Dear Choy,

just a few addition to Jan's explanation. I studied computer graphics and geometry at university, and worked with splines for a number of years, also professionally as a game developer.

I will start with a summary and bore you with details further below.

Pros & cons of gml:CubicSpline (C2 continuous natural cubic spline):
+ (PRO) compact representation = less coordinates in GML document (because to satisfy C2 continuity you have to figure out the 1st/2nd derivatives yourself by solving a tridiagonal system of N linear equations with N unknowns, where N is the number of control points in gml:CubicSpline)
+ (PRO) good visual smoothness
- (CON) more complex algorithm for determining the curve shape. I am willing to bet 100 EUR that some vendors will take shortcuts, and just implement whatever random cubic spline they had in their software until now passing through the points specified in gml:CubicSpline. And we will have the same situation as with BUFR SIGWX, where everyone draws the spline somewhat differently.
- (CON) not available directly in drawing APIs = more math/programming for everyone
- (CON) no straightforward translation to SVG or PDF
- (CON) lack of local control (I will explain at the end why this is a problem for us, if you survive it to the end of the e-mail) - (PRO/CON) I was hoping to see the C2 natural cubic splines implemented at least somewhere during my lifetime to be able to play with them (which makes trying to talk you out of gml:CubicSpline a rather sad effort)

Pros & cons of gml:Bezier:
+ (PRO) Every 2D drawing API (GDI, GDI+, Direct2D, Cairo, Skia, AGG, Qt QPainter, ...) supports Bezier cubic splines out of the box.
+ (PRO) If you are UK/US WAFC and have your data as gml:CubicSpline and you know the algorithm, translation to gml:Bezier is straightforward for you as the originator, because you already have the algorithm implemented
+ (PRO) Translation to SVG / PDF Beziers trivial
- (CON) More coordinates than gml:CubicSpline (roughly 2x due to Bezier control points that are directly related to 1st order derivatives at segment endpoints)

Now the details:

The WAFCs in my opinion made a striking omission in the past with BUFR SIGWX, where they only specified that the points stored in the BUFR should be interpreted as "cubic splines". But they never actually said which particular type of a cubic spline one should use out of the myriad available. This resulted in SIGWX visualisations which differ from one software to another. We at IBL are using "cardinal splines" with zero tension and a particularly constructed knot sequence. Other vendors are using whatever other cubic spline they feel like using. There can be big visual differences between e.g. "cardinal cubic splines", and "natural cubic splines".

The GM_CubicSpline and its gml:CubicSpline counterpart are very unfortunately named, with a term used for the entire class of all cubic splines to denote a very specific kind of them. You would be led to believe these can express cubic splines in general. But according to the information we found, these are so called "natural cubic splines".

"Natural cubic splines" (sometimes also called "smooth cubic splines") in geometry denote C2 continuous cubic splines. C2 continuity requires first and second derivative of the polynomial to be continuous where

the successive spline segments meet. This is why gml:CubicSpline only requires tangents at spline endpoints, but not in-between (because you can calculate the rest if you know how). There is a textbook algorithm for solving the natural splines described in the book excerpt that Jan linked...

Jan made a small mistake saying that a Bezier and gml:CubicSpline are equivalent and can be converted back and forth - this is true only in one direction from gml:CubicSpline to gml:Bezier, but not in reverse. Bezier splines are general, they can express any cubic spline (cardinal, Catmull-Rom, Kochanek-Bartels, anything). But gml:CubicSpline is constrained by the 2nd derivative continuity requirement = more general cubic splines can not be represented as a gml:CubicSpline at all.

These C2 continuous natural cubic splines are teached at universities. That was the first and last time I encountered them until now. How they found their way into GML under the generic "CubicSpline" term amazes me. The natural cubics never gained much popularity in computer graphics. They are not available in vector editing tools like Adobe Illustrator, or Inkscape. No drawing API (that we know of) supports them. Drawing APIs, or vector formats like SVG and PDF simply implement the trusty basic Bezier splines. TrueType fonts also rely on quadratic and cubic Bezier splines.

If gml:Bezier would be used instead, any vendor will be able to understand it out of the box, since Beziers are ubiquitous.

If the UK WAFC does indeed produce gml:CubicSpline (C2 continuous natural cubic spline), for them conversion gml:Bezier is a few lines of code (if you know the spline points and 1st derivatives, then conversion to a Bezier is very straightforward with a couple of additions and multiplications per each point).

For recipients of IWXXM SIGWX with gml:CubicSpline, they would have to go through a bigger ordeal:

1. Set up a tridiagonal set of N linear equations with N unknowns
2. Solve the system
3. Convert to a Bezier spline anyway, because the graphics APIs like GDI, GDI+, Direct2D, Cairo, AGG, Skia, Qt QPainter etc. expect Beziers on input

If you survived it this far, I will explain what is "lack of local control".

Because natural cubic spline (gml:CubicSpline) requires C2 continuity, all the spline points contribute to the equation. If a spline has 200 control points and you move 1 point, the shape of the entire curve changes trying to satisfy the C2 continuity from beginning to the end.

This makes computations slow, because you have to recalculate the entire curve shape completely each time you move anything. Computer graphics artists also do not like C2 continuous natural cubics because of this lack of local control, because moving a point keeps changing your curve in areas where you do not want it to change anymore.

On the other hand if you have a simple Bezier, moving a control point only affects a small area around the point that you have moved. This is a very important property for writing efficient numerical approximation algorithms, to be able to make only local adjustments to the curve, instead of the curve changing shape all around.

