N-D C^k B-Spline Scattered Data Approximation

Nicholas J. Tustison and James C. Gee

September 25, 2006

Penn Image Computing and Science Laboratory
University of Pennsylvania

Abstract

The ability to reconstruct multivariate approximating or interpolating functions from sampled data finds many practical applications in medical image analysis. Parameterized reconstruction methods employing B-splines have typically utilized least-squares methodology for data fitting. For large sample sets, solving the resulting linear system is computationally demanding as well as susceptible to ill-conditioning. We present a generalization of a previously proposed fast surface fitting technique for cubic B-splines which avoids the pitfalls of the conventional fitting approach. Our proposed generalization consists of expanding the algorithm to n dimensions, allowing for arbitrary spline degree in the different parametric dimensions, permitting wrapping of the parametric domain, and the assignment of confidence values to the data points for more precise control over the fitting results.

Keywords: *b-splines, data approximation*

1 Introduction

Given a set of uniformly or nonuniformly distributed data samples, the process of scattered data approximation constructs a smooth approximation to the scattered data. This provides a smooth representation of the scattered data for further analysis. It also allows for the inference of function values at points which are not part of the original scattered data set.

Due to its general applicability as well as ease of use, a popular technique for approximation of scattered data using B-splines is based on the minimization of the sum of the squared residuals between the individual data points and the corresponding function values (commonly referred to as *least squares fitting*). Such techniques generally involve the construction of a linear system from a set of data points. Standard matrix transformations are then used to solve for the values of the governing control points. Unfortunately, such techniques require the inversion of potentially large matrices which is not only computationally demanding but susceptible to memory problems. In addition, locally insufficient data point placement can lead to ill-conditioned matrices producing undesirable results.

Lee et al. proposed a uniform B-spline approximation algorithm in [5] which circumvents the problematic issues associated with conventional least squares fitting for 2-D cubic B-spline surfaces. The authors also introduce this algorithm within the context of a multilevel framework for hierarchical surface fitting. While they discuss the possibility of extending their algorithm to multi-dimensions, both the details of the implementation of their algorithm as well as the applications are restricted to cubic B-spline surfaces (bivariate functions). In this paper, we generalize the algorithm for more encompassing applicability and describe the ITK implementation of our algorithm first given in [10]. Our proposed generalization accommodates multivariate B-spline objects of arbitrary degree where the degree is allowed to vary in each parametric direction. We also allow for confidence values to be associated with each data point for refined control of the fitting results. In addition, we allow for a "wrapping" of the parametric domain in user-specified dimensions. We also describe some modifications to generalize the original B-spline kernel function class for creating B-spline objects with spline degrees > 3.

2 B-Spline Scattered Data Approximation

2.1 B-spline Basics

B-splines, an acronym for basis splines,[1] have become the de facto standard for computational geometric representation. This is, in large part, due to the discovery of a stable, recursive algorithm for calculating the B-splines known as the *Cox-de Boor recurrence relation* [1, 2]. Two important properties of B-splines are the following:

- *Locality:* Each point of the piecewise B-spline curve of degree d is influenced by the neighboring $d+1$ control points. Similarly, each point of an n-dimensional B-spline object, is influenced by the corresponding n-dimensional grid consisting of $\prod_{j=1}^{n}(d_j+1)$ control points where d_j is the degree of spline in the j^{th} dimension.
- *Continuity*: A B-spline curve of degree has continuity C^{d-k} at the connecting point of the polynomial segments (called *knots*). It is smooth elsewhere.

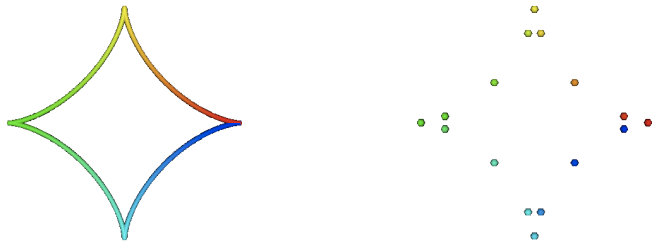
B-spline objects are defined by a set of B-splines and a grid of control points. In reconstructing the object, the B-splines serve as a weighted averaging function for the set of control points. As an example, the B-spline surface is a bivariate function given by

$$\mathbf{S}(u,v) = \sum_{i=1}^{M}\sum_{j=1}^{N}\mathbf{P}_{i,j}B_{i,d}(u)B_{j,e}(v) \tag{1}$$

which can be generalized to formulate a B-spline object of n-dimensions

$$\mathbf{S}(u_1,\ldots,u_n) = \sum_{i_1=1}^{M_1}\cdots\sum_{i_n=1}^{M_n}\mathbf{P}_{i_1,\ldots,i_n}\prod_{j=1}^{n}B_{i_j,d_j}(u_j). \tag{2}$$

[1] Contrary to popular usage in the CAGD community, we follow de Boor's distinction discussed in [3] of using the term 'B-spline' to denote the shape function of limited support and not the B-spline object (e.g. curve, surface, volume).



