Cubic and Hermite splines for VTK

Release 1.00

Nicole Z. Kovacs[1], Terry M. Peters[1,2,3] and Elvis C. S. Chen[1,2,3]

August 19, 2016

[1]Imaging Research Laboratories, Robarts Research Institute, University of Western Ontario, Canada
[2]Biomedical Engineering Graduate Program, University of Western Ontario, Canada
[3]Department of Medical Biophysics, University of Western Ontario, Canada

Abstract

A cubic spline is a spline where each curve is defined by a third-order polynomial, while a Hermite spline has each polynomial specified in Hermite form, being computed using tangent information as well as the position of the points. We propose two new classes for VTK, vtkCubicSpline and vtkHermiteSpline, which compute interpolating splines using a Cubic and a Hermite Spline Interpolation function, respectively. We also propose two new auxiliary classes, vtkParametricCubicSpline and vtkParametricHermiteSpline, that create parametric functions for the 1D interpolating aforementioned splines.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3561]

Contents

1 Introduction

Splines are useful tools for interpolation of numeric data to obtain smooth continuous functions. They can be used in computer graphics to visualize piece-wise curves and surfaces, beyond other applications. VTK pro-

vides some spline classes such as vtkCardinalSpline and vtkKochanekSpline, but they lack a few approaches. We propose two new classes for VTK, vtkCubicSpline and vtkHermiteSpline. The first implements cubic splines, having third-degree polynomials for each curve, and the second uses tangent information of the points as well as their position to be computed by implementing the Hermite spline a spline curve where each polynomial of the spline in is Hermite form.

Additionally, we propose two other new classes for VTK, vtkParametricCubicSpline and vtkParametricHermiteSpline, that present parametric functions for the one-dimensional splines mentioned, mapping a single parameter u into a 3D point (x, y, z) using three instances of interpolating Cubic/Hermite splines.

As an example of application, the vtkHermiteSpline class was applied in a research [2][3] on a spatial orientation device used in natural orifice transluminal endoscopic surgery (NOTES). NOTES involves the passage of endoscopic and surgical tools through natural orifices, such as mouth, rectum, or vagina, with subsequent entry into the peritoneum. It is an approach to minimally invasive surgery, where spatial orientation is one of the challenges. In this research, a tracking system that provides certain points and their tangential information was used for 3-dimensional imaging of the shape and orientation of the endoscope. The vtkHermiteSpline class was used to create a spline that represented the endoscope tracked by receiving its tracking information as input.

2 Algorithm

Both classes inherit from vtkSpline, thus being its concrete implementations. Let us explain the theory behind them:

2.1 Cubic Spline

The vtkCubicSpline class implements the Cubic Spline [2], which is represented by:

$$q_k(t) = a_k + b_k t + c_k t^2 + d_k t^3 \tag{1}$$

where $q_k(t)$ is the k^{th} piece of the spline and $t \in [0,1]$. With this information we can deduce that:

$$q_k(0) = p_k = a_k \tag{2}$$
$$q_k(1) = p_{k+1} = a_k + b_k + c_k + d_k \tag{3}$$
$$q'_k(0) = D_k = b_k \tag{4}$$
$$q'_k(1) = b_k + 2c_k + 3d_k \tag{5}$$

Solving equations (2) to (5) yields:

$$a_k = p_k \tag{6}$$
$$b_k = D_k \tag{7}$$
$$c_k = 3(y_{k+1} - y_k) - 2D_k - D_{k+1} \tag{8}$$
$$d_k = 2(y_k - y_{k+1}) + D_k + D_{k+1} \tag{9}$$

The cubic spline also requires that the second derivatives match the control point, thus:

$$q_{k-1}(1) = p_k \tag{10}$$
$$q'_{k-1}(1) = q'_k(0) \tag{11}$$
$$q_k(0) = p_k \tag{12}$$
$$q''_{k-1}(0) = q''_k(0) \tag{13}$$

This results in a tridiagonal system [5] which can be resolved efficiently [4]:

$$\begin{bmatrix} 2 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ \vdots & \cdots & \cdots & \cdots & \cdots & \vdots \\ & & 1 & 4 & 1 & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \vdots \\ D_{n-2} \\ D_{n-1} \\ D_n \end{bmatrix} = \begin{bmatrix} 3(y_1 - y_0) \\ 3(y_2 - y_0) \\ 3(y_3 - y_1) \\ \vdots \\ 3(y_{n-1} - y_{n-3}) \\ 3(y_n - y_{n-2}) \\ 3(y_n - y_{n-1}) \end{bmatrix} \tag{14}$$

For a natural cubic spline, the second derivative at the end points are set to zero. Note that four control points are needed to plot a cubic spline segment.

