Uniform B-Splines for the VTK Imaging Pipeline

Release 1.0

D. G. Gobbi[1] and Y. P. Starreveld[2]

February 24, 2011

[1]Seaman Family Centre, Foothills Medical Centre, Calgary, Canada
[2]Department of Clinical Neurosciences, University of Calgary, Canada

Abstract

Uniform B-splines are used widely in image processing because they provide maximal smoothness compared to any other piecewise polynomial of the same degree and support. This paper describes VTK classes for performing two functions: image interpolation via B-splines, and non-rigid coordinate transformation via B-splines. Special attention is paid to different boundary conditions for the ends of the spline: image interpolation supports clamped, mirrored, and repeated boundary conditions while B-spline grid transformations support clamped and zero-past-boundary conditions. The use of these classes for image deformation is demonstrated.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3252]
Distributed under Creative Commons Attribution License

Contents

1 Introduction

Splines have long been used as a means of generating continuous, smooth functions from discretely sampled data sets. Originally their use on images was limited due to their high computational expense, but the theoretical framework of uniform B-spline basis functions led to highly efficient algorithms that have made them practical [1, 2, 3, 4]. When used for image interpolation, B-splines provide the highest degree of continuity for any chosen kernel size.

The smoothness of B-splines has also made them popular as a means of describing free-form deformations in medical imaging. A uniform grid of B-spline knots, in combination with a suitable image similarity metric and optimization algorithm, provides a straightforward method for deformable image registration. Efficient optimization of the knot values is still an area of extensive research [5, 6].

In our previous work, we described a framework for implementing deformable coordinate transformations in VTK, and demonstrated its use on medical images [7]. The two deformable transformations described in that work were based on thin-plate splines and on interpolated grids of deformation vectors; in a later work we utilized the latter to generate a database of electrophysiological brain recordings for Parkinson's surgery within a standard coordinate system [8]. Our current paper describes an expansion of our original set of VTK classes to include image interpolation and deformation via uniform cubic B-splines.

2 New and modified classes

The three classes included in our source package are described below. Important methods of these classes are listed in Appendices A, B, and C.

2.1 B-spline solver: vtkImageBSplineCoefficients

As with any spline, interpolation with a B-spline requires two distinct steps. First, the N data points are converted into N spline knot coefficients by solving a set of linear equations. Second, the piecewise polynomials are constructed from these coefficients and used to generate the spline curves.

It was discovered by Unser et al. [2] that solving for the coefficients does not require an $N \times N$ tridiagonal matrix solver, as was traditionally done for splines, but can instead be done with an efficient back-and-forth filter operation. We have written a class called vtkImageBSplineCoefficents that uses the original code of Thévenaz and Unser [3] to compute the coefficients, but we have multi-threaded the operation and extended it from two dimensions to three dimensions. To maximize the efficiency of the computation while minimizing the memory overhead, we apply the filter in three in-place passes over the output data, one pass per dimension (or fewer passes in the case of lower-dimensionality data). Coefficients can be computed for B-splines of up to degree 9.

Once the image has been converted to a grid of B-spline coefficients (one coefficient per voxel), there are two options for performing the interpolation. For cases where only a few interpolated values are desired, the `Evaluate(x,y,z)` method of the class can be used to compute those values. The alternative is to pass the coefficents as an input to vtkImageReslice, which will then produce a interpolated image volume or slice as its output.

Distributed under Creative Commons Attribution License

2.2 B-spline boundary conditions

As with any spline, it is necessary to establish boundary conditions when solving for the coefficients. This is done by making assumptions about the data that hypothetically lies beyond the edge of the image, and then performing an extrapolation. It is important that the assumptions are reasonable, since the results close to the boundary will depend strongly on what those conditions are.

