MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

FasTFit: A Fast T-spline Fitting Algorithm

Feng, C.; Taguchi, Y.

TR2017-141 November 2017

Abstract

T-spline has been recently developed to represent objects of arbitrary shapes using a smaller number of control points than the conventional NURBS or B-spline representations in computer aided design, computer graphics, and reverse engineering. However, existing methods for fitting a T-spline over a point cloud are slow. By shifting away from the conventional iterative fitand-refine paradigm, we present a novel split-connect-fit algorithm to more efficiently perform the T-spline fitting. Through adaptively dividing a point cloud into a set of B-spline patches, we first discover a proper topology of T-spline control points, i.e., the T-mesh. We then connect these B-spline patches into a single T-spline surface with different continuity options between neighboring patches according to the data. The T-spline control points are initialized from their correspondences in the B-spline patches, which are refined by using a conjugate gradient method. In experiments using several types of large-sized point clouds, we demonstrate that our algorithm is at least an order of magnitude faster than state-of-the-art algorithms while provides comparable or better results in terms of quality and conciseness.

Computer-Aided Design

FasTFit: A Fast T-spline Fitting Algorithm

Chen Feng*, Yuichi Taguchi

Mitsubishi Electric Research Laboratories (MERL)
201 Broadway, Cambridge, Massachusetts, U.S.

Abstract

T-spline has been recently developed to represent objects of arbitrary shapes using a smaller number of control points than the conventional NURBS or B-spline representations in computer aided design, computer graphics, and reverse engineering. However, existing methods for fitting a T-spline over a point cloud are slow. By shifting away from the conventional iterative fit-and-refine paradigm, we present a novel split-connect-fit algorithm to more efficiently perform the T-spline fitting. Through adaptively dividing a point cloud into a set of B-spline patches, we first discover a proper topology of T-spline control points, i.e., the T-mesh. We then connect these B-spline patches into a single T-spline surface with different continuity options between neighboring patches according to the data. The T-spline control points are initialized from their correspondences in the B-spline patches, which are refined by using a conjugate gradient method. In experiments using several types of large-sized point clouds, we demonstrate that our algorithm is at least an order of magnitude faster than state-of-the-art algorithms while provides comparable or better results in terms of quality and conciseness.

Keywords: T-spline, point clouds, surface fitting, Bézier patch

1. Introduction

In recent years 3D point clouds of objects or environments can be readily acquired by various sensors such as consumer-grade depth cameras (most no-

*Corresponding author

Email addresses: cfeng@merl.com (Chen Feng), taguchi@merl.com (Yuichi Taguchi)

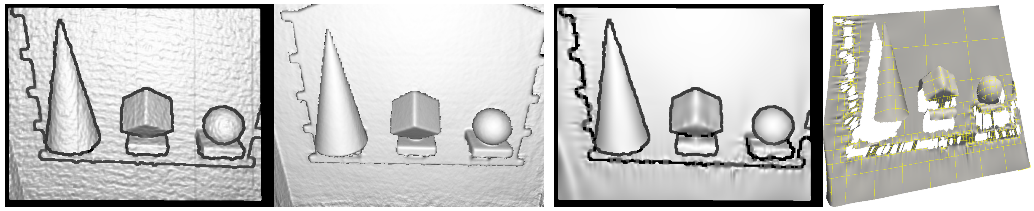


Figure 1: We present an algorithm for efficiently fitting a T-spline surface to a 2D-parameterized point cloud. From left to right: an input raw point cloud obtained with a Kinect sensor; a result obtained with KinFu by fusing multiple point clouds; our result obtained by fitting a T-spline surface to the single input point cloud; and a 3D visualization of our surface fitting result where the knot lines of the T-spline are superimposed on the mesh model. Our algorithm enhances the quality of point clouds similar to KinFu but by using only a single point cloud.

tably Kinect), stereo cameras, and LIDARs. However these large amount of point cloud data are rarely directly useful for many downstream applications, including reverse engineering, 3D modeling, and 3D printing. Usually they need to be further processed and compactly represented to be CAD-compliant parametric models. Only after this conversion can they be systematically edited and improved in those applications more easily, compared to direct operations on meshes from raw point clouds. Besides, having parametric models often enables better and smoother normal estimation on those point clouds, which in many cases is necessary for better rendering in visualization.

B-spline or NURBS surfaces have been the industry-standard CAD-compliant models [1]. By refining knot vectors and adjusting control points, they can be used to represent arbitrarily complex scenes. However since such a tensor product surface representation requires each row of the control mesh to have the same number of control points, it only allows the so-called global refinement of knot vectors. This often results in many redundant control points when representing a complex scene, which can decrease both the fitting speed and the efficiency of many subsequent operations on the model. T-spline [2, 3] and its variations [4, 5, 6] have been proposed to address such shortcoming. By allowing T-junctions in the control mesh, this type of models enables local knot refinement to avoid superfluous control points. Due to this significant improvement and its backward compatibility with the industry-standard NURBS, it has been widely accepted in the design communities and applied to different research areas from isogeometric analysis [7] to multi-view stereo [8].

