


Fig. 4: B-spline Curve Knot Insertion.

Adding a new knot increases the value of m is by 1, which causes violation of the above identity. Restoring this identity is achieved through an identical increase of the number of control points. In general, calculating the effects of knot insertion involves only k consecutive control points (see fig. 4). During the process, some existing control points are replaced by new ones through *corner cutting*.

For example, suppose that the new knot t lies in knot span $[t_i, t_{i+1}[$ (see fig. 4). Inserting t is accomplished by finding k-1 new control points: Q_i on edge $P_{i-1}P_i$, Q_{i-1} on edge $P_{i-2}P_{i-1}$, ..., and Q_{i-k+2} on edge $P_{i-k+1}P_{i-k+2}$ such that the old polygon between P_{i-k+1} and P_i is replaced by $P_{i-k+1}Q_{i-k+2}...Q_iP_i$ by cutting the corners at P_{i-k+2}, ..., P_{i-1}. All other control points remain intact, which mean that k-2 control points of the original control polygon are replaced by k-1 new points. Each new control point Q_j, on edge $P_{j-1}P_j$, is determined by: $Q_j = (1 - \alpha_j)P_{j-1} + \alpha_jP_j$, where:

$$\alpha_j = \frac{t - t_j}{t_{j+k-1} - t_j}, \; i - k + 2 \le j < i \tag{6}$$

4.4 An Alternative Formulation of the Knot Insertion Routine

Without any loss of generality, the discussion below is conducted for cubic B-spline curves (i.e. degree 3 or k = 4).

In view of what has been said above, for any given index i, the basis function $N_{i,4}$ depends on 5 consecutive elements of the knot vector (τ) forming a subsequence starting at the i^{th} element. Accordingly, the B-spline curve expression (4) can be rewritten as:

$$P(t) = \sum_{i=0}^{n} N_{i_0,4}(t)P_i, \; t_{\min} \le t < t_{\max} \tag{7}$$

Where $N_{i0,k}$ is evaluated with respect to a subsequence consisting of those 5 consecutive knots, starting at the i^{th} element of the knot vector (τ). Here, i_0 refers to the first element of this subsequence.

Thus, during processing, any representation of the curve needs only to hold the control points of the curve together with the corresponding knot subsequences, since the individual points of the curve can always be calculated from such information, through equation (7).
We now describe a process that is able to reproduce knot insertion by manipulating the summation in equation (7) directly. This is done simply by substituting equals for equals. We start by providing a data structure (see fig. 5) that can hold the information required by the process, on the assumption that it will eventually be embodied in the knowledge base of the corresponding rule-based system.

t_0	t_1	t_2	t_3	t_4	…	t_{n+2}	t_{n+3}	t_{n+4}
		P_0 $[t_0 \dots t_4]$	P_1 $[t_1 \dots t_5]$	P_2 $[t_2 \dots t_6]$	… …	P_n $[t_n \dots t_{n+4}]$		

Fig. 5: Data Structure for Alternative Formulation of Knot Insertion.

Given the initial sequence of control points (π) and the initial knot vector (τ), we construct an array A[0 .. n+4], where each element A[i] holds:

- the corresponding knot t_i and
- a set of pairs <P, T>, where P is a control point and T is the 5 contiguous knot subsequence *obtained* from the knot vector (τ) and associated with P; i.e. the middle element of T should precisely be t_i. At initialization, the set residing at A[i] contains *at most* a single pair <P, T>.

4.5 The Metaphor

At this stage, it is important to motivate the intuition that can lead someone to believe that a fundamental algorithm such as knot insertion is implementable as a rule-based system. We do that through outlining a suitable metaphor. This same metaphor already played a basic role in the development of fundamental AI concepts such as *belief revision* [19] and *truth-maintenance* systems [10].
Very briefly, in *belief revision*, the working memory is composed of a set of *consistent* beliefs of the domain of application. Such beliefs are interrelated in their consequences in the sense that they embody varying relationships tying the same entities. This means that, in the face of a new incoming belief, the system will have to undergo a revision of its whole belief set, in order to ensure its *consistency*. The nature of revision and the depth of possible modifications depends on the relationships between existing beliefs (and their consequences) and the new incoming belief.
Similarly, *truth maintenance* refers, among other things, to the process the working memory has to undergo to maintain *consistency* when one (ore more) of its belief elements is no longer true.
It is maybe suggestive to point out here that this same vocabulary and the same metaphor are used in the description of the T-spline local refinement [16].