(a) (b)

Figure 1: (a) The original hypocycloid curve. (b) The 16 sample points of the hypocycloid.

2.2 Least Squares Fitting

Given a set of sampled data consisting of M points and a set of user-selected parametric values for each point, one can construct the linear system $\bar{\mathbf{S}} = \mathbf{B}\bar{\mathbf{P}}$ where $\bar{\mathbf{S}}$ is a column vector composed of the M data points, $\mathbf{B}$ is the sparse matrix of size $M \times N$ consisting of the B-spline values at the user-specified parametric values (commonly called the *observation matrix*), and the column vector $\bar{\mathbf{P}}$ which represents the unknown N control point values.

For least squares fitting, one must ensure that the Schoenberg-Whitney conditions (discussed in [6]) are satisfied. These conditions concern the placement of the data within the parametric domain. Violation leads to instability in the system. This is illustrated with the following example. Suppose we wish to fit a a B-spline curve to the hypocycloid curve shown in Figure 1(a) using the 16 sample points of the curve shown in Figure 1(b). This curve can be described parametrically by the following set of equations:

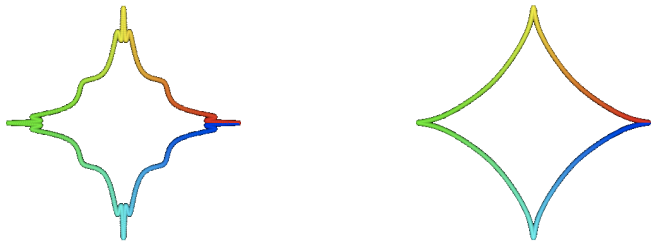
$$x = 3\cos(\phi) - \cos(0.75\phi) \tag{3}$$

$$y = 3\sin(\phi) - \sin(0.75\phi) \tag{4}$$

where $0 \leq \phi < 2\pi$. The reconstructed curve using least squares fitting is shown in Figure 2(a) whereas fitting using our method is shown in Figure 2(b). Note that the least squares fitting scenario involves an underdetermined system which violates the previously specified conditions. Fortunately, the method we propose does not suffer from these inherent difficulties.

3 Generalized B-Spline Fitting

Given the limitations associated with least squares fitting using B-splines, we present our generalized n-D B-spline fitting algorithm which is a significant extension of the surface fitting algorithm presented in [5].



(a) (b)

Figure 2: Reconstruction of the hypocycloid (Figure 1(a)) using a periodic cubic B-spline curve of 32 control points. Visual comparison of the resulting solutions using (a) least squares and (b) our proposed method demonstrates the advantages of our method.

3.1 Extension to Multivariate B-Spline Objects with Confidence Values

For a single isolated data point one can solve for the values of the surrounding subset of control points such that the B-spline object interpolates that point. Note that only a subset of control points affects the single data point due to the locality property of B-splines. Since such a system is underdetermined, the constraints inherent in the pseudoinverse produce the solution which minimizes the sum of the squares of the control point values. The solution for the multivariate case (Equation (2)) is acquired from

$$\mathbf{P}_{i_1,\ldots,i_n} = \frac{\mathbf{S}_c \left(\prod_{j=1}^{n} B_{i_j,d_j}(u_j^c) \right)}{\sum_{k_1=1}^{d_1+1} \cdots \sum_{k_n=1}^{d_n+1} \prod_{j=1}^{n} B_{k_j,d_j}^2(u_j^c)} \tag{5}$$

where $(u_1^c, \ldots, u_n^c)$ are the assigned parametric values for the c^{th} data point $\mathbf{S}_c$. The denominator is calculated from the subset of control points which have non-zero B-spline values for the point $\mathbf{S}$.

With irregularly placed data, it is likely that multiple data points will correspond to overlapping control points. In contrast to isolated data points, such a scenario requires finding a solution which provides an approximation to this subset of data points. Specifically, suppose that a single control point is influenced by M neighboring data points. Lee et al. chose a criterion which minimizes the sum of the error for each of the M data points assuming that each data point determines the control point values separately. We generalize this criteria and add a confidence term, δ, for each of the data points such that the new criteria is

$$e(\mathbf{P}_{i_1,\ldots,i_n}) = \sum_{m=1}^{M} \delta_m \left(\mathbf{P}_{i_1,\ldots,i_n} \prod_{j=1}^{n} B_{i_j,d_j}(u_j^m) - \mathbf{P}_m \prod_{j=1}^{n} B_{i_j,d_j}(u_j^m) \right)^2 \tag{6}$$