Using T-spline models to fit 3D data points is non-trivial, even after data parameterization, which is itself an ongoing research topic and is out of the scope of this paper. Two problems need to be addressed essentially: (1) How to find a proper T-mesh, i.e., the topology of control points in the parametric domain of a T-spline; (2) How to find the optimal 3D positions of all control points in that T-mesh efficiently. Existing T-spline fitting methods [9, 10, 11], which will be discussed later, follow the same strategy as in the classical NURBS fitting: iteratively alternate between (1) topology refinement and (2) control point fitting. However, unlike a NURBS control mesh which is fully connected and thus allows efficient separable fitting from the two parametric direction sequentially [1], a T-spline loses such benefit due to the T-junctions in its T-mesh, and its control points have to be fitted with all data points at once. This makes the control point fitting step computationally expensive when we have large data size and large number of control points.

In this paper, we focus on how to rapidly and accurately fit a T-spline to a 2D-parameterized point cloud. During the fitting, we assume a fixed data parameterization, i.e., each data point has a fixed 2D parameter. For an organized point cloud obtained from sensors such as Kinect and stereo camera, this parameterization can be identical to its image index, and thus the T-mesh lives on its image domain. For an unorganized point cloud generated, e.g., by registering multiple measurements, this parameterization can be given by mapping 3D points onto a plane or sphere.

Our core contribution is a novel fast T-spline fitting strategy using a bottom-up approach to avoid conventional iterative fit-and-refine paradigm: Instead of finding a global T-mesh using all the data points in a top-down iterative manner as in the existing methods, we first divide a point cloud into a set of local regions and fit a simpler B-spline patch for each local region. The local B-spline patches are then connected with different continuity options according to the data and used to define the global T-mesh. The local B-spline patches are also used to initialize the control points of the T-spline surface, which are finally refined by using a conjugate gradient method. With this strategy, our algorithm, FasTFit, achieves near real-time performance on VGA-sized Kinect point clouds, which is an order of magnitude faster than existing state-of-the-art methods that we are aware of.

2. Related Work

In this section, we briefly review algorithms for fitting B-spline, NURBS, and T-spline surfaces. For all of those parametric models, an important pre-processing step has to be done for assigning 2D parameters to each data point, referred to as the data parameterization step. Traditionally, uniform, chord-length, and centripetal parameterizations have been widely applied due to their simplicity and effectiveness [1]. There exist more sophisticated parameterization methods to further improve final fitting results, such as mean value coordinates [12], a neural network based method [13], and a curvature based method [14]. There is also a method avoiding parameterization by introducing active contour model for evaluating fitting error [15].

As mentioned in the introduction, we assume that our input point clouds have been parameterized, as is assumed in the prior work [1, 9, 16]. Our strategy is not limited to specific parameterization methods, but we use simpler approaches for faster computation. Since most sophisticated parameterizations are time consuming, we believe it is reasonable to bear with slightly more control points in trade of faster computations.

2.1. B-spline and NURBS Fitting

There are mainly two strategies for fitting B-spline or NURBS surface. The first one, as mentioned above, always starts from a simple control mesh, performs global knot refinement at the area with large fitting error, solves the optimal positions of all control points, updates data parameterization if necessary, and then repeats this process until the fitting error becomes small enough [17]. The second one, which is less popular, reverses that procedure by starting from an over-complicated mesh and iteratively simplify it by knot removal [1].

Besides the two traditional strategies, there are a few other methods trying to avoid the iterative control mesh refinement. For example, the multi-level B-spline [18, 19, 20] adaptively partitions the point cloud into a quad-tree structure and fits a B-spline on the fitting residual of each quad-tree level; instead of spending time in the iterative mesh refinement, these methods can directly fit B-splines from coarse to fine levels. Another interesting method has been recently developed which applies level set to capture data topology and then sequentially fits the data into quadrilateral meshes, Catmull-Clark subdivision surfaces, and finally B-spline surfaces [21]. These B-spline fitting methods are similar to our method in a way that the time consuming mesh refinement and re-fitting are avoided for faster computation.