For our coefficient solver, we provide a choice between three boundary conditions. The first, the `Clamp` condition [4], assumes that the coefficient values at the edge of the image continue indefinitely past the edge. This is the default, since this is the same assumption made by vtkImageReslice when interpolating the image. It gives very good results and its only drawback is a small amount of localized ringing if the pixel values change steeply near the edge. The second condition is the `Mirror` condition, favored by Thévenaz and Unser [3]. As should be apparent from its name, it assumes that the image is repeated in a mirror copy at each boundary, and has the advantage of minimizing edge ringing, although the mirroring effect might be undesired when voxel values are extrapolated in the border region just beyond the edge. The third condition is `Repeat`, which assumes an image that repeats indefinitely. This boundary condition is useful when the image is meant to represent one cycle of a continuous loop, but produces poor results if the image intensity is not continuous from one edge of the image to the opposite edge.

2.3 Image interpolator: vtkImageReslice

We have named our new image interpolation class vtkImageBSplineReslice in our source package, but for the purposes of this discussion we will simply call it vtkImageReslice since we expect our new class to be merged into that class in the near future.

The purpose of vtkImageReslice is to resample an image through a given interpolation kernel after applying a given coordinate transformation to the position of each voxel. For B-splines, the interpolation kernel weights are applied not to the original image, but to the B-spline coefficients that have been computed for the image. In terms of cost, applying a cubic B-spline kernel is no more expensive than applying the non-spline cubic kernel, but it provides $C(1)$ and $C(2)$ continuity instead of just $C(1)$ continuity. There is a hidden cost, of course, in that the coefficients have to be computed beforehand, but for interactive visualization or image registration the coefficients need only be computed once, and then the image can be resliced in as many ways as desired.

Note that vtkImageReslice can only use B-splines of degree 3, and would require further modification to support B-splines of higher degree; in contrast, vtkImageBSplineCoefficients supports up to degree 9. This was not considered to be a significant deficit, since $C(2)$ continuity is all that is required for most purposes, and use of higher degree polynomials for image interpolation is rare. In situations where more precision is required, vtkImageReslice provides sinc-approximating interpolation kernels.

As with vtkImageBSplineCoefficients, vtkImageReslice provides different boundary conditions for dealing with the image border. The default is to clamp the image lookup indices to the bounds of the input image, and to color the output with a user-selectable background color beyond the bounds of the image. The other two, the `Mirror` and `Repeat` conditions, are selectable via the `MirrorOn()` and `WrapOn()` methods. Whichever condition is selected, it must be the same as was chosen for the coefficient solver, or else the interpolated values will be incorrect near the boundary.

Distributed under Creative Commons Attribution License

2.4 Deformable transformation: vtkBSplineTransform

Within the field of medical imaging, perhaps an even more common use for B-splines than image interpolation is image transformation. B-splines are an excellent tool for decribing of smooth deformations, since they are the most efficient means computing a $C(2)$ continuous piecewise polynomial curve. Our vtkBSplineTransform class allows one to interpolate a 2D or 3D grid of B-spline coefficients, stored in a vtkImageData with three vector components, in order to produce smoothly varying deformation vectors at any point in space. The vtkBSplineTransform is a subclass of vtkAbstractTransform, and can therefore be utilized by any VTK filter that accepts a vtkAbstractTransform.

As with any VTK transform, the B-spline transform is invertible. This is not done by recomputing the coefficient grid, which is possible but expensive, but is instead done by setting an "inverse" flag in the transform object that will cause each coordinate transform to be inverted on-the-fly via Newton's method [7]. Since the B-spline has a continuous second derivative and since there is only one root for Newton's method to find, stability of this method is guaranteed. Computing the inverse transformation within a tolerance of 10^{-12} typically requires four or five forward-transformation evaluations.

With regards to boundary conditions, we decided that the `Repeat` and `Mirror` conditions are not relevant for transformations, and instead provide three boundary choices that relate to the locality of the transformation. The default is to clamp the coefficient indices to the bounds of the grid, resulting in a deformation value that converges to a constant value past the edge of the grid. This, essentially, uses the grid to describe a transformation with an infinite spatial extent. In practice, it is useful for ensuring good behavior of the transformation around the border of the grid.

The other two boundary conditions allow the deformation to fall to zero at a small distance from the edge of the B-Spline grid, providing a localized transformation. The `Zero` condition allows the deformation to roll off to zero at a distance of two grid node spacings from the edge while maintaining $C(2)$ continuity. The `ZeroAtBorder` condition forces the deformation to zero at just a single node-spacing beyond the grid, but it does so by sacrificing $C(2)$ and $C(1)$ continuity at the point where the deformation reaches zero.

