Title:
T-Spline Local Refinement as a Belief Revision System: A Rule-based Implementation

Authors:
Abdulwahed M. Abbas, abbas@balamand.edu.lb, The University of Balamand
Ahmad H. Nasri, anasri@fbsu.edu.sa, Fahad Bin Sultan University

Keywords:
T-spline surfaces, local refinement, belief-revision systems, rule-based systems

DOI: 10.14733/cadconfP.2016.194-199

Introduction:
The best known methods (B-Splines, NURBS [7] and recursive subdivision [5], for examples), for modeling geometric surfaces, start from a so-called control mesh on the basis of which the desired surface may be constructed. In this sense, T-splines[1] [9, 10, and 11] are no different, because modeling there starts from an initial control mesh. However, the initial control mesh here is abstracted away through embedding its constituents (i.e., vertices, edges and faces) inside a two-dimensional grid called a T-mesh, presumably considered within the knot domain.

By comparison with other modeling schemes, a T-mesh permits fewer connections between pairs of control points. Moreover, the structure of the T-mesh allows for the automatic inference of the knot information associated with its constituent control points [9]. More importantly perhaps, it comes equipped with a refinement routine whose local effects are comparable to the classical curve knot insertion [3]. However, as already published [9, 10], T-spline local refinement is cast in an algorithmic language lacking the mathematic decisiveness manifested in curve knot insertion, keeping in the shadow many of the details that need to be available for implementation purposes.

Accordingly, this paper gives a tutorial exposition of this process cast in the spirit of a belief revision metaphor [8], and provides a detailed rule-based implementation [2] of that. The naturalness of this interpretation motivates the introduction of an edge-insertion feature that is never explicit anywhere in previously published versions of this routine. Moreover, applications of this routine will multiply: e.g., T-spline polygonal complexes [4] and T-spline skinning [6], to name just a couple. Finally, the data structure [1] brought into call, to satisfy the requirements of the implementation of local refinement viewed as such, is worthy of note in this context.

Main Idea:
T-spline local refinement largely misses the mathematical decisiveness encountered in the curve knot insertion. It is described instead using an algorithmic language that may be summarized as follows: the initial T-mesh starts in a consistent state, where each control point has its associated knot information inferred with respect to the relative positions of neighboring control points and their connecting edges. Consequently, when a new knot is inserted in the T-mesh plane, consistency is violated in the sense that some knot information associated with one or more control points of the T-mesh may no longer be valid. This would trigger a revision process for the purpose of restoring consistency, which may involve the addition of more control points (and edges) to the T-mesh.

The above language is probably better understood when associated with the metaphor of truth maintenance and belief revision systems; classical topics more frequently encountered in traditional Artificial Intelligence research [8].

[1] Our reference to T-splines here is restricted to the specifications of that mentioned in ref. [9] and [10], thus excluding any new features that may have been added to this technology since then [10].

Furthermore, in the absence of a firm mathematical theory, or a robust algorithm, to support this metaphor, the recommended mechanism in similar Artificial Intelligence situations suggests a rule-based approach for the implementation of T-spline local refinement [3]. This association reveals that the first published version of T-spline local refinement [10] is cast as a backward-chaining rule-based system, while the second version [9] is cast as a forward-chaining rule-based system.

This association also offers an explanation of the ailments suffered by both versions of local refinement, since these are, in fact, similar to the ailments suffered by those corresponding systems. Consequently, the cure can draw on the long and well-established history of research in those domains.

In this respect, given a local knot vector $\boldsymbol{t} = [t_0, t_1, t_2, t_3, t_4]$ and $N(t) = N[t_0, t_1, t_2, t_3, t_4](t)$ is the cubic B-spline basis function, one of the following rules (taken from [9]) should fire whenever a knot t_* is inserted in the corresponding knot vector:

Rule 1. If $\boldsymbol{t}' = [t_0, t_*, t_1, t_2, t_3, t_4]$ then $N(t) = c_1 N[t_0, t_*, t_1, t_2, t_3](t) + d_1 N[t_*, t_1, t_2, t_3, t_4](t)$
where $c_1 = (t_* - t_0)/(t_3 - t_0)$ and $d_1 = 1$

Rule 2. If $\boldsymbol{t}' = [t_0, t_1, t_*, t_2, t_3, t_4]$ then $N(t) = c_2 N[t_0, t_1, t_*, t_2, t_3](t) + d_2 N[t_1, t_*, t_2, t_3, t_4](t)$
where $c_2 = (t_* - t_0)/(t_3 - t_0)$ and $d_2 = (t_4 - t_*)/(t_4 - t_1)$