2.2. T-spline Fitting

As a brief review, a degree d T-spline equation is similar to the NURBS formulation, which represents each T-spline surface point $Q(u, v) \in \mathbb{R}^3$ with associated parameters u and v as a combination of all control points $C_k \in \mathbb{R}^3, k = 1, \cdots, K$, as follows

$$Q(u, v) = \frac{\sum_k w_k T(u, v; U_k, V_k) C_k}{\sum_k w_k T(u, v; U_k, V_k)}, \tag{1}$$

where U_k and V_k are the local knot vectors in $\mathbb{R}^{2d-1}$ associated with the k-th control points, $T : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the T-spline blending function taking u, v as input variables and U_k, V_k as function parameters, and w_k is the weight for each control point. We enforce $w_k = 1, \forall k$ in this work to focus on standard T-splines which are polynomial instead of rational. Details of this function can be found in [2, 3]. Details of an efficient implementation of T-spline data structure can be found in [22].

Since T-spline was developed based on B-spline and NURBS, the existing T-spline fitting algorithms follow the conventional iterative B-spline fitting strategy described above. Fitting a z-map to T-spline [11] was probably the first T-spline fitting work. It followed the same strategy as in B-spline fitting except for changing the global knot refinement to T-spline enabled local knot refinement. However as its authors mentioned, this algorithm is too time-consuming, thus not suitable for large-sized point clouds (e.g., VGA-sized Kinect point clouds). Several methods have been proposed with the similar strategy to convert a scanned triangular [23] or quadrilateral mesh [24] into a T-spline. The computation time reported in these works ranges from 1 to 15 minutes for point cloud sizes close to VGA. Recently periodic T-spline has been proposed as a variation of the original T-spline for fitting tubular surfaces [25], unfortunately no computation time reported. The above strategy also showed in a T-spline based surface skinning method [26], which firstly fits B-splines on each row of scanned data and then iteratively refine knot lines between each row. Unfortunately no computation time was reported in the paper as well. Even if this method may be efficient, due to their similarity to the separable B-spline fitting that goes over data points row by row, it would be most suitable for data that are sparse in one direction and dense in the other direction. Thus it is not suitable for Kinect-like dense point clouds, as too many redundant control points will be produced. Another recent T-spline fitting method with the conventional strategy takes data curvature into consideration when refining the T-mesh, so as to allocate more control points at feature rich areas [10]. Although again no computation time was reported for this work, it is reasonable to assume a slow speed when data size grows, since that same iterative strategy was employed.

Note that none of these discussed T-spline fitting methods are designed to enable fast computation on large-sized point clouds. On one hand this is because of the iterative nature of the adopted conventional fitting strategy. On the other hand, the fact that T-spline is not a tensor product surface further slows down the computation, since solving the least squares fitting equation now has to be done with all data and control points together. A detailed study investigated both direct Cholesky and Gauss-Seidel methods for solving such fitting equations [17]. Later a progressive method was proposed and compared with Gauss-Seidel, Conjugate Gradient (CG), and Preconditioned CG (PCG) methods, showing its speed advantages for solving such fitting equations [9]. Although this new solver is shown to be fast regardless of the number of control points, due to the use of that conventional strategy, the total fitting time reported in that paper was still very long, e.g., 3 minutes for fitting a T-spline in the RGB space over a 512×512 Lena image.

Since existing methods are either slow for certain downstream applications, or not suitable for large-sized dense point clouds, fast T-spline fitting on a Kinect-like VGA sized point cloud is indeed a non-trivial and challenging task, not to mention that the raw data we are dealing with could have much poorer quality (e.g., raw Kinect scanning point clouds) than high-quality point cloud data used in the above works.

3. FasTFit Algorithm

Algorithm 1 shows an overview of our FasTFit algorithm, consisting of the following steps:

FitBezierPatchAdaptive First, the input point cloud is adaptively divided into a set of B-spline/Bézier patches, according to a prescribed fitting error threshold. Each of these patches corresponds to a rectangular sub-domain in the input data parameter domain.

Algorithm 1 Fast T-spline Fitting

```
function FasTFit(F)
    B ← FitBezierPatchAdaptive(F)
    (U, V, T_mesh) ← InferLocalKnotVector(B, F)
    C ← SolveControlPoint(B, T_mesh, U, V, F)
    return (C, U, V)
```

InferLocalKnotVector Second, these sub-domains are composed into a T-mesh with different connection options, depending on both a prescribed final model continuity and data continuities at shared edges of neighboring patches. This determines the number of control points and allows the inference of their corresponding local knot vectors to generate a T-mesh.

SolveControlPoint After finding this fixed T-mesh, we finally solve a large sparse linear system for obtaining the optimal control points using PCG initialized with the Bézier patch fitting results.