Often, one will apply a local deformation transformation on top of an affine transformation. For grid transformations we did not wish to constrain the ways in which an affine transformation would be applied, e.g. whether it would be applied in the source coordinate system, in the target coordinate system, or perhaps a separate affine transformation for each coordinate system. Instead, the deformation can be logistically concatenated with other transformations in various ways via the vtkGeneralTransform [7].

3 Results

3.1 B-spline boundary conditions for image deformation

To demonstrate the B-spline transformation, we have used it to approximate a thin-plate spline transformation which we use as ground truth. This is not an unreasonable application of B-splines, since the B-spline grid transformation can be computed with far greater efficiency and serves as a useful approximation. We created a thin-plate spline with eight control points, and concatenated it with the inverse of its base affine transformation so that it decayed to the identity transformation at infinity. It was then sampled onto a 17×17 grid of deformation vectors, which were converted into cubic B-spline coefficients by vtkImageBSplineCoefficients. These were used as the coefficients for vtkBSplineTransform. We described the use of thin-plate splines in VTK in an earlier publication [7].

Distributed under Creative Commons Attribution License

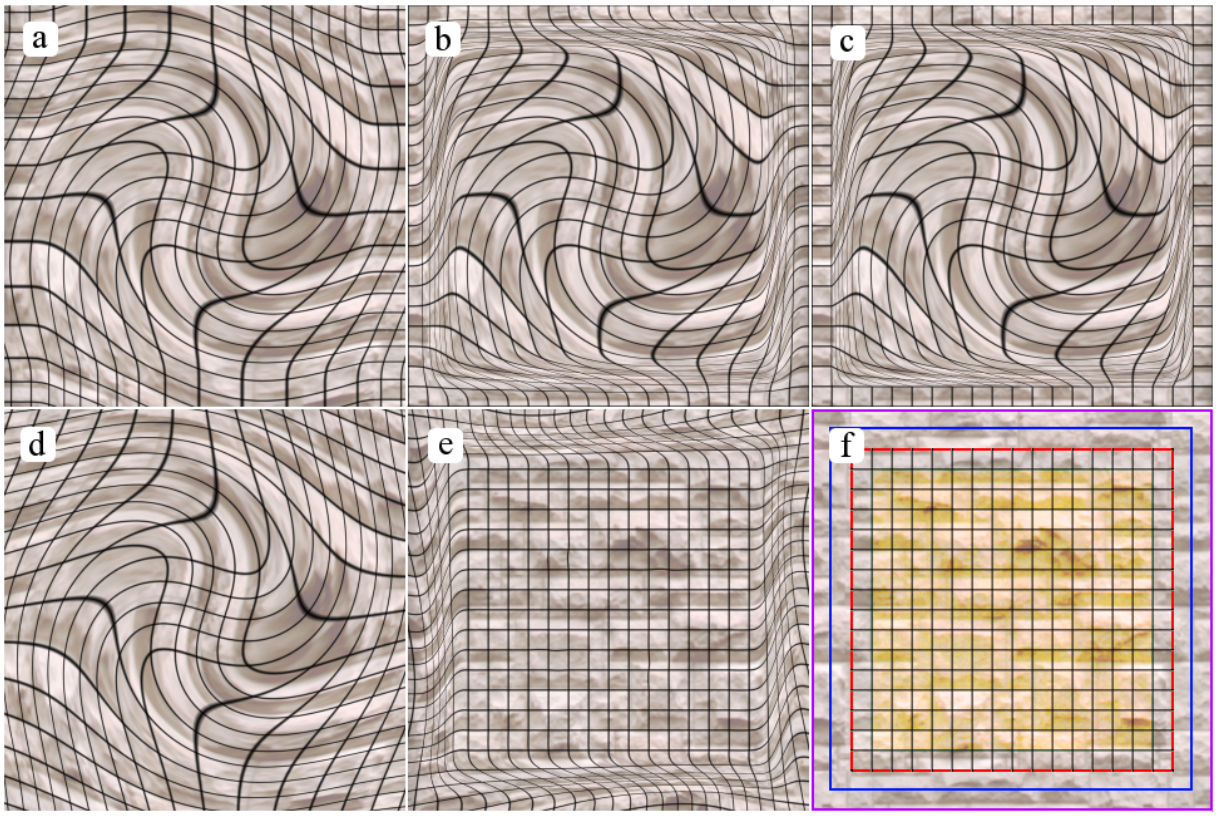

Figure 1: (a) Clamp coefficients to edge value, (b) set coefficients to zero beyond edge, (c) clamp spline to zero at border, (d) original thin-plate spline deformation, (e) portion of transformation b not present in d, (f) the B-Spline grid (black), region of full support (yellow), grid edge (red), border (blue), and limit of influence (mauve). A sufficiently dense grid can model any deformation accurately within the full-support region of the grid. The transform within this region is identical for a, b, and c.

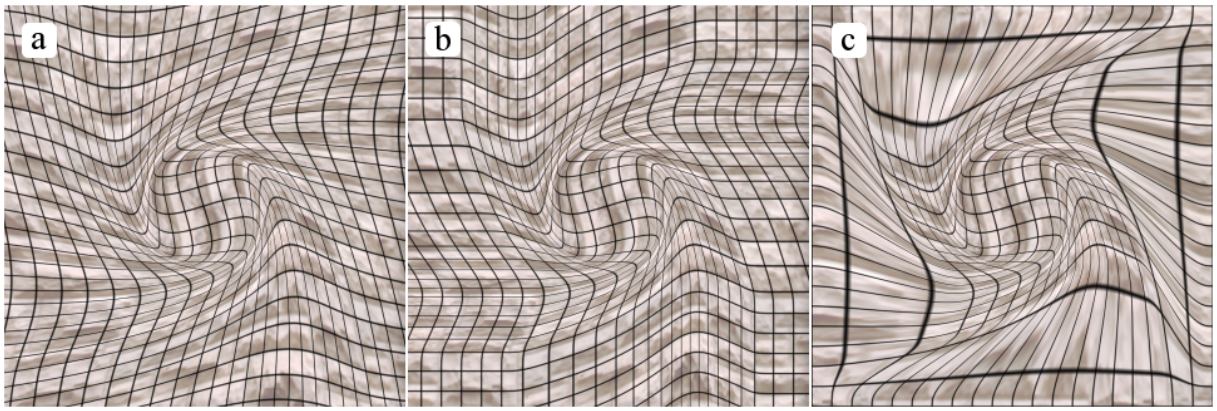

Figure 2: (a) the inverse of the thin-plate spline, (b) the inverse of Fig. 1a, (c) the inverse of Fig. 1b. Ordinarily, a B-spline grid should not be used to model a deformation that is larger than the full-support region of the grid, but (b) shows that clamping the coefficients to their edge values provides a 0th-order extrapolation of the deformation beyond the edge of the grid. Note that the region of full support is not rectangular for the inverse transformation.