2.2 HermiteSpline

The vtkHermiteSpline class implements the Hermite Spline [1]. The Hermite interpolation in this algorithm was computed with the aid of 4 basis functions:

$$h_{00} = 2t^3 - 3t^2 + 1 = (1+2t)(1-t)^2 \tag{15}$$
$$h_{10} = t^3 - 2t^2 + t = t(1-t)^2 \tag{16}$$
$$h_{01} = -2t^3 + 3t^2 = t^2(3-2t) \tag{17}$$
$$h_{11} = t^3 - t^2 = t^2(t-1) \tag{18}$$

These functions are plotted in Figure 1. They compose the Hermite interpolation for they directly incorporate tangential function. Given a point p_k at $t = 0$ and a point p_{k+1} at $t = 1$, a tangent m_k at p_k and a tangent m_{k+1} at p_{k+1}, the Hermite interpolation was computed as:

$$q_k(t) = h_{00}p_k + h_{10}m_k + h_{01}p_{k+1} + h_{11}m_{k+1} \tag{19}$$

where $q_k(t)$ is the k^{th} curve of the spline and $t \in [0,1]$.

For an ordered set of n control points with known tangents, a spline can be computed with the ordered pair $\{\{p_1,p_2\},\{p_2,p_3\},...,\{p_{n-1},p_n\}\}$. As $q_k(0) = p_k$, $q_k(1) = p_{k+1}$, $q'_k(0) = m_k$, and $q'_k(1) = m_{k+1}$, the Hermite spline is both C^0 and C^1 continuous, that is, the curves joined with the tangents are equal at the control points. In order to interpolate p in a non-unit (arbitrary) interval $[p_k, p_{k+1}]$, the tangent value must be scaled to:

$$q_k(t) = h_{00}p_k + h_{10}hm_k + h_{01}p_{k+1} + h_{11}hm_{k+1} \tag{20}$$

where $h = p_{k+1} - p_k$ and $t = \frac{(p-p_k)}{h}$.

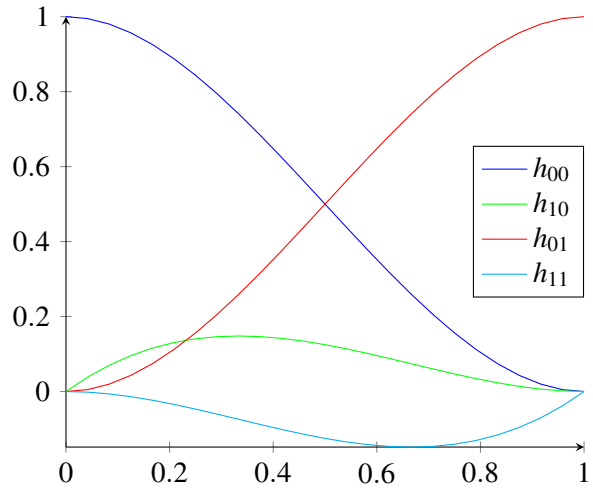
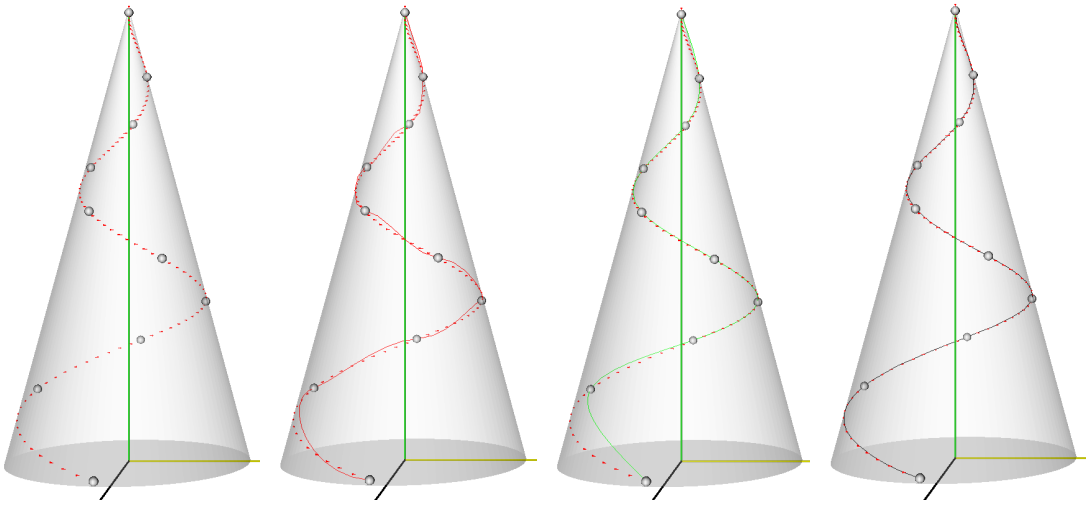


Figure 1: The basis functions for Hermite interpolation. Note that $q_k(0) = p_k$, $q_k(1) = p_(k+1)$, $q'_k(0) = m_k$, and $q'_k(1) = m_(k+1)$, as q_k is a linear combination of these functions.