Minimization of this criterion leads to the solution for $\mathbf{P}_{i_1,\ldots,i_n}$,

$$\mathbf{P}_{i_1,\ldots,i_n} = \frac{\sum_{m=1}^{M} \delta_m \mathbf{P}_m \prod_{j=1}^{n} B_{i_j,d_j}^2(u_j^m)}{\sum_{m=1}^{M} \delta_m \prod_{j=1}^{n} B_{i_j,d_j}^2(u_j^m)} \tag{7}$$

3.2 Extension to Arbitrary Order

The original formulation in [5] restricted discussion to cubic B-splines. Such a restriction is a natural choice since it can be shown that a cubic curve which interpolates a set of data points also minimizes its curvature [9]. Also, many algorithms restrict themselves to cubic splines (e.g. [11]). However, some algorithms, such as [10], demonstrate a preference for non-cubic B-splines. Although, for our implementation, the default spline is cubic, we allow for B-spline polynomials of arbitrary degree.

3.3 Wrapping of the Parametric Domain

Since B-splines are often used to construct closed curves and other closed objects, e.g. cylindrical and toroidal structures, we accommodate these structures in our algorithm. B-splines of this type are often referred to as *periodic*. In constructing the periodic B-spline object, one simply constrains the first d control points in a given dimension to be identical to the last d control points. Algorithmically, this adds some simple bookkeeping requirements for the algorithm.

3.4 Multilevel Fitting for n-D C^k B-Spline Objects

For greater accuracy, a multilevel approach was proposed in the original algorithm. We extend this approach to arbitrary dimension and arbitrary spline degree. Due to computational efficiency and its ubiquity in other multiresolution algorithms, each level is characterized as having twice the grid resolution of the previous level. In ascending to the next higher level of higher resolution, the initial step requires calculating the new control point values which produce the same B-spline object with a higher control point resolution. Unfortunately, the description in [5] only provides the coefficients for calculating the higher resolution cubic B-spline surface from the lower resolution surface. Discussion in [4] describes the derivation of the coefficients for both the univariate and bivariate cubic B-spline case. We present a brief discussion of doubling the resolution for multivariate B-spline objects of arbitrary order.

At a given resolution level, the B-splines are simply scaled versions of the B-splines at the previous value such that $B'_{i,d}(x) = B_{i,d}(2x)$ where B' is the B-spline at the higher level. Thus, for each polynomial span in 1-D, finding the nodal values entails first formulating the following equality:

$$\sum_{i=1}^{d+1} \mathbf{u}_i B_{i,d}(x) = \sum_{i=1}^{d+1} \mathbf{u}'_i B_{i,d}(2x) \tag{8}$$

Grouping the monomials of the separate B-splines on both sides of the equality and setting like groups of terms equal to each other, we obtain the following linear system:

$$\begin{bmatrix} b_{1,1} & \dots & b_{1,d+1} \\ \vdots & \ddots & \vdots \\ b_{d+1,1} & \dots & b_{d+1,d+1} \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_{d+1} \end{bmatrix} = \begin{bmatrix} 2^0 b_{1,1} & \dots & 2^0 b_{1,d+1} \\ \vdots & \ddots & \vdots \\ 2^d b_{d+1,1} & \dots & 2^d b_{d+1,d+1} \end{bmatrix} \begin{bmatrix} \mathbf{P}'_1 \\ \vdots \\ \mathbf{P}'_{d+1} \end{bmatrix} \tag{9}$$

where it is assumed that the B-splines are expressed in polynomial form, i.e. $B_{i,d}(x) = b_{d+1,i}x^d + b_{d,i}x^{d-1} + \ldots + b_{1,i}$. This relatively small system is easily solved using standard matrix routines. Since each uniform B-spline is simply a parametric translation of a single representative B-spline function and since each B-spline is symmetric with respect to its maximum value, the matrix obtained from multiplying the inverse of the B-spline coefficient matrix on the left side of Equation (9) with the B-spline coefficient matrix on the right side contains consists of "translated" row pairs. Therefore, one can simply use the top two rows of coefficients of the resulting matrix product to calculate the control point values at the given level from the control point values at the previous level. Extending this formulation to the multivariate case is fairly straightforward. One simply calculates the tensor product of the relevant row of coefficients for each dimension. The coefficients for the multivariate (n-D) case are given by the elements of the n-tensor

$$\mathbf{T}_n = \mathbf{C}_1 \otimes \ldots \otimes \mathbf{C}_n \tag{10}$$

where $\mathbf{C}_i$ is one of the two rows of coefficients for the i^{th} dimension and $\otimes$ denotes the outer or tensor product between two vectors. Note that different dimensions might employ different degrees of B-splines. Similarly, transcending one level might require refinement in one dimension but not in the other. These two cases are also handled by our algorithm.