Before explaining the details of each step, we note again that in this paper we focus on point clouds with fixed parameterization. In our implementation, for an organized point cloud (either depth images or z-map data), we use a simple uniform parameterization, since many 3D sensors' raw output can be easily organized into a set of 2D indexed 3D points $\mathcal{F} = \{p_{i,j} \in \mathbb{R}^3; i = 1, \cdots, M, j = 1, \cdots, N\}$, where the 2D indices (i, j) and $(i \pm 1, j \pm 1)$ reflect the 3D proximity relationship between corresponding points unless there are depth discontinuities. Thus our parameterization of each data point directly becomes its 2D index (i, j), i.e., $Q(i, j)$ on the fitted T-spline surface from Eq. (1) corresponds to the data point $p_{i,j}$ [27]. For an unorganized point cloud $\mathcal{F}$, we use PCA-based parameterization, i.e., $\mathcal{F} = \{(p_i; u_i, v_i), p_i \in \mathbb{R}^3, u_i \in \mathbb{R}, v_i \in \mathbb{R}; i = 1, \cdots, M\}$ where the (u_i, v_i) parameters as the input variables to Eq. (1) for each 3D point p_i are obtained by projecting p_i onto the plane spanned by the two eigenvectors corresponding to the two largest eigenvalues [16].

Note that this formulation is also applicable to partial regions in a parameterized point cloud; we can optionally use pre-segmentation of an input point cloud and apply our algorithm to each segment.

3.1. Adaptive Patch Generation

Conventional T-spline fitting algorithms always approximate the input point cloud starting from a simple B-spline or Bézier surface. Then the control mesh is locally refined at places with large fitting errors by inserting more control points into the mesh. After this mesh refinement, all input data points are used again to refine the new set of control points. This means that at each mesh refinement iteration, every single data point will be accessed for a new least squares fitting. For large-sized point clouds with many fine details, such mesh refinement and least squares fitting have to be performed many times so as to achieve a reasonable balance between a high fitting accuracy and a small number of control points. Thus it is difficult for such conventional T-spline fitting strategy to achieve fast computation.

It is interesting to note that any T-spline surface can be always converted to a set of independent B-spline or more simply Bézier surfaces, by using repeated knot insertion at each knot line until its multiplicity equals to the order of the underlying B-spline basis functions, i.e., $d + 1$. This inspires us to think in the reverse procedure: why not adaptively divide the input points into smaller patches until each of them can be well represented by a simple B-spline with fixed knot vectors, or even by a Bézier patch? Once that is done, we only need to compose all such patches together into a single T-spline with proper continuity at the shared knot lines, i.e., boundary of each patch in the parameter domain.

This leads to our adaptive patch generation described in Algorithm 2. Based on our notation, for unorganized point clouds, Domain($\mathcal{F}$)$\triangleq$ $[\min\{u_i\}, \max\{u_i\}] \times [\min\{v_i\}, \max\{v_i\}]$; for organized point clouds, Domain($\mathcal{F}$)$\triangleq$ $[1, M] \times [1, N]$. Similar to [9], we uniformly split the entire input domain into several regions by the *InitSplit* function, to avoid unnecessary initial patch fitting. In our implementation, we always start with 4×4 blocks.

The *FitBezierPatch* function takes all data points within the domain r to fit a Bézier patch b defined on that r. This is done by solving the following equation using either standard QR or Cholesky factorization [1]:

$$\mathbf{P}^{\star} = \arg\min_{\mathbf{P}} \parallel \mathbf{BP} - \mathbf{Q} \parallel_F^2 + \lambda \parallel \mathbf{SP} \parallel_F^2 . \tag{2}$$

Here each row of $\mathbf{P}$ represent a Bézier control point. Input data points within region r are stored in each row of $\mathbf{Q}$, and the (i,j) entry of the $\mathbf{B}$ matrix stores the j-th control point's Bézier basis function value evaluated at the parameter of the i-th data point in r. Note that unlike fitting B-spline, there is no need

Algorithm 2 Generate Bézier Patches Adaptively

function FitBezierPatchAdaptive($\mathcal{F}$)
  $B \leftarrow \varnothing$
  $R \leftarrow$ InitSplit(Domain($\mathcal{F}$))
  for each $r \triangleq [u_{min}, u_{max}] \times [v_{min}, v_{max}] \in R$ **do**
    $R \leftarrow R \setminus r$
    $b \leftarrow$ FitBezierPatch($r, \mathcal{F}, d$)
    if b is $\varnothing$ **then continue**
    if NeedSplit(b) and CanSplit(r) **then**
      $\{r_0, r_1\} \leftarrow$ Split(r)
      $R \leftarrow R \cup \{r_0, r_1\}$
    else
      $B \leftarrow B \cup \{b\}$
  return B

to determine knot vectors for Bézier patches. Thus $\mathbf{B}$ only depends on the size of the region r. Sometimes the *FitBezierPatch* function cannot perform the least squares fitting due to rank deficiency of $\mathbf{B}$. This usually occurs at small regions with large detail variations, or regions with too many missing data. We either return an empty fit b and ignore the corresponding r, or add linear constraints $\mathbf{S}$ between control points with trade-off parameter λ to make the above system rank sufficient. Example constraints can be either simply forcing neighboring control points to be close to each other, or more sophisticated ones to suppress wiggling fit as explained in [17].