Distributed under Creative Commons Attribution License

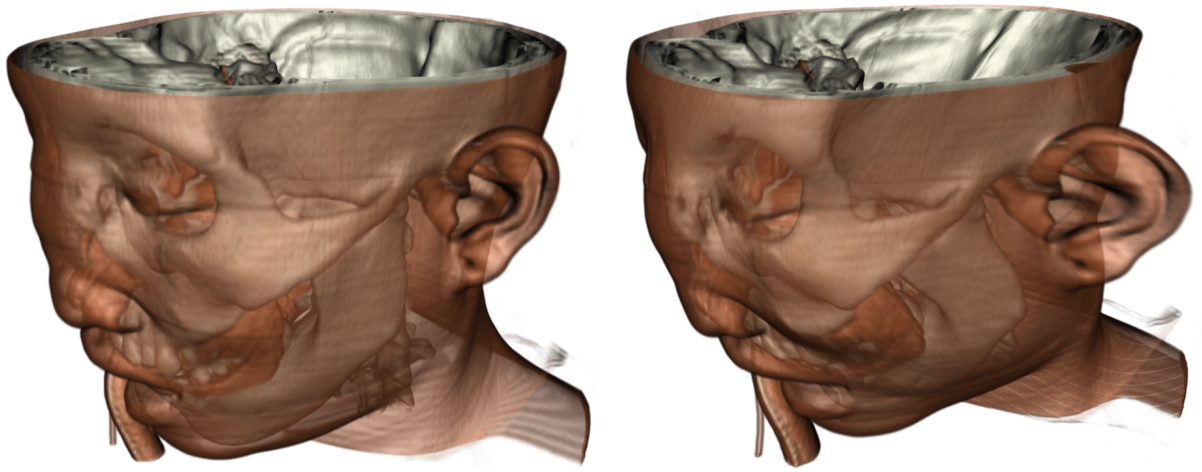

Figure 3: A volume-rendered CT head after super-sampling with B-spline interpolation (left), and after super-sampling with B-spline interpolation through a $5\times5\times5$ B-spline deformation grid (right).

Fig. 1 shows the original thin-plate spline transformation applied to an image of a grid via vtkImageReslice, as well as the B-spline transform applied to the same grid with the three boundary conditions described in the previous section. The images were super-sampled by a factor of four to reduce aliasing. As shown in Fig. 1e, the B-spline faithfully reproduces the thin-plate spline transformation within its region of full support, but does a poor job outside of this region. In practice, the B-spline grid will be used either to model a local deformation, or will be constructed so as to be at least one grid-space larger than the image it is meant to deform. In the latter case, the boundary condition has no influence on the forward transformation, but might influence the inverse transformation.

The inverse transformation is shown in Fig. 2, and it should be noted that the area within which the B-spline matches the original thin-plate spline is no longer rectangular. This is because full-support region is set with respect to the (x,y,z) coordinate system, rather than the (x',y',z') coordinate system.

3.2 Three-dimensional medical image deformation

Cubic B-splines are expected to work well for resampling 3D image sets prior to volume rendering, since their $C(2)$ continuity implies that any vector quantities such as gradients that vary smoothly with position in the original image, will also vary smoothly with position in the resampled image.

To generate the renderings in Fig. 3, we resampled the "headsq" CT that comes with the VTK data package to an isotropic voxel spacing of 0.3 mm with cubic B-splines, using vtkImageBSplineCoefficients and vtkImageReslice to perform the resampling, and then rendered the result with vtkFixedPointVolumeRayCastMapper. The gradient opacity function was used to make the skin translucent and the bone surface opaque, while leaving most of the tissue transparent.

The image on the right has been transformed with a B-spline grid of random (x,y,z) coefficients, resulting in an intentionally severe deformation of the head. Apart from the deformation, the appearance is very similar to the undeformed rendering. The primary discrepancies are the specular highlighting surface opacity, which are due partly to the changes in gradient magnitude that accompany a severe deformation, and due to the changes in orientation of the surfaces with respect to the scene lighting.

Distributed under Creative Commons Attribution License

4 Discussion

Prior to this work, the highest level of image interpolation supported by vtkImageReslice and vtkGridTransform was cubic, which provides $C(1)$ but not $C(2)$ continuity. By adding cubic B-splines, we provide an extra degree of smoothness comes at essentially no additional computational expense. The only caveat, with respect to image interpolation, is that the image must be filtered to compute the B-spline coefficients prior to interpolation.

The class that computes the B-spline coefficients, vtkImageBSplineCoefficients, is itself capable of performing interpolation with B-splines of degree 2 through 9, though it is performs the interpolation at single points and is not by itself capable of producing a resampled image or being used as a VTK transform. We chose to limit vtkImageReslice and vtkBSplineTransform to cubic B-splines because that is the degree that is preferred for most applications, and the use of higher-degree B-splines in practice is rare.

Our straightforward use of Newton's method to compute the inverse of the B-spline grid transformation is also a compromise. Within any piecewise segment of the spline, the inverse can be computed analytically, and if there was a check at each iteration of Newton's method to see if the correct segment had been found then the overall inversion process could be accelerated. We also considered the use of higher-order Householder methods to replace Newton's method, but were concerned that they would not be stable since, globally, the spline is limited to $C(2)$ continuity.