4 ITK Implementation

4.1 B-Spline Kernel Function

Currently, the B-spline kernel template class utilized within the ITK library is only capable of producing B-splines of degree less than or equal to three. However, to generalize to any degree we implemented the Cox-DeBoor recursion relation [**?**] within the `BSplineKernelFunction` class to generate the necessary polynomial pieces (using the `vnl_real_polynomial` class of the vnl library). This allows one to change the degree of the B-spline kernel function during the existence of a particular instantiation even though it is templated over a specific order. The templated aspect was kept to maintain backwards compatibility. In addition, we added the following functions:

- `double EvaluateDerivative( const double & u )`
- `MatrixType GetShapeFunctions()`
- `MatrixType GetShapeFunctionsInZeroToOneInterval()`
- `void SetSplineOrder( unsigned int order )`

The first function evaluates the derivative at the given parameter value. The second function returns a matrix where each row contains the coefficients of a single polynomial piece which, together with the coefficients in the other rows, form the basis function. The third function is similar except it returns the piecewise polynomials in the zero-to-one interval. Finally, the fourth function simply resets the kernel to be of a specified order. Shown in Figure 3 are both the B-splines of different degrees and the derivatives reproduced from the revised kernel class.

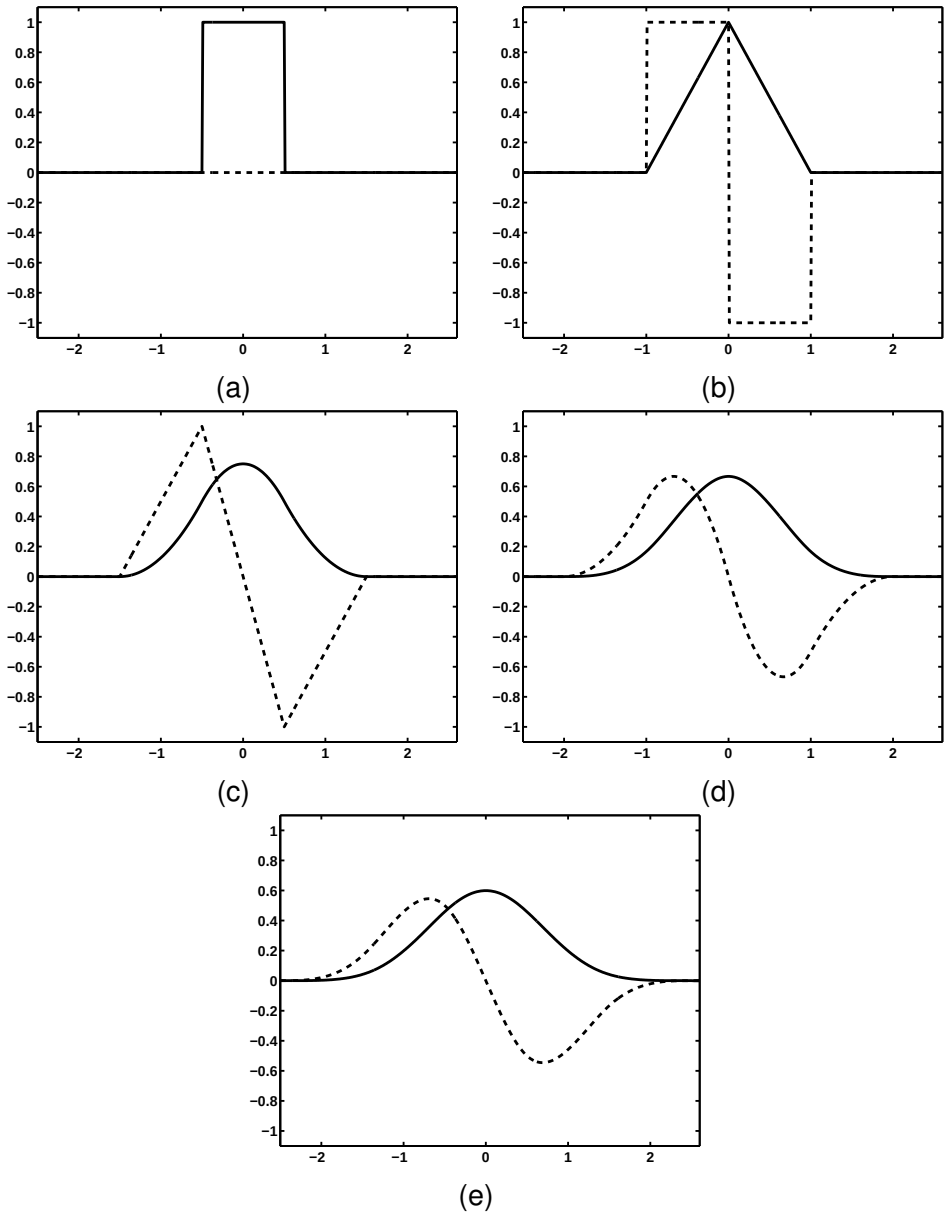


Figure 3: B-splines of different degrees (solid line) plotted with their derivatives (dashed line). Shown are B-splines of (a) degree 0, (b) degree 1, (c) degree 2, (d) degree 3, and (e) degree 4. Note that the current incarnation of the B-spline kernel function class is incapable of producing B-splines of degree > 3.