4.6 Knot Insertion as a Rule Based-System

Following the same suggestive metaphor, the array of fig. 5 will be in a *consistent* state at initialization, in the sense that it has the appropriate information to calculate the summation (7). However, when a new knot t is inserted in the sequence (and therefore in the corresponding array), *inconsistency* is introduced to the summation in the sense that one or more of the subsequences will no longer be *validly* associated with the corresponding point as a result of this insertion.
As a consequence, the array has to undergo a *revision* process, whose purpose is to *restore consistency* to the array, while keeping the above summation intact. Here, inconsistency at a point P is detected when the newly inserted knot t appears somewhere in T thereby distorting its value, while equivalence is taken to mean that the above summation remains intact. Thus, the task is to eliminate from the array all those inconsistent pairs <P, T> and reinserting equivalent but consistent pairs at the *appropriate* slots of the array.
When t is inserted, a new entry for it is made at the appropriate place in the array. However, given the way the array is conceived; the addition of the new entry will not by itself affect the contents of any other entry in the array. But, when a subsequence T, originally equal to $[t_{i-2}, t_{i-1}, t_i, t_{i+1}, t_{i+2}]$, is re-inferred and found to have t appearing among its t_i's, the action taken will depend on the exact position of t there; hence the rules listed below, adapted to our context from [16]. Actually, the source of these rules is explicitly identified in [3].

- if $t_0 < t < t_1$
  then Delete <P,T> and add $<c_0P,[t_0,t,t_1,t_2,t_3]>$ and $<d_0P, [t,t_1,t_2,t_3,t_4]>$ to the appropriate slots in the array, where $c_0 = (t-t_0)/(t_3-t_0)$ and $d_0 = 1$
- if $t_1 < t < t_2$
  then Delete <P,T> and add $<c_1P,[t_0,t_1,t,t_2,t_3]>$ and $<d_1P, [t_1,t,t_2,t_3,t_4]>$ to the appropriate slots in the array, where $c_1 = (t-t_0)/(t_3-t_0)$ and $d_1 = (t_4-t)/(t_4-t_1)$
- if $t_2 < t < t_3$
  then Delete <P,T> and add $<c_2P,[t_0,t_1,t_2,t,t_3]>$ and $<d_2P, [t_1,t_2,t,t_3,t_4]>$ to the appropriate slots in the array, where $c_2 = (t-t_0)/(t_3-t_0)$ and $d_2 = (t_4-t)/(t_4-t_1)$
- if $t_3 < t < t_4$
  then Delete <P,T> and add $<c_3P,[t_0,t_1,t_2,t_3,t]>$ and $<d_3P, [t_1,t_2,t_3,t,t_4]>$ to the appropriate slots in the array, where $c_3 = 1$ and $d_3 = (t_4-t)/(t_4-t_1)$

Rule applications will remove and add points on the basis of the current state of the working memory; that is, on the basis of the effects of earlier rules applied so far.
Given that the decision procedure for this task is still the general loop stated in pseudo code of section 2, we stress that the underlying process yields an identical result to the traditional knot insertion algorithm. The proof of that emanates from the data structure representing the initial polygon and the associated knot information (see fig. 5).
This process terminates successfully when no more inconsistent subsequences occur in the working memory. For this reason, the termination condition should be modified accordingly. Moreover, the proof that the loop achieves what is intended at termination is quite essential here.
This proof is easy to obtain. In fact, since the process does not create any new knot values, the number of iterations of the loop has an upper bound. The bound is the number of subsequences of length 5 of the initial knot vector (τ) that might contain the newly inserted knot in the position where it is inserted.
In the next two sections, we will show that the above formulation of the knot insertion algorithm can make it straightforward to obtain related concepts as simple generalizations in neighboring domains. Such concepts are intuitively easy to relate to, but not as easy to realize using the traditional formulation of the knot insertion algorithm.

5. FIRST GENERALIZATION: KNOT INSERTION FOR B-SPLINE AND NURBS SURFACES

For B-spline and NURB Surfaces [12, 14], the knot-insertion routine now involves two knots s and t. Accordingly, the system described above can be generalized as suggested by equation (4) to deal with knot insertion in the s or in the t direction. In these terms, the associated data structure grows to a 2-dimensional array (see fig. 6), where each entry is a set of triplets <P, S, T>. Here, P is a control point, S is the subsequence of knots in the s direction and T is the subsequence of knots in the t direction.
The associated rules are accordingly generalized. The following couple of rules are a sample of those:

- If $s_0 < s < s_1$
  Then Delete <P,S,T> and add $<c_0P,[s_0,s,s_1,s_2,s_3],T>$ and $<d_0P,[s,s_1,s_2,s_3,s_4],T>$ to the appropriate slots in the matrix, where $c_0 = (s-s_0)/(s_3-s_0)$ and $d_0 = 1$
- If $t_0 < t < t_1$
  Then Delete <P,S,T> and add $<c_0P,S,[t_0,t,t_1,t_2,t_3]>$ and $<d_0P,S,[t,t_1,t_2,t_3,t_4]>$ to the appropriate slots in the matrix, where $c_0 = (t-t_0)/(t_3-t_0)$ and $d_0 = 1$

It is important to note here that inserting both knot values may involve the insertion in the matrix of a whole new empty row or column, as the case might be. However, given the way the matrix is conceived, this addition by itself will not affect the values in other entries of the matrix.