The *NeedSplit* function can use different criteria to determine whether or not b is a bad fit and thus needs to be further split into smaller parts. If the input data is known to have the same isotropic error everywhere, then this function can check the L^∞ fitting error with a prescribed threshold. Otherwise, for example, for Kinect data which is known to have depth-dependent errors, this function can check the fitting error with a dynamic depth-dependent threshold [28, 29].

The *CanSplit* function tests whether a given domain r can be further split. In our implementation, for unorganized point clouds, it always returns true. For organized point clouds, if the split domains r_0, r_1 could not have enough data inside for a valid Bézier fit, i.e., both $u_{max} - u_{min} + 1$ and $v_{max} - v_{min} + 1$ are smaller than $2d + 1$, then this r cannot be split. This is because after splitting, the resulting blocks will have less data points than control points,

Algorithm 3 Infer Local Knot Vectors

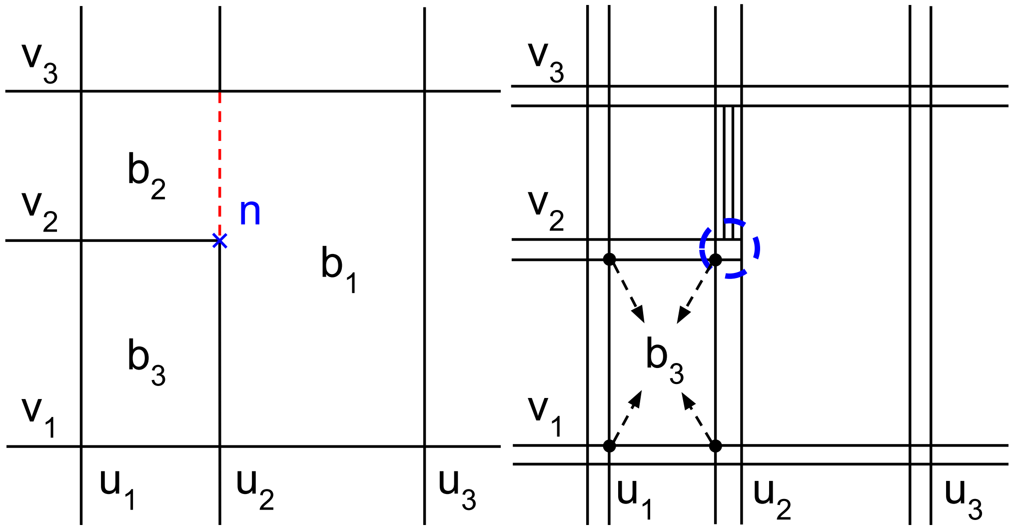
```
function InferLocalKnotVector(B, F)
    T_mesh ← BuildMesh(B)
    FindKnotMultiplicity(T_mesh, F)
    U ← ∅, V ← ∅
    for each vertex n in T_mesh do
        (U_tmp, V_tmp) ← GenLocalKnotVector(n, T_mesh)
        U ← U ∪ U_tmp, V ← V ∪ V_tmp
    return (U, V, T_mesh)
```

leading to a rank deficient system unless we perform the constrained fitting in Eq. (2). There is a case where a patch needs to be split but can not be split. This usually happens at small blocks with too significant details that cannot be represented as a simple Bézier surface. There are two options to handle this case: either performing B-spline refinement on that Bézier surface until the fitting error is small enough, or simply discarding this small part of data. In our implementation we select the latter one because those tiny details are usually caused by sensor noise and ignoring them often would not hurt the final fitting result significantly.

The *Split* function can have different behaviors: split at the patch center, or adaptively split according to fitting errors. In our implementation, we always split at the middle of the longer side of the domain to avoid thin domains which do not tend to give good fitting results.

3.2. Local Knot Vector Inference

The output B of Algorithm 2, a set of Bézier patches, is essentially a valid T-spline already, with every knot line's multiplicity equals to the chosen order, $d+1$, of all the Bézier patches. However it is more desirable to prescribe a surface parametric continuity to ensure smoothness across boundaries of patches. As previously mentioned, the boundaries of all returned patches in the parameter domain are treated as the recovered T-mesh for our T-spline fitting, which will remain fixed in the following steps. For example in Figure 2(a), a simple T-mesh is generated from the three patches b_1, b_2, and b_3. However the pre-image of the T-mesh, i.e., the T-mesh in the so-called index/parametric space [30], remains to be determined for inferring local knot vectors. This is because we can assign different multiplicity to each edge of the T-mesh, respecting both the prescribed parametric continuity



(a) Parameter Space (b) Index/Parametric Space

Figure 2: Local knot vector inference example for $d = 3$. In this example, 3 Bézier patches (b_1, b_2, b_3) are connected to generate a single T-mesh with prescribed surface continuity of C^1. Each patch boundary is classified into continuous (black solid) or discontinuous (red dashed) according to the data, which is used to determine the knot multiplicities (the number of duplicated knot lines) in the index/parametric space. The 6 junctions inside the blue dashed circle in (b) correspond to the vertex n (blue cross) in (a). The control points computed for the Bézier patches (black dots) are used to initialize those for the final T-spline.

of a desired surface model and the data continuity obtained from the input point cloud.