4.2 B-Spline Scattered Data Approximation Filter

Our generalized algorithm is derived from the `PointSetToImageFilter` class. The rationale for this choice will help elucidate the usage of this filter. For the generic data approximation problem with B-splines, the input typically consists of several points whereas the output is a smoothed B-spline approximation to those points. For B-spline fitting to occur, the point data must have two components: 1) its parametric location within the B-spline object and 2) its actual value. These two com-

ponents are readily available in the `PointSet` class. The type `PointSet::PixelType` is a vector (`itk::Vector#`) containing the actual value of the data whereas the `PointSetType::PointType` is the parametric location of the individual scattered data points.

B-spline objects, by definition, are defined *parametrically* on rectilinear grids. Therefore, we utilize the region defined by the the `Image` data type as the parametric domain. Each point on the parametric domain (i.e. each pixel in the image domain) of the output image is the 'smoothed' B-spline object value at that parametric location. This framework also provides a simple query mechanism for evaluating the B-spline object at any point within the continuous parametric domain in case one does not want to evaluate the B-spline object values over the entire parametric domain.

Below, we illustrate the use of our contribution by two disparate applications. The first involves approximating a smooth estimate of a scalar field defined on a 2-D image. For this example, the parametric dimension is equal to 2 whereas the data dimension is equal to 1. The second example reconstructs a sampled parametric curve using B-splines. Since this is a curve, the parametric dimension is equal to one whereas the data dimension is equal to 3.

2-D Scalar Images

Suppose we have a two-dimensional image which consists of scattered non-zero pixels on a background with pixel value equal to zero. The point set that we wish to approximate consists of those pixels with non-zero values. As we iterate through the input image, we add those points which satisfy the non-zero criteria. This is illustrated by the following bit of code.

```
17  const unsigned int ParametricDimension = 2;
18  const unsigned int DataDimension = 1;
19
20  typedef int PixelType;
21  typedef itk::Image<PixelType, ParametricDimension> InputImageType;
22  typedef float RealType;
23  typedef itk::Vector<RealType, DataDimension> VectorType;
24  typedef itk::Image<VectorType, ParametricDimension> VectorImageType;
25  typedef itk::PointSet
26    <VectorImageType::PixelType, ParametricDimension> PointSetType;
27  PointSetType::Pointer pointSet = PointSetType::New();
28
29  typedef itk::ImageFileReader<InputImageType> ReaderType;
30  ReaderType::Pointer reader = ReaderType::New();
31  reader->SetFileName( "brain.hdr" );
32  reader->Update();
33
34  itk::ImageRegionIteratorWithIndex<InputImageType>
35    It( reader->GetOutput(), reader->GetOutput()->GetLargestPossibleRegion() );
36
37  // Iterate through the input image which consists of multivalued
38  // foreground pixels (=nonzero) and background values (=zero).
39  // The foreground pixels comprise the input point set.
40
41  for ( It.GoToBegin(); !It.IsAtEnd(); ++It )
42    {
43    if ( It.Get() != itk::NumericTraits<PixelType>::Zero )
44      {
45      // We extract both the 2-D location of the point
46      // and the pixel value of that point.
47
48      PointSetType::PointType point;
49      reader->GetOutput()->TransformIndexToPhysicalPoint( It.GetIndex(), point );
```

```
50
51      unsigned long i = pointSet->GetNumberOfPoints();
52      pointSet->SetPoint( i, point );
53
54      PointSetType::PixelType V( DataDimension );
55      V[0] = static_cast<RealType>( It.Get() );
56      pointSet->SetPointData( i, V );
57      }
58    }
```

Now that we have the point set we wish to approximate with our B-spline filter, we simply instantiate our filter and modify the desired components as follows:

```
61  // Instantiate the B-spline filter and set the desired parameters.
62  typedef itk::BSplineScatteredDataPointSetToImageFilter
63    <PointSetType, VectorImageType> FilterType;
64  FilterType::Pointer filter = FilterType::New();
65  filter->SetSplineOrder( 3 );
66  FilterType::ArrayType ncps;
67  ncps.Fill( 4 );
68  filter->SetNumberOfControlPoints( ncps );
69  filter->SetNumberOfLevels( 3 );
70
71  // Define the parametric domain.
72  filter->SetOrigin( reader->GetOutput()->GetOrigin() );
73  filter->SetSpacing( reader->GetOutput()->GetSpacing() );
74  filter->SetSize( reader->GetOutput()->GetLargestPossibleRegion().GetSize() );
75
76  filter->SetInput( pointSet );
77
78  try
79    {
80    filter->Update();
81    }
82  catch (...)
83    {
84    std::cerr << "Test 1: itkBSplineScatteredDataImageFilter exception thrown"
85              << std::endl;
86    return EXIT_FAILURE;
87    }
```