Similarly, firing a rule may involve the insertion in the matrix of a whole new empty row or column, as the case might be. Again, given the way the matrix is conceived, this addition will not by itself affect the values in other entries of the matrix.
The particular choice of the condition part of the rules will mean that knot insertion will propagate to wholly cover the corresponding row and column as it should do.
More interestingly perhaps, generalization here does not necessarily have to stop at two dimensions, but can go on to higher dimensions thus opening the way for more possibilities.

	t_0	t_1	t_2	t_3	…	t_{n+2}	t_{n+3}	t_{n+4}
s_0								
s_1								
s_2			P_{00} $[s_0 \dots s_4]$ $[t_0 \dots t_4]$	P_{01} $[s_0 \dots s_4]$ $[t_1 \dots t_5]$	…	P_{0n} $[s_0 \dots s_4]$ $[t_n \dots t_{n+4}]$		
s_3			…	…	…	…		
…			…	…	…	…		
s_{m+2}			…	…	…	…		
s_{m+3}								
s_{m+4}								

Fig. 6: Data Structure for B-spline Surface Formulation.

6. SECOND GENERALIZATION: T-SPLINE LOCAL REFINEMENT

T-splines [16, 17, 20] generalize both B-spline and NURB surfaces in the sense that T-junctions are allowed in the control mesh. That is, a row (or column) of control points does not have to carry through from one side of the T-mesh to the other, but can in fact be broken anywhere along the way (see fig. 7).

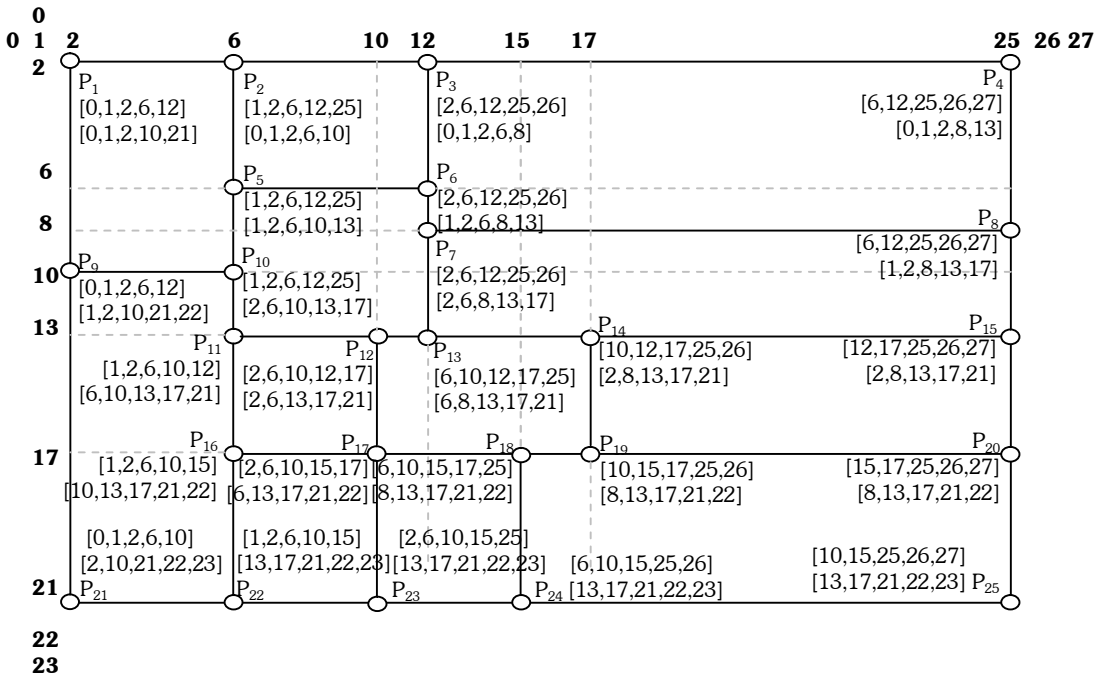


Fig. 7: Knot Information associated with individual control points of a T-mesh.