Rule 3. If $\boldsymbol{t}' = [t_0, t_1, t_2, t_*, t_3, t_4]$ then $N(t) = c_3 N[t_0, t_1, t_2, t_*, t_3](t) + d_3 N[t_1, t_2, t_*, t_3, t_4](t)$
where $c_3 = (t_* - t_0)/(t_3 - t_0)$ and $d_3 = (t_4 - t_*)/(t_4 - t_1)$

Rule 4. If $\boldsymbol{t}' = [t_0, t_1, t_2, t_3, t_*, t_4]$ then $N(t) = c_4 N[t_0, t_1, t_2, t_3, t_*](t) + d_4 N[t_1, t_2, t_3, t_*, t_4](t)$
where $c_4 = 1$ and $d_4 = (t_4 - t_*)/(t_4 - t_1)$

Rule 5. If $t_* \le t_0$ or $t_* \ge t_4$, N t does not change

Each of the above rules specify *how an inconsistent knot vector can be split to counter the effects of the inserted knot and the weights that should be* associated with each split component in order to preserve the integrity of the T-mesh in the original state this vector is coming from.

Thus, the control that governs a rule-based system [3] is to repeatedly select and fire a rule from the above list to eliminate any inconsistency detected in the T-mesh until no such inconsistencies exist.

Curve Knot Insertion as a Rule-Based System

This subsection is included here for the purpose of motivating our implementation of T-spline local refinement. In fact, it describes an alternative formulation of curve knot insertion as a rule-based system. The data structure associated with this interpretation is depicted in Fig. 1.

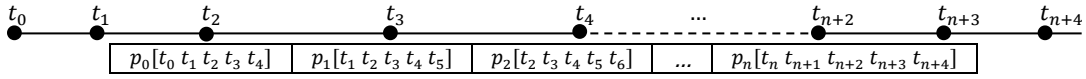

Fig. 1: Data Structure for Alternative Formulation of Knot Insertion.

Under the assumption that the curve is a cubic B-spline curve, this data structure is composed of a list of vertices $< p, \boldsymbol{t} >$, where p is a control point and $\boldsymbol{t}$ is a local vector consisting of 5 consecutive knots of the global knot vector (τ). This way, when the inserted knot t_* falls within the span of a local knot vector of a vertex of this sequence, an appropriate rule would be invoked, thereby replacing this vertex by two vertices and thus restoring consistency to this particular slot of the sequence.

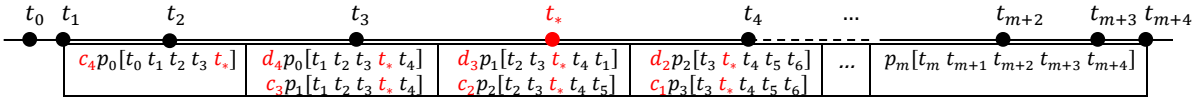

Fig. 2: The effects of Knot Insertion using appropriate rules.

Each vertex of the pair replacing the initial vertex will be placed in the appropriate slot of the sequence specified by the middle knot of the corresponding knot vector. This means that some of these slots should be designed to receive more than a single vertex (see Fig. 2). However, various vertices within the same slot of the sequence will end up being associated with the same local knot vector. The final step of the routine will add the vertices of each slot together to end up forming the refined control polygon emanating from the original one the whole process started off with in Fig. 1.

T-Spline Local Refinement as a Rule-Based System

The data structure associated with this process is designed on the basis of the requirements of the problem at hand [1].

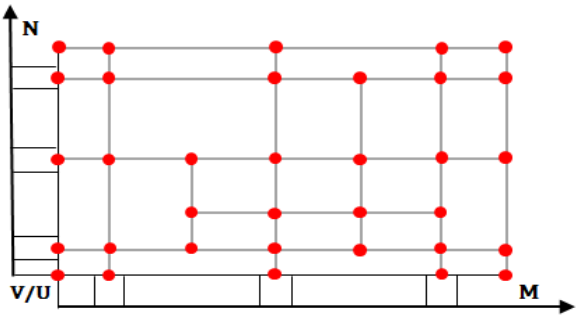

Fig. 3: The T-Spline Mesh (T-Mesh).

This data structure (partly depicted in Fig. 3) revolves around a grid composed of the following items:

- Grid dimensions: M and N, where M is the number of columns and N is the number of rows.
- The global knot vectors: U and V, where U has M knots and V has N knots. These vectors are extendable in order to permit the accommodation of possibly additional knots.
- The grid G itself is simply an $M \times N$ sparse matrix of nodes. This matrix is sparse in the sense that a sizable number of its elements are non-existent. Moreover, in the same way as the vectors U and V, the grid is row-wise and column-wise extendible, as this may be needed in order to accommodate more nodes that are required to be inserted on additional rows and/or columns.
  Note here that, in order to preserve consistency, a newly inserted row into the grid needs to correspond to a new node that is inserted in the knot vector V. In the same way, a newly inserted column into the grid needs to correspond to a new node that is inserted in the knot vector U.
  Thus, a node element existing at position $< i, j >$ of the matrix will also have knot coordinates $< U[i], V[i] >$. These latter coordinates will remain constant no matter how many times and in how many different ways the grid is extended.
  Moreover, this matrix representation provides a low-cost method of navigation between various elements of the grid, which will need to be performed very frequently during the local refinement process.
- A node has four associated knot values: L (*left*), R (*right*), U (*up*) and D (*down*). These are used to mark the existence of an edge joining a pair of node elements of the grid. A default value is assigned to the corresponding field in case such an edge is non-existent.
- Finally, a node is also associated with a sequence of *vertices* residing at that position of the grid. Initially, such a sequence has only one element having two local knot vectors