The filter is defined by the B-spline order (= polynomial degree), the number of control points in each dimension (which is of type `FixedArray`), and the number of levels (≥ 1). Each of these parameters have defaults if they are not set explicitly. If the multilevel approach is used by specifying the number of levels to be greater than one, the number of control points that is specified by the user is the number of control points at the coarsest level of the multilevel scheme. The number of control points used at each subsequent level is such that the B-spline grid size doubles in resolution over the previous level.

The user must also specify the input point set as well as the origin, size, and spacing of the image region over which the B-spline approximation occurs. These parameters also define the output image.

We illustrate our implementation in Figure 4. Shown in Figure 4(a) is a 2-D image of an initial brain segmentation. We wish to fit a B-spline surface to the pixels with nonzero value. The values of these pixels are used to estimate a fourth order B-spline surface using the multilevel approach. Notice that as the grid resolution increases, the approximation approaches that of the original data.

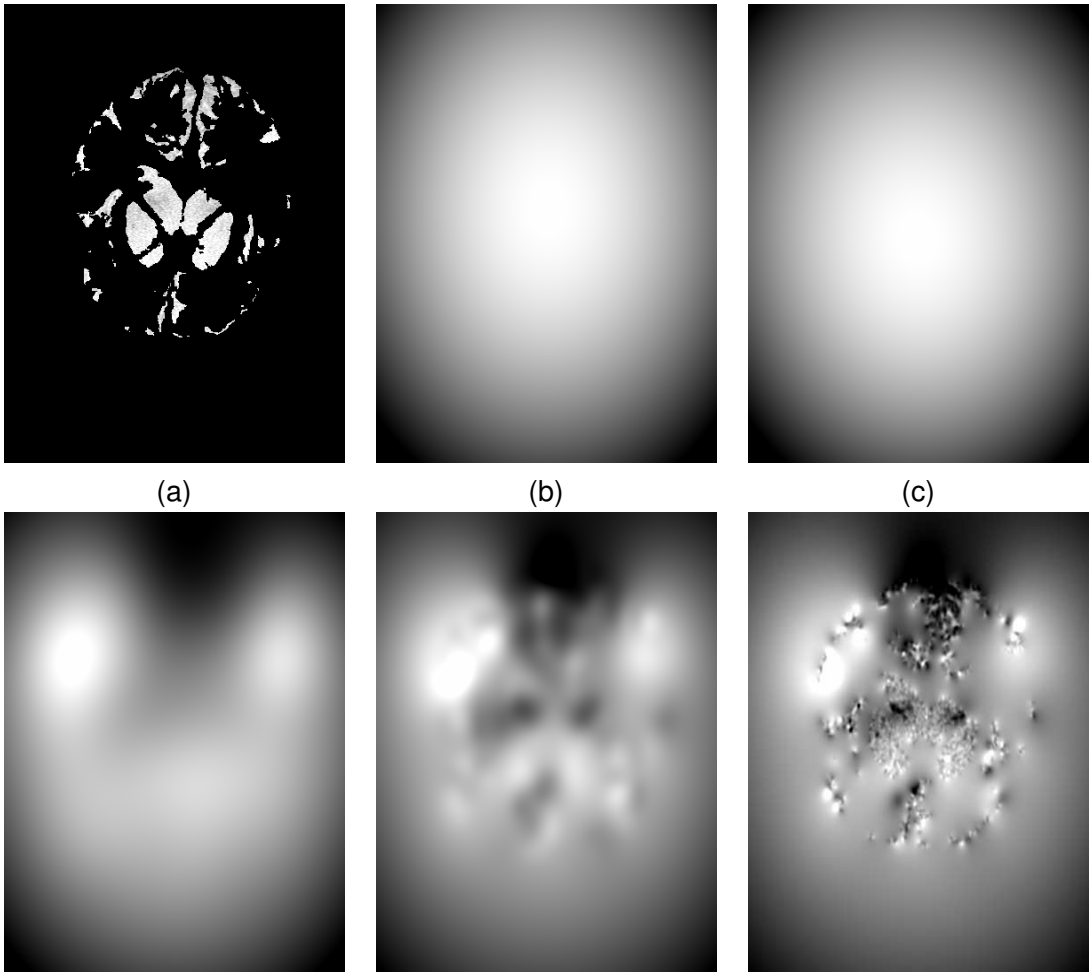


Figure 4: (a) Original segmented 2-D brain slice showing pixels labeled as gray matter. Using this image we calculate 2-D fourth order B-spline surfaces using a multilevel approach where the coarsest level is defined by 5×5 control points. The number of levels in each of the subsequent images are (b) 1 level, (c) 3 levels, (d) 5 levels, (e) 7 levels, and (f) 10 levels.