One issue that has not been addressed by these classes is aliasing. When an image is sub-sampled, i.e. reduced in resolution via resampling, details in the original image that are of higher resolution than the resampled image will appear as artifacts. The most common artifacts are staircasing and discontinuous lines that were continuous in the original image. These artifacts can be eliminated by using a band-pass filter to blur the details prior to resampling the image, and for image deformation, the degree of blurring varies across the image since the sampling will not uniform. This is, in fact, what is done in the computer graphics industry to anti-alias texture maps that undergo perspective transformations. For the results presented in this paper, we have avoided aliasing by super-sampling all of our images.

5 Conclusion

We have provided new classes for VTK that provide image resampling and image deformation via uniform cubic B-splines. Our intention is for these classes to become part of the main VTK distribution package, and to maintain them within that context. One aspect of B-splines that we have paid particular attention to is boundary conditions, which we realized would vary depending on the application. For image interpolation, we provide clamped, mirrored, and repeated boundary conditions, and for B-spline deformations we provide a clamped boundary condition, as well as two fall-to-zero boundary for use with local deformations.

Acknowledgment

We send our sincere appreciation to Dr. Philippe Thévenaz of the Biomedical Imaging Group at EPFL, Switzerland, for providing us with permission to use his B-spline image interpolation code in our work.

Distributed under Creative Commons Attribution License

Appendix A: Selected vtkImageBSplineCoefficients interface methods

```
class vtkImageBSplineCoefficients : public vtkThreadedImageAlgorithm
{
    public:
         void SetInput(vtkDataObject* input);   set the image to be interpolated [1]
vtkImageData* GetOutput();                      get the B-spline coefficients
         void SetDegree(int N);                 set the degree of the B-spline (default 3) [2]
         void SetBorderModeToClamp();           use clamped boundary condition (default)
         void SetBorderModeToMirror();          use mirror boundary condition
         void SetBorderModeToRepeat();          use wrap-around boundary condition
         void Update();                         compute the coefficients from the input
       double Evaluate(const double p[3]);      evaluate spline at (x,y,z) and return value [3]
}
```

[1] Use `SetInputConnection()/GetOutputPort()` to connect this filter to a pipeline.

[2] The degree should be between 2 and 9.

[3] Use `Evaluate(const double p[3], double *val)` to interpolate multi-component images.

Appendix B: Selected vtkImageReslice interface methods

```
class vtkImageReslice : public vtkThreadedImageAlgorithm
{
    public:
         void SetInput(vtkDataObject* input);   set input image or B-spline coefficients [1]
vtkImageData* GetOutput();                      get the resampled image
         void SetInterpolationModeToLinear();   use linear interpolation
         void SetInterpolationModeToCubic();    use cubic interpolation
         void SetInterpolationModeToBSpline();  use cubic B-spline interpolation
         void BorderOn();                       use clamped boundary condition (default) [2]
         void MirrorOn();                       use mirror boundary condition
         void WrapOn();                         use wrap-around boundary condition
         void SetResliceTransform(              set transform to apply while interpolating
                vtkAbstractTransform* t);
         void SetOutputOrigin(double o[3]);     set corner of output sample grid
         void SetOutputSpacing(double s[3]);    set spacing of output sample grid
         void SetOutputExtent(int e[6]);        set index range of output sample grid [3]
         void Update();                         resample the input to the output grid
}
```

[1] `SetInputConnection()/GetOutputPort()` should be used to connect filter to pipeline.

[2] `BorderOff()` stops interpolation at input image bounds, and removes the half-voxel border.

[3] The extent consists of six numbers: the low index and high index for x, y, and z. For example, use extent [0, 255, 0, 255, 0, 119] for a 256×256×120 image.