u and v determined by the associated knot inference process. This sequence indicates that the data structure being described here is a straightforward generalization of that described for the curve case in Fig. 1 and Fig. 2.

At termination, all vertex elements of each sequence will have identical local knot vectors, which permit their final collapse into a single *vertex* through addition.

Note here that, while, in the curve case, revision can be accomplished in a single scan of the initial polygon, this can rarely be done in the surface case. In fact, two additional factors will need to be taken into account in this context:

- When a vertex is split, it will carry with it the other knot vector of the current vertex to each of the split halves half. This might cause a new kind of inconsistency. In fact, some knots associated with a given vertex might not be supported by an actual knot in the current T-mesh state. Such an inconsistency might need repeated point and edge insertion in order to cure.
- Finally, the data structure will also require a revision list in order to keep tract of potentially inconsistent vertices possibly requiring revision. This list should reduce the amount of search needed for detecting these inconsistencies over the whole grid.

Accordingly, the control structure that governs local refinement may be summarized as follows:

Insert first knot pair to the grid
add potentially affected vertices to revision list
loop
select an element from revision list
revise this element,
which may involve splitting a vertex,
inserting a point or an edge
add potentially effected vertices to revision list
remove this element from revision list
until revision list is empty
tidy up

Finally, the decision as to whether any given vertex in the revision list requires revision and what kind of revision is needed rests with a matching process that compares the currently associated local knot vector with the inferred version of this knot vector of the same given vertex.

T-Spline Local Refinement Example

We conclude with a local refinement Example. In this example, the starting configuration of the T-mesh is depicted in Fig. 4 (a).

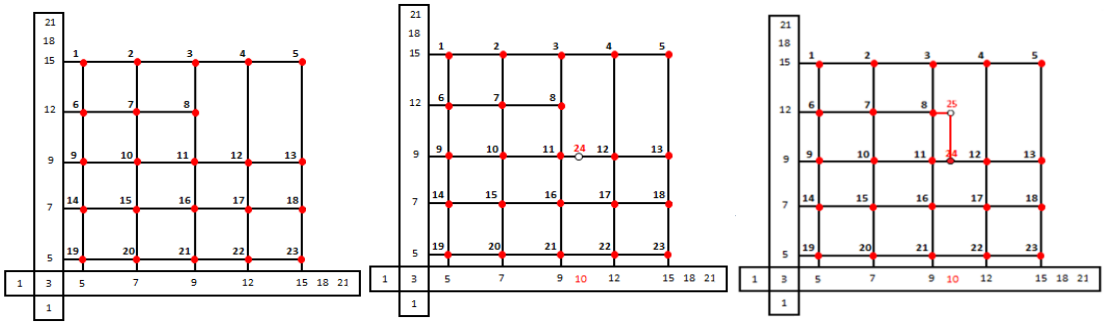

Fig. 4: The T-Spline Local Refinement Example: (a) Initial State (b) Intermediate State (c) Final State.

Insert at knot position **<10, 9>** in Fig. 4 (a).
Vlist: **{10, 11, 12, 13}** along the ***u*** direction
Revise 13: match old **u**: **[9 12 15 18 21]**
and new **u**: **[10 12 15 18 21]** of **13**

Split u into: **[9 10 12 15 18]** and **[10 12 15 18 21]** ⇒ no further revision is needed here since same ***v*** in all newly arising cases
Revise 12: match old **u**: **[7 9 12 15 18]**
and new **u**: **[9 10 12 15 18]** of **12**
Split u into: **[7 9 10 12 15]** and **[9 10 12 15 18]** ⇒ no further revision is needed here since same ***v*** in all newly arising cases
Revise 11: match old **u**: **[5 7 9 12 15]**
and new **u**: **[5 7 9 10 12]** of **11**
Split u into: **[5 7 9 10 12]** and **[7 9 10 12 15]** ⇒ no further revision is needed for first half since same ***v*** in all newly arising cases, besides, **24** is now fully initialized
Revision for second half: match old **v**: **[5 7 9 12 15]**
and new **v**: **[5 7 9 15 18]** of **24**
Insert at knot position **<10, 12>** in Fig. 4 (b): insert **edge** to closest node and in Fig. 4 (c).