3-D Curves

If the values over the entire domain do not need to be reproduced, values only at specific points can be calculated. To illustrate this capability, we sample an arbitrary curve and fit a B-spline curve to the sampled data. The chosen curve is a helix defined as follows:

$$\begin{aligned} \mathbf{C}(u) &= (x(u), y(u), z(u)) \\ &= (0.25\cos u, 0.25\sin u, 4u), \qquad 0 \leq u \leq 1. \end{aligned} \tag{11}$$

We utilize a sampling rate of $\Delta u = 0.05$. This set of points comprises the point set input to our B-spline filter. The initialization of the point set proceeds similarly to the previous example.

```
117  const unsigned int ParametricDimension = 1;
118  const unsigned int DataDimension = 3;
119
120  typedef double RealType;
121  typedef itk::Vector<RealType, DataDimension> VectorType;
122  typedef itk::Image<VectorType, ParametricDimension> ImageType;
123
124  typedef itk::PointSet<VectorType, ParametricDimension> PointSetType;
125  PointSetType::Pointer pointSet = PointSetType::New();
126
127  // Sample the helix.
128  for ( RealType t = 0.0; t <= 1.0+1e-10; t += 0.05 )
129    {
130    unsigned long i = pointSet->GetNumberOfPoints();
131
132    PointSetType::PointType point;
133    point[0] = t;
134    pointSet->SetPoint( i, point );
135
136    VectorType V;
137    V[0] = 0.25*cos(t*6.0*3.141);
138    V[1] = 0.25*sin(t*6.0*3.141);
139    V[2] = 4.0*t;
140
141    pointSet->SetPointData( i, V );
142    }
```

Subsequent to the creation of the point set, we instantiate the filter and set the necessary parameters. However, unlike before, we disable the image output and simply choose to sample to curve at the points we select.

```
144  // Instantiate the filter and set the parameters
145  typedef itk::BSplineScatteredDataPointSetToImageFilter
146     <PointSetType, ImageType>  FilterType;
147  FilterType::Pointer filter = FilterType::New();
148
149  // Define the parametric domain
150  ImageType::SpacingType spacing;  spacing.Fill( 0.001 );
151  ImageType::SizeType size;  size.Fill( 1.0/spacing[0] );
152  ImageType::PointType origin;  origin.Fill( 0.0 );
153
154  filter->SetSize( size );
155  filter->SetOrigin( origin );
156  filter->SetSpacing( spacing );
157  filter->SetInput( pointSet );
158
159  filter->SetSplineOrder( 3 );
160  FilterType::ArrayType ncps;
161  ncps.Fill( 4 );
162  filter->SetNumberOfControlPoints( ncps );
163  filter->SetNumberOfLevels( 5 );
164  filter->SetGenerateOutputImage( false );
165
166  try
167    {
168    filter->Update();
169    }
170  catch (...)
171    {
172    std::cerr << "Test 2: itkBSplineScatteredDataImageFilter exception thrown" << std::endl;
173    return EXIT_FAILURE;
174    }
175
176  for ( RealType t = 0.0; t <= 1.0+1e-10; t += 0.01 )
```

```
    {
    PointSetType::PointType point;
    point[0] = t;

    VectorType V;
    filter->Evaluate( point, V );
  }
```

These points are then collected and used to reconstruct the approximating curve. Four reconstructed approximating curves are illustrated in Figure 5.