Accordingly, T-splines come together with a knot inference mechanism that is able to generate for each control point the knot information it requires. They also come together with a local refinement mechanism that is able to consistently utilize the knot inference mechanism from one refinement step to the next.

The T-splines local refinement mechanism, presented in [17], is easily identifiable with the *backward-chaining* mode of a rule-based system, and seems to suffer from the same problems that are generally associated with that mode.

For this reason perhaps, the work described in [16] seems to be partly motivated by the desire to rectify these problems. In fact, the authors mention that the number of new control points that are introduced during local refinement is now considerably smaller. Furthermore, the process now provably terminates, though no explicit mention of a proof is made anywhere in the paper.

In so far as we are concerned, the control flow of the local refinement procedure described in [16] most closely resembles what is depicted in pseudo code of section 2. However, the data structure depicted in fig. 6 is now a sparse matrix, in the sense that the corresponding slots do not have to be all populated at initialization time. Furthermore, knot subsequences are inferred differently.
These differences do not however affect the way the process work as a rule-based system, because the way the system works is abstracted away from both differences. Consequently, though the knot inference mechanism is different, similar rules to those used for B-spline and NURB surfaces can still be applied here! In fact, with reference to the data structure in fig. 6, a couple of these rules are listed below:

- If existing subsequence S of <P, S, T> is different from newly inferred S' Then Revise (P, S, S', T)
- If existing subsequence T of <P, S, T> is different from newly inferred T' Then Revise (P, S, T, T')

In order to make the local refinement process more understandable in the T-spline case, we need only compare it to the simpler knot insertion process in the NURBS case, for example.
In fact, in a NURB surface, the difference between S and S' (respectively T and T') occurs because a new knot s (respectively t) appears in S' but not in S (respectively T' but not in T). In this case, revision will take place according to the rules listed in section 4.6, considered for s or t, as the case might be.
By contrast, in a T-mesh, the knot subsequences associated with every one of its control points are not evenly formed (see fig. 7), by comparison to those of a NURB surface. Furthermore, the firing of a rule here is also tied up to other conditions related to the formation of new edges that the new knot might incur to the T-mesh. Consequently, the firing of those rules in the context of T-splines need not propagate along entire rows and columns in general.
Seeing how abruptly this section is concluded, a tutorial exposition of the design decisions made in this section is to be found in [3]. This exposition is fundamentally based on the theoretical framework drawn in this paper. Another reference concerning T-splines that is worth looking at in the context of interpolation is [11].

7. CONCLUSIONS AND FURTHER RESEARCH

The main contribution of this paper is at the level of conceptual modeling and methodology. The metaphor presented in this paper is being advanced to play a role for knot insertion as fundamental as that of the notion of *ratio* in the design or curves and surfaces from control meshes. The role of rule-based systems here is in providing the required flexibility for design and implementation. It is this flexibility which allowed the many guises of knot insertion across domains to be expressed as simple particular cases of a single unifying theme.

7.1 Rule-Based Reasoning versus CAD Applications

The preceding material presented the notions of a formula, an algorithm and a rule-based system as varying approaches for problem solving in computer-based applications. The approaches are listed in decreasing order of preferences by programmers. The order is related to the amount of effort required for implementation.
Thus, while a formula embodies most commitment to the solution of a programming problem, a rule-based system shows most flexibility in the sense that it encompasses all previous approaches. This makes it a last refuge when the first two approaches seem to be unavailable for implementation. In this vein, the paper shows how well established algorithms are straightforwardly implementable as rule-based systems.
The additional benefit that is emphasized throughout the paper is that a rule-based implementation could make transparent useful features of a solution that are not as explicitly apparent through the first two approaches. This *eagle* view helps in detecting particular instances, as well as potential generalizations and applications, of a given process.
Yet, in spite of the clarity and simplicity of the underlying formulation, the reader may remark that, in the context of B-spline curves and surfaces, the existence and simplicity of the knot insertion algorithm makes the employment of a rule-based process something of overkill. The same remark applies probably as well to the context of subdivision algorithms.
However, in the context of T-splines, the knowledge that the programmer needs in order to perform the task comes packaged as a set of actions to be done in a finite number of situations, which makes it best embodied in a corresponding set of separate rules. The simple loop of the pseudo code of section 2 can then perform the task thus saving the programmer delving into the twists and turns of an algorithm (see [17]) that could be awkward to express anyway.
A prime motivation for writing this paper is actually a desire to put in context the local refinement routine of [16]. Linking that routine to rule-based systems will further make explicit many easy generalizations. In fact, the first generalization that comes to mind here is 3-D T-splines.