For example, we have algorithms that approximate freehand drawn curves using mouse/tablet, or isocontours traced from NWP models with C1 continous cardinal splines. However, if we had to do that with C2 continuous natural cubic splines, that would break the approximation algorithm completely, as it

would not only turn from linear to quadratic time complexity, but it would also have trouble converging to a solution, due to the fact that changing a single point changes the entire spline shape...

Very long e-mail, but I hope that some of this will convince you. Especially the bit about out of the box support for Beziers in APIs and formats. If gml:CubicSpline is adopted, data consumers will be taken by surprise of "what is this uncommon thing" (or they will take a shortcut and just use whatever spline they have in their code already and make it pass through the control points written in gml:CubicSpline).

By using gml:Bezier, no one can complain that the curve is complex, or uncommon, and everybody will be drawing exactly the same shape.

Well at least in EPSG:4326 CRS. What happens when people display a chart in polar stereographic projection is a question:

- Either the cubic spline is defined strictly in EPSG:4326, and when reprojecting to polar stereographic you have to subdivide the spline to small linear line segments and reproject those.

- Or people will just reproject the spline control points to polar stereographic, and then construct a cubic spline in polar stereographic passing through these points (which is what most probably everybody is doing right now with BUFR SIGWX).

Best reagards,
Boris

On 15-Mar-21 22:01, Jan Korosi wrote:

Hi Choy,

I discussed the **gml:CubicSpline** usage in the SIGWX forecast with Boris.

It is not clear what kind of spline it is (at least by definition available at WMO schemas page). We found the definition at google book preview.

GM_CubicSpline

A cubic spline (class GM_CubicSpline) consists of a sequence of segments each with its own defining function. A cubic spline uses the control points and a set of derivative parameters to define a piecewise 3rd degree polynomial interpolation.

The function describing the curve must have a continuous 1st and 2nd derivative at all points and pass through the controlPoints in the order given. Between each pair of the control points, a curve segment is defined by a cubic polynomial. At each control point, the polynomial changes in such a manner that the 1st and 2nd derivative vectors are the same from either side.

A special provision must be made for the first and last point of the spline because the tangent at these points remains undefined. The control parameters record must contain a vectorAtStart and a vectorAtEnd as these are the unit tangent vectors at controlPoint[1] and controlPoint[n] where n = controlPoint.count.

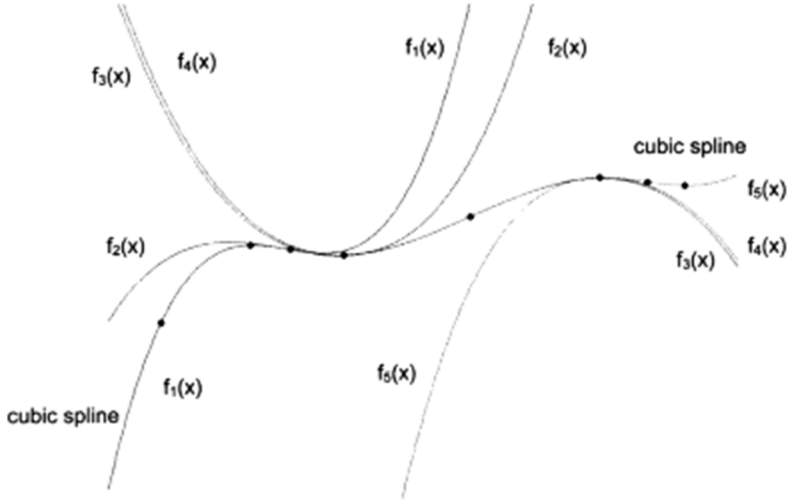

Fig. 4.11. GM_CubicSpline

Function 1: $f_1(x) = a_{01} + a_{11}*x + a_{21}*x^2 + a_{31}*x^3$
Function 2: $f_2(x) = a_{02} + a_{12}*x + a_{22}*x^2 + a_{32}*x^3$

Function 5: $f_5(x) = a_{05} + a_{15}*x + a_{25}*x^2 + a_{35}*x^3$
The first and the last segment are a part of the functions f_1 and f_5 respectively (Nitschke 2003).

Boris explained to me that it is called **natural cubic spline**. We do not implement it because it is not appropriate for the editing of curves. The reason is that you don't have a local control (over the shape of the curve). Any change in any segment (a line between control points) may change the whole curve. In other words, you have to calculate the entire curve each time you change only one control point. For this reason, we implement the **Hermit spline**, which provides local control.

We discussed the option to implement a natural cubic spline. Many of our customers are used to producing derivated products from SWH and SWM WAFS forecasts. So it is logical to be able to edit curves directly, despite we think it could not be so userfriendly. Unfortunately, we don't know about any graphic software which implements it to try it.
The second option is to convert natural cubic spline to, e.g. bezier, the user will edit bezier and subsequently convert bezier back to natural cubic spline. So we can still use our already developed algorithms for the processing of curves (e.g. simplification of curves). On the other hand, each unnecessary conversion means a lost of precisions and introduces an additional computation error.

In the end, despite that Boris would like to see natural cubic spline ;-), we recommend replacing **gml:CubicSpline** with **gml:Bezier** (degree of 3).

There are two reasons:

1. It is easy to convert gml:CubicSpline to gml:Bezier. The WAFS developers can easily extend the current implementation of automatic SIGWX forecast generator. They only need to compute control points (e.g. https://math.stackexchange.com/questions/3770662/convert-cubic-spline-to-b%C3%A9zier-curve-and-get-control-points).
2. All widely used vector render engines have already implemented Beziers (e.g. QPainter, Cairo, AGG, OpenGL, GDI, PDF, SVG, ...). The availability of ready-to-use and tested libraries will significantly decrease the implementation requirements on the consumer side.

I believe that Boris will correct me or add missing info.

Best regards,
Jan