Vlist: {7 and 8} along the ***u*** direction, no further revision is needed here since same ***v*** in all newly arising cases and only **{24}** along the ***v*** direction, no further revision is needed here since same ***u*** in this newly arising case.

The simplicity of the example presented above is dictated by the maximum size allowed for this abstract. In this respect, one warning is due. In fact, not all executions of our implementation are expected to terminate so quickly, as this is dependent on the complexity of the selected case.

In fact, seeing that a T-mesh is nothing but a deformation of a regular grid obtained by the removal a number of vertices and edges of this grid, one would not be incorrect in judging that the complexity of any given case is directly dependent on how badly a deformation is the associated T-mesh from a regular grid (see Fig. 5, for example).

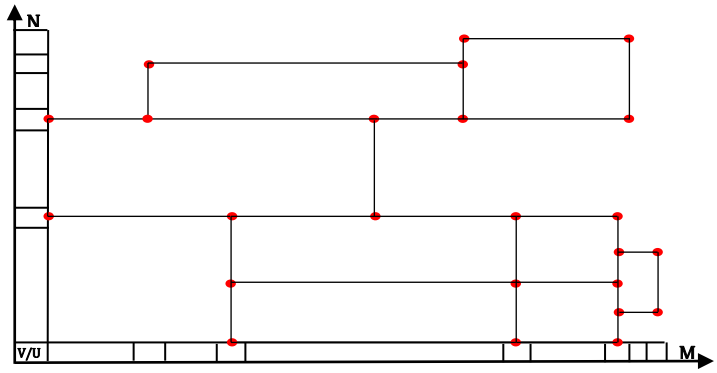

Fig. 5: A Potential Source of a Worst Case Scenario.

Revising the consequences of a knot inserted at arbitrary position of the T-mesh depicted in Fig. 5, our implementation will considerably take more time, as all forward-chaining rule-based system might do.

However, the implementation will never go into an infinite loop. In fact, the worst case scenario will end up with the T-mesh becoming a fully regular grid, with maybe one additional row or one additional column or both added to it.

Conclusions:

Given the nature of this algorithm and the frequency of its use in any given application [4], its implementation has to respond to a number of concerns:

- Efficiency in terms of time and space
- The conflicts that a knot causes should be detected easily preferably without causing too many checks, especially when these checks could possibly become exhaustive.
- Determining the nature of the conflict should be performed efficiently so as to quickly determine how to resolve it.
- The effects of the algorithm should remain local with respect to entirety of the T-mesh.

- Finally during processing, when the system reaches a situation where more than one alternative may be taken, the system should be able to decide on the optimal route to take, since the algorithm does not have the means to backtrack from the position it is currently at.

In this respect, one would hope that the view of local refinement as a belief revision system serves to widen the scope of the discussion that can be made in this context. Moreover, the implementation of this process following a rule-based architecture will add to its flexibility thus making it possible to accommodate more useful features such as edge-insertion, which has never been explicitly mentioned in the original literature on the subject.

Acknowledgement:
The research work reported herein is supported by a grant from the Lebanese Council for Scientific Research, for the academic year 2015-2016. Thanks are also due to Lama Ouayjan for implementing the programming system that generated the figures included in this paper.

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. http://dx.doi.org/10.1109/aiccsa.2003.1227520

[2] Abbas, A.: A Framework for Rule-Based Systems in CAGD Software, the 7th International CAD Conference and Exhibition, Honolulu, Hawaii, June 25-29, 2007.

[3] Abbas, A.: A Generic Rule-Based Formulation of Knot Insertion across CAD Applications, Computer-Aided Design and Applications, 5(6), 2008, 831-840. http://dx.doi.org/10.3722/cadaps.2008.831-840

[4] Abbas, A.: T-Spline Polygonal Complexes, Computer Aided Design & Applications, 12(4), 2015, 465-474. http://dx.doi.org/10.1080/16864360.2014.997643

[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] Nasri, A.- H.; Sinno, K.; Zheng, J.: Local T-spline surface skinning, The Visual Computer, 28(6-8), 2012, 787-797. http://dx.doi.org/10.1007/s00371-012-0692-1

[7] Piegl, L.; Tiller, W.: The NURBS Book, Second Edition, Springer Verlag, 1997. http://dx.doi.org/10.1007/978-3-642-59223-2

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

[9] 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. http://dx.doi.org/10.1145/1015706.1015715

[10] Sederberg, T.-W.; Zheng, J.; Bakenov, A.; Nasri, A.: T-Splines and T-NURCCS, ACM Transactions on Graphics, 22(3), 477-484, 2003. http://dx.doi.org/10.1145/882262.882295

[11] T-splines, http:// www.tsplines.com