Thus, as described in Algorithm 3, the first step is to build a *face-edge-vertex* represented mesh [31] T_{mesh} from boundaries of all the Bézier patches, using the function *BuildMesh*. The vertices of this T_{mesh} correspond to the corners of all the patches. The edges of this T_{mesh} correspond to the sides of all the patches that do not contain any other in-between vertices. The faces of this T_{mesh} correspond to all the patches. This T_{mesh} should also allow for the operation of shooting rays from a vertex (u_2, v_2) to its up, down, left, and right directions to determine its local knot neighbors as $[\cdots, u_1, u_2, u_3, \cdots]$ in the u-direction and $[\cdots, v_1, v_2, v_3, \cdots]$ in the v-direction, as depicted in Figure 2(a), using the same rule as described in [2].

After building such T_{mesh}, the *FindKnotMultiplicity* function needs to go through each edge in the T_{mesh} to classify it into continuous or discontinuous by examining the input data continuity between the two faces/patches lying on both sides of that edge. In our implementation, we detect the discontinuity between two patches by thresholding the largest distance between their boundary points. This threshold is depth dependent for Kinect data [28, 29]. Without properly handling such discontinuities in a T-mesh, a resulting T-spline will have less representation power for fitting disconnected input points. Note that even if we use the pre-segmentation of the input point cloud, data discontinuities might remain in each segment; thus we still need to perform such detection to add knot multiplicity when inferring local knot vectors.

Finally, this algorithm goes through each vertex in the T_{mesh}, generates and stores a set of local knot vectors for control points associated with that vertex, using the *GenLocalKnotVector* function. This is done with the help of the above mentioned T_{mesh} operation of finding a vertex's local knot neighbors. For example, in Figure 2(a), the red dashed edge is marked as discontinuous since the input points are disconnected at this edge. Thus, if a C^1 continuity is prescribed for the surface model to be fitted, the *GenLocalKnotVector* function will generate a T-mesh in the index/parametric space as Figure 2(b), from its source T-mesh in the parameter space as Figure 2(a). The same operation can be used to output 6 pairs of knot vectors for the vertex n, corresponding to the 6 junctions, or the so-called anchors [30], inside the blue dashed circle.

3.3. Control Point Initialization and Refinement

As previously mentioned, in general, merely enforcing shared boundary control points in Bézier patch fitting does not result in an optimal T-spline surface in terms of either surface smoothness or fitting errors. Once a T-mesh is discovered with the set of local knot vectors $(\mathbf{U}, \mathbf{V})$ output from Algorithm 3, one has to build a sparse linear system for solving the best control point positions as

$$\mathbf{C}^\star = \arg\min_{\mathbf{C}} \parallel \mathbf{TC} - \mathbf{Q} \parallel_F^2, \tag{3}$$

where $\mathbf{T}$ is a $MN \times K$ matrix holding T-spline blending function values from Eq. (1) for each data point per row, $\mathbf{Q}$ is a $MN \times 3$ matrix holding each input data point per row of the same row order as $\mathbf{T}$, and $\mathbf{C}$ is a $K \times 3$ matrix holding each unknown control points per row of the same column order as $\mathbf{T}$ ($N = 1$ for unorganized point clouds).

If both the number of control points and the input data size are small, this sparse linear system can be solved using direct solvers such as Cholesky or QR decomposition. However when the problem size is large, it is intractable to use direct solvers, and an iterative solver such as PCG or progressive fitting as in [9] can be used. Thus, a good initialization is necessary to reduce the number of iterations for fast computation.

We adopt a heuristic, denoted as the function *AssignRelevantControlPoint* in Algorithm 4, to set each initial T-spline control point as its corresponding one in its associated Bézier patches B. Figure 2(b) illustrates one such example, where the four dots show the anchors of those T-spline control points, and they are associated with the patch b_3. After this heuristic initialization, the *RefineControlPoints* function uses an iterative solver to refine that initial guess. In our implementation, we chose PCG with the Jacobi preconditioner for such refinement.

Note that solving Eq. (3) requires the non-singularity of the basis matrix $\mathbf{T}$; i.e. the T-spline blending functions in Eq. (3) should be linearly independent. According to [30], not every T-mesh ensures the linear independence, except for a few classes such as the analysis-suitable T-mesh [32]. Similar to [9], we assume this holds for our T-mesh. Although in our extensive experiments, we never encountered singularity issues for solving the above equation, in-depth theoretical analysis will be beneficial in the future.