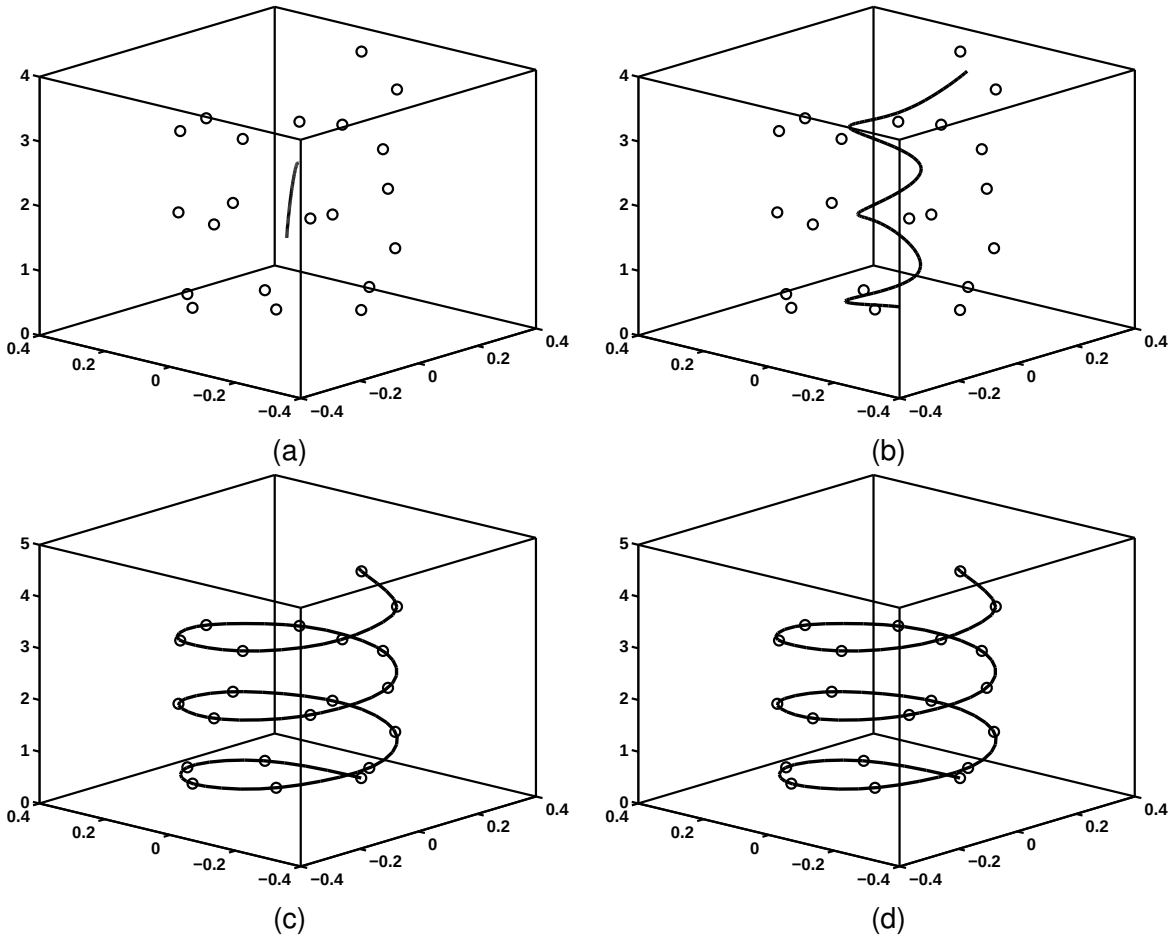


Figure 5: Approximating fourth order B-spline curves with ascending number of levels for increasing accuracy. The sampled points of the original helix are illustrated by the spheres. The approximating curves are created using (a) 1 level, (b) 5 levels, (c) 10 levels, and (d) 20 levels.

5 Discussion

The extensions to the original algorithm of Lee et al. presented in this paper generalizes the cubic bivariate B-spline surface fitting algorithm that the authors developed. These extensions include

generalization to n dimensions and arbitrary degree of B-spline. Although it wasn't shown in either of the two examples, we do allow for wrapping of the parametric domain for closed structures (e.g. Figure 1).

References

[1] M. G. Cox. The numerical evaluation of B-splines. *Jour. Inst. Math. Applic.*, 10:134–149, 1972. 2.1

[2] C. de Boor. On calculating with B-splines. *Jour. Approx. Theory*, 6:50–62, 1972. 2.1

[3] Carl de Boor. *Fundamental Developments of Computer-Aided Geometric Modeling*, chapter B-spline basics, pages 27–49. American Press, 1993. 1

[4] Oyvind Hjelle. Approximation of scattered data with multilevel B-splines. Technical report, 2001. 3.4

[5] Seungyong Lee, George Wolberg, and Sung Yong Shin. Scattered data interpolation with multilevel B-splines. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):228–244, 1997. 1, 3, 3.2, 3.4

[6] Weiyin Ma and J. P. Kruth. Parameterization of randomly measured points for least squares fitting of B-spline curves and surfaces. *Computer-Aided Design*, 27:663–675, 1995. 2.2

[7] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer, 1997.

[8] R. F. Riesenfeld. *Applications of B-Spline Approximation to Geometric Problems of Computer-Aided Design*. PhD thesis, Syracuse University, 1975.

[9] I. J. Schoenberg. Spline functions and the problem of graduation. *Proc. Nat. Acad. Sci.*, 52:947–950, 1964. 3.2

[10] Nicholas J. Tustison and Amir A. Amini. Biventricular myocardial strains via nonrigid registration of anatomical NURBS model [corrected]. *IEEE Trans Med Imaging*, 25(1):94–112, Jan 2006. 1, 3.2

[11] Nicholas J. Tustison, Victor G. Davila-Roman, and Amir A. Amini. Myocardial kinematics from tagged MRI based on a 4-D B-spline model. *IEEE Trans Biomed Eng*, 50(8):1038–1040, Aug 2003. 3.2