7.2 More Directions for Further Research

In a rather different direction, a direct application that best demonstrate the flexibility of this architecture is on the interpolation of points and lines. This is simply obtained by altering the firing of one or the other of the rules of the system so as to lead to the desired interpolation effects (see [2] for a suggestion of how rule alteration can produce certain desirable effects).

Another area for further research is an exploration of how to boost efficiency for any particular application of those systems. In fact, selecting which rule to fire next requires further attention in relation to the amount of explicit knowledge the domain can provide in order to restrict the amount of search involved. Efficiency can drastically be improved by restricting the firing tests on only those rules that are *likely* to fire. These restrictions are easily detected in the implementation of the basic knot insertion algorithm. However, providing an explicit quantification of these benefits in any new application domain can constitute a subject for longer-term research.

8. ACKNOWLEDGEMENT

Thanks are due to Ahmad Nasri for valuable comments on initial versions of this paper. This work was supported by a grant (2007-2008) from the Lebanese Council for Scientific Research.

9. REFERENCES

[1] Abbas, A.; Nasri, A.: Synthesizing data structure requirements from algorithm specifications: case studies from recursive subdivision for computer graphics and animation, ACS/IEEE International Conference on Computer Systems and Applications, 2003.

[2] Abbas, A.; Nasri, A.; Ma, W.: An Approach for Embedding Regular Analytic Shapes within Subdivision Surfaces, 24th Computer Graphics International Conference (CGI' 2006), Zhejiang University, Hangzhou, China, June 26-28, 2006.

[3] Abbas, A.: T-splines: a closer look (in preparation).

[4] Bezdek, J.-C.; Keller, J.; Krisnapuram, R.; Pal, N.-R.: Fuzzy Models and Algorithms for Pattern Recognition and Image Processing, Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[5] Catmull, E.; Clark, J.: Recursively Generated B-Spline Surfaces On Arbitrary Topological Meshes, Seminal Graphics, Ed. Rosalee Wolfe, ACM Press, ISBN 1-58113-052-X, 1998, 183-188.

[6] Friedman-Hill, E.: Jess in Action: Rule-Based Systems in Java, Manning Publications, July 2003.

[7] Goldman, R.; Lyche, T. (editors): Knot Insertion and Deletion Algorithms for B-Spline Curves and Surfaces, SIAM Publication, 1993.

[8] Haykin S.: Neural Networks, 2nd Edition, Prentice Hall, 1999.

[9] Jackson, P.: Introduction to Expert Systems – 3rd Ed., Addison-Wesley, 1999.

[10] McAllester, D.: Truth Maintenance, Proceedings of the Eighth National Conference on Artificial Intelligence, Melno Park, CA. AAAI Press, 1109-1116, 1990.

[11] Nasri, A.: Sinno, K..; Zheng, J.; Sederverg, T.: Skinning T-spline Surfaces, Technical report TR-01-07, Department of Computer Science, American University of Beirut, 2007.

[12] Piegl, L.; Tiller, W.: The NURBS Book, Second Edition, Springer Velag, 1997.

[13] Rayward-Smith, V.-J.; Osman, I.-H.; Reeves, C.-R.; Smith G.-D. (editors): Modern Heuristic Search Methods, John Wiley & Sons, December, 1996.

[14] Roger, D.-F.: An Introduction to NURBS with Historical Perspective, Morgan Kaufmann Publishers 2001.

[15] Russell, S.; Norvig, P.: Artificial Intelligence: a modern approach (2nd Edition), Prentice Hall, 2003.

[16] Sederberg, T.-W.; Cardon, D.-L.; Finnigan, G.-T.; North, N.-S.; Zheng, J.; Lyche, T.: T-Spline Simplification and Local Refinement, ACM Transactions on Graphics, 23(3), 2004.

[17] Sederberg, T.-W.; Zheng, J.; Bakenov, A.; Nasri, A.: T-Splines and T-NURCCS, ACM Transactions on Graphics, 22(3), 477-484, 2003.

[18] Weiss, M.; Stetter, F.: A hierarchical blackboard architecture for distributed AI systems, Proceedings of the fourth International Conference on Software Engineering and Knowledge Engineering, Capri, Italy, 349-355, 15-20 June 1992.

[19] Williams, M.-A.: Applications of Belief Revision, International Seminar on Logic Databases and the Meaning of Change, Transactions and Change in Logic Databases, Lecture Notes In Computer Science; Vol. 1472, 287–316, Springer-Verlag, London, UK, 1996.

[20] http:// www.tsplines.com