Also, we would like to clarify the difference between FasTFit and any control point removal process that might be used for fitting T-spline surfaces. Indeed, after all the local B-spline patches are obtained in the function *FitBezierPatchAdaptive*, those patches already form a valid, although discontinuous, T-spline surface. Yet how to properly remove control points in such a T-spline surface to enforce the prescribed surface continuity is still unclear. Moreover, applying control point removal will quickly lead us back to an iterative removal and refinement process, which we try to avoid for faster computation. Thus, we choose to firstly construct the topology of T-mesh directly without setting the corresponding geometric positions of the control points. With a valid T-mesh in the index/parametric domain, we can build the T-spline basis/blending functions for each control point, and then directly fit their optimal positions or refine from their initial positions obtained heuristically as explained above.

Algorithm 4 Solve Control Points

function $\textsc{SolveControlPoint}(B, T_{\text{mesh}}, \mathbf{U}, \mathbf{V}, \mathcal{F})$
  $\mathbf{C}_0 \leftarrow \mathbf{0}$
  for each $b \in B$ **do**
    $\textsc{AssignRelevantControlPoint}(b, \mathbf{C}_0, T_{\text{mesh}})$
  $\mathbf{C} \leftarrow \textsc{RefineControlPoints}(\mathbf{C_0}, \mathbf{U}, \mathbf{V}, \mathcal{F})$
  return C

4. Results

We evaluate our FasTFit algorithm's speed and accuracy over several simulated and real-world datasets. We consider three kinds of 3D point clouds, organized point clouds obtained with Kinect, those obtained as a z-map, as well as unorganized point clouds. For Kinect point clouds, we use simple pre-segmentation based on Euclidean distance and applied our algorithm to each segment. We found this segment-based approach leads to a smaller number of control points, thus faster computation, than fitting a single T-spline for the entire point cloud. For the other data, we fit a single T-spline for the entire data. We implement the algorithm in C++ with OpenMP parallelization and conduct all experiments on a standard desktop PC with Intel Core i7 CPU of 3.4 GHz. A video comparing the fitted T-spline surfaces in the following experiments can be found as the supplementary material of this paper.

4.1. Kinect Point Clouds

We first evaluate FasTFit over each single frame of Kinect point cloud. We performed both quantitative and qualitative comparisons between our method with two relevant state-of-the-art methods.

4.1.1. Quantitative Comparison with Conventional Strategy

Subregional knot insertion (SKI) proposed in [9] is the most recent T-spline fitting method for organized input data that follows the conventional strategy. In each mesh refinement iteration, it uniformly divides the whole input data into a number of subregions. This number increases quadratically with the iterations. Then a fixed percentage, termed insertion ratio $\alpha\%$, of the subregions with largest fitting root-mean-squared-error (RMSE) are selected. Subsequently for each selected subregion, a knot is inserted at the

Table 1: Comparisons between SKI and FasTFit for VGA-sized Kinect point cloud fitting.

Mean±Std	Time (ms)	RMSE (mm)	#ctrl
Without *FindKnotMultiplicity*			
C31	**561±98**	12.0±5.0	**1936±692**
SKI-C31CTRL	8883±4280	**10.4±4.1**	1972±706
SKI-C31RMSE	8179±3723	11.1±4.7	1954±671
With *FindKnotMultiplicity*			
C31	**732±574**	9.1±4.1	**3254±1223**
SKI-C31CTRL	20845±12325	**8.1±3.3**	3280±1232
SKI-C31RMSE	24721±19923	8.5±3.9	3514±1518
C32	**710±410**	6.8±3.0	6896±2629
SKI-C32CTRL	90855±68624	**6.1±2.8**	6897±2626
SKI-C32RMS	62709±45291	6.6±2.9	**6039±2248**

center of the T-mesh face that contains the data point with the largest fitting error inside that selected subregion. Finally, a progressive fitting algorithm, instead of PCG, is proposed in [9] to optimize control points of this new T-spline.

We implemented the SKI strategy, with $\alpha\% = 0.1$ as used in their original paper, for comparing it with our FasTFit strategy. Note that we compare the conventional vs. FasTFit fitting strategies, instead of specific sparse linear system solving algorithms. In this experiment the linear system in Eq. (3) was solved by PCG for both SKI and FasTFit.