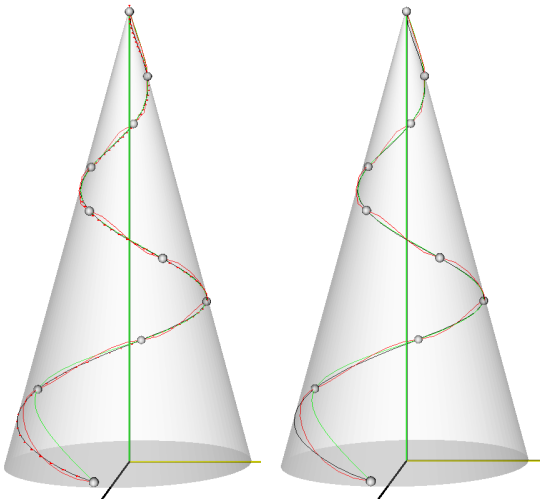
3 Demonstration

As a demonstration, we sampled a set of points from a cone which was translated and rotated. You can see in Figure 2 the x, y, and z axes, a cone for better visualization of the sample points, a dashed red line indicating all the sample points, and spheres representing the ten points given to the splines. The red continuous curves represent the Hermite spline, the green line represents the cubic spline, and the blue line represents the Hermite spline with all the sample points as input (not only 10). We have represented the three splines together in Figure 3 for comparison purposes.



(a) Sample points (b) Hermite spline (c) Cubic spline (d) Hermite spline of all sample points

Figure 2: demonstration of the different splines.



(a) Sample points and all splines

(b) All splines

Figure 3: Comparison between the three representations of splines with and without the sample points plotted.

4 Code Snippet

```
vtkSmartPointer<vtkPoints> splinePoints = readerPoints->GetOutput->GetPoints();
vtkSmartPointer<vtkPoints> splineTangents = readerTangents->GetOutput->GetPoints();

// hermite spline
vtkSmartPointer<vtkHermiteSpline> hSplineX = vtkSmartPointer<vtkHermiteSpline>::New();
vtkSmartPointer<vtkHermiteSpline> hSplineY = vtkSmartPointer<vtkHermiteSpline>::New();
vtkSmartPointer<vtkHermiteSpline> hSplineZ = vtkSmartPointer<vtkHermiteSpline>::New();

vtkSmartPointer<vtkParametricFunctionSource> paraHSplineSource =
 vtkSmartPointer<vtkParametricFunctionSource>::New();
vtkSmartPointer<vtkParametricHermiteSpline> hermiteSpline =
 vtkSmartPointer<vtkParametricHermiteSpline>::New();

hermiteSpline->SetXSpline( hSplineX );
hermiteSpline->SetYSpline( hSplineY );
hermiteSpline->SetZSpline( hSplineZ );

hermiteSpline->SetPoints( splinePoints );
hermiteSpline->SetTangents( splineTangents );

paraHSplineSource->SetParametricFunction( hermiteSpline );

// cubic spline
vtkSmartPointer<vtkCubicSpline> cSplineX = vtkSmartPointer<vtkCubicSpline>::New();
vtkSmartPointer<vtkCubicSpline> cSplineY = vtkSmartPointer<vtkCubicSpline>::New();
```

```
vtkSmartPointer<vtkCubicSpline> cSplineZ = vtkSmartPointer<vtkCubicSpline>::New();

vtkSmartPointer<vtkParametricFunctionSource> paraCubicSplineSource =
 vtkSmartPointer<vtkParametricFunctionSource>::New();
vtkSmartPointer<vtkParametricCubicSpline> cubicSpline =
 vtkSmartPointer<vtkParametricCubicSpline>::New();

cubicSpline->SetXSpline( cSplineX );
cubicSpline->SetYSpline( cSplineY );
cubicSpline->SetZSpline( cSplineZ );
cubicSpline->SetPoints( splinePoints );

paraCubicSplineSource->SetParametricFunction( cubicSpline );
```

References

[1] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, San Francisco, CA, nov 1987. 2.2

[2] Elvis C. S. Chen, Sharyle A. Fowler, Lawrence C. Hookey, and Randy E. Ellis. Representing flexible endoscope shapes with hermite splines. In Kenneth H. Wong and Michael I. Miga, editors, *Medical Imaging 2010: Visualization, Image-Guided Procedures, and Modeling*, volume 7625. SPIE, feb 2010. 1, 2.1

[3] Sharyle Fowler, Mohamed S. Hefny, Elvis C.S. Chen, Randy E. Ellis, Dale Mercer, Diederick Jalink, Andrew Samis, and Lawrence C. Hookey. A prospective, randomized assessment of a spatial orientation device in natural orifice transluminal endoscopic surgery. *Gastrointestinal Endoscopy*, 73(1):123127, jan 2011. 1

[4] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++*. Cambridge, second edition, 2002. 2.1

[5] Eric W. Weisstein. Cubic spline. http://mathworld.wolfram.com/CubicSpline.html. [Online; accessed 28-July-2015]. 2.1