Distributed under Creative Commons Attribution License

Appendix C: Selected vtkBSplineTransform interface methods

```
class vtkBSplineTransform : public vtkWarpTransform
{
public:
     void SetCoefficients(vtkImageData* input);          set the B-spline coefficients
     void SetDisplacementScale(double s);                scale the deformation vectors
     void SetBorderModeToEdge();                         clamped boundary condition (default)
     void SetBorderModeToZero();                         displacement falls to zero past border
     void SetBorderModeToZeroAtBorder();                 displacement falls to zero at border [1]
     void Inverse();                                     set transform inversion flag [2]
     void TransformPoint(const double in[3],             transform (x,y,z) to (x',y',z')
                         double out[3]);
     void TransformPoints(vtkPoints* in,                 transform a list of points
                          vtkPoints* out);
     void TransformVectorAtPoint(const double p[3],      apply transformation to a vector [3]
            const double in[3], double out[3]);
     void TransformNormalAtPoint(const double p[3],      apply transformation to a normal
            const double in[3], double out[3]);
     void InternalTransformDerivative(                   transform a point and get the Jacobian
            const double in[3], double out[3],           matrix (advanced method) [4]
            double J[3][3]);
}
```

[1] The border is defined to be one grid spacing beyond the edge of the coefficient grid.

[2] `GetInverse()` will return a shallow copy of the transform with the inversion flag set.

[3] This refers to a contravariant, or tangent, vector.

[4] The Jacobian can be used to transform tensors and covariant vectors.

The InternalTransformDerivative() method returns the Jacobian matrix of the partial derivatives of the output coordinates with respect to the input coordinates. For a nonlinear transformation this Jacobian is not stationary, i.e. it is a function of (x,y,z). For any cubic B-spline it varies smoothly with position, since a cubic B-spline is $C(2)$ continuous.

$$J(x,y,z) = \begin{pmatrix} \partial x'/\partial x & \partial x'/\partial y & \partial x'/\partial z \\ \partial y'/\partial x & \partial y'/\partial y & \partial y'/\partial z \\ \partial z'/\partial x & \partial z'/\partial y & \partial z'/\partial z \end{pmatrix} \tag{1}$$

Contravariant vectors are transformed by the following equation:

$$\begin{pmatrix} v'_x \\ v'_y \\ v'_z \end{pmatrix} = J \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{2}$$

while covariant vectors (e.g. gradients and normals) are transformed by the following equation:

$$\begin{pmatrix} v'_x & v'_y & v'_z \end{pmatrix} = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix} J^{-1} \tag{3}$$

Distributed under Creative Commons Attribution License

References

[1] M. Unser, M. Aldroubi, and M. Eden. B-spline signal processing: Part I — Theory. *IEEE Trans. Signal Process.*, 41(2):821–833, 1993. 1

[2] M. Unser, M. Aldroubi, and M. Eden. B-spline signal processing: Part II — Efficient design and applications. *IEEE Trans. Signal Process.*, 41(2):834–848, 1993. 1, 2.1

[3] P. Thévenaz, T. Blu, and M. Unser. Interpolation revisited. *IEEE Trans. Med. Imag.*, 19(7):739–758, 2000. 1, 2.1, 2.2

[4] D. Ruijters and P. Thévenaz. GPU prefilter for accurate cubic B-spline interpolation. *The Computer Journal*, 6 pages, 2010. doi: 10.1093/comjnl/bxq086. 1, 2.2

[5] D. Rueckert, L.I. Sonoda, C. Hayes, D.L.G. Hill, M.O. Leach, and D.J. Hawkes. Nonrigid registration using free-form deformations. *IEEE Trans. Med. Imag.*, 18(8):712–721, 1999. 1

[6] S. Klein, M. Staring, and J.P.W. Pluim. Evaluation of optimization methods for nonrigid image registration using mutual information and B-splines. *IEEE Trans. Image Process.*, 16(12):2879–2890, 2007. 1

[7] D.G. Gobbi and T.M. Peters. Generalized 3D nonlinear transformations for medical imaging: An object-oriented implementation in VTK. *Comput. Med. Imaging Graphics*, 27:255–265, 2003. 1, 2.4, 3.1

[8] K.W. Finnis, Y.P. Starreveld, A.G. Parrent, A.F. Sadikot, and T.M. Peters. A three-dimensional atlas of subcortical electrophysiology for the planning and guidance of functional neurosurgery. *IEEE Trans. Med. Imag.*, 21:93–104, 2003. 1

Distributed under Creative Commons Attribution License