There are three critical statistics for this comparison: fitting time, RMSE, and the number of control points. They respectively represent the speed, quality, and model conciseness of a fitting strategy. We designed six fitting configurations: *C31, SKI-C31CTRL, SKI-C31RMSE, C32, SKI-C32CTRL, SKI-C32RMSE*. The *C31* (3-degree spline with prescribed knot multiplicity of 1) and *C32* (3-degree with knot multiplicity of 2) configurations use FasTFit with prescribed surface continuity of C^2 and C^1 respectively. All other configurations apply SKI to fit C^2 surfaces using different stopping conditions for SKI's mesh refinement iteration. *SKI-C31CTRL* stops its mesh refinement once the number of control points equals or exceeds that of our *C31* fit, while *SKI-C31RMSE* stops once the current fitting RMSE becomes equal to or smaller than that of our *C31* fit. *SKI-C32CTRL* and *SKI-C32RMSE* are defined similarly with *C32* fit's number of control points and RMSE.

Table 2: Processing time for each step of VGA-sized Kinect point cloud fitting.

Mean±Std		Time (ms)
EuclideanSegmentation		16.20±0.9
FasTFit	*FitBezierPatchAdaptive*	79.10±24.7
	InferLocalKnotVector	11.75±6.7
	SolveControlPoint	470.44±87.3

We collected more than 1200 frames of VGA-sized Kinect point clouds over typical indoor scenes, and performed the six fitting experiments over these point clouds. The results are summarized in Table 1, from which one can observe the following advantages of FasTFit:

Speed As expected, FasTFit runs at least 10 to 15 times faster than SKI. The processing time for each step of FasTFit is shown in Table 2. Notice that segmentation time is not included in Table 1.

Quality When SKI is stopped at the same level of control point numbers, FasTFit results in comparable fitting RMSE (less than 0.2% difference in the RMSE of *SKI-C31CTRL* against *C31* in Table 1). Due to knot insertion, SKI cannot stop at exactly the same number of control points in FasTFit, so *SKI-C31CTRL*'s average number of control points is slightly larger than that of *C31*. Even under such condition, there are about 29% cases where *C31* have smaller RMSE than *SKI-C31CTRL*.

Conciseness When SKI is stopped at the same level of RMSE, FasTFit almost always results in smaller number of control points, (comparing the number of control points of *SKI-C31RMSE* against *C31* in Table 1). Similarly, SKI cannot stop at exactly the same RMSE in FasTFit, so *SKI-C31RMSE*'s average RMSE is slightly smaller than that of *C31*.

Also notice the performance differences in Table 1 between cases where *FindKnotMultiplicity* is performed or not during local knot vector inference. As explained above, by respecting data discontinuities through *FindKnotMultiplicity*, FasTFit can efficiently increase fitting quality with only 30% longer time (using a same threshold for *FitBezierPatchAdaptive*) while conventional strategies like SKI need 200% to 300% longer time.

In summary, FasTFit more efficiently provides surface representations with quality and conciseness that are either comparable to or better than

SKI. The advantage of FasTFit is that it can efficiently discover a suitable T-mesh without time-consuming iterative mesh refinement, and the resulting T-spline is a satisfactory surface fit, as can be seen in the following qualitative comparisons.

4.1.2. Qualitative Comparison

To evaluate the quality of surface fitting, in addition to SKI, we compare our results with those obtained by KinFu, an open-source implementation of KinectFusion [33] that provides the state-of-the-art surface reconstruction quality by fusing multiple point clouds in a truncated signed distance field (TSDF). Note that KinFu does not fit a parametric surface. Thus, it does not have a corresponding concept of fitting error as in FasTFit. Due to the large noises of Kinect sensors at long depth ranges, KinFu cannot easily fit the wall as smooth as FasTFit unless it observes the wall with higher quality point clouds (such as getting closer to the wall). Therefore, we only use KinFu for a qualitative comparison to show that our method can achieve comparable or even better surface quality using a single frame of Kinect point cloud.

Figure 3 shows the comparison, where we visualize all the results using the same shading pipeline implemented in KinFu. Our algorithm provides high-quality surface reconstruction comparable with KinFu yet using only a single frame. Note also that our representation is compact, as shown in the T-mesh visualization in Figure 3, compared to the TSDF representation with a fixed voxel resolution. Please refer to the supplementary video for comparisons over the entire sequences.

4.2. Z-map Point Clouds

We next evaluate FasTFit on z-map point clouds using a scaled version (1025×1025 pixels with inter-pixel spacing of 100m and pixel unit of 0.5m) of the well-known Puget Sound Terrain digital elevation map (DEM)[1]. Comparison with SKI method can be found in in Figure 5. Note that in this case the conventional strategy runs much slower (4 orders of magnitude) than FasTFit due to the significantly larger number of control points, thus the computation cost of finding T-mesh faces containing largest RMSE regions, shape-preserving control point insertion, and maintaining T-mesh in each iteration further slows down SKI, i.e., the conventional strategy.

[1] `http://www.cc.gatech.edu/projects/large_models/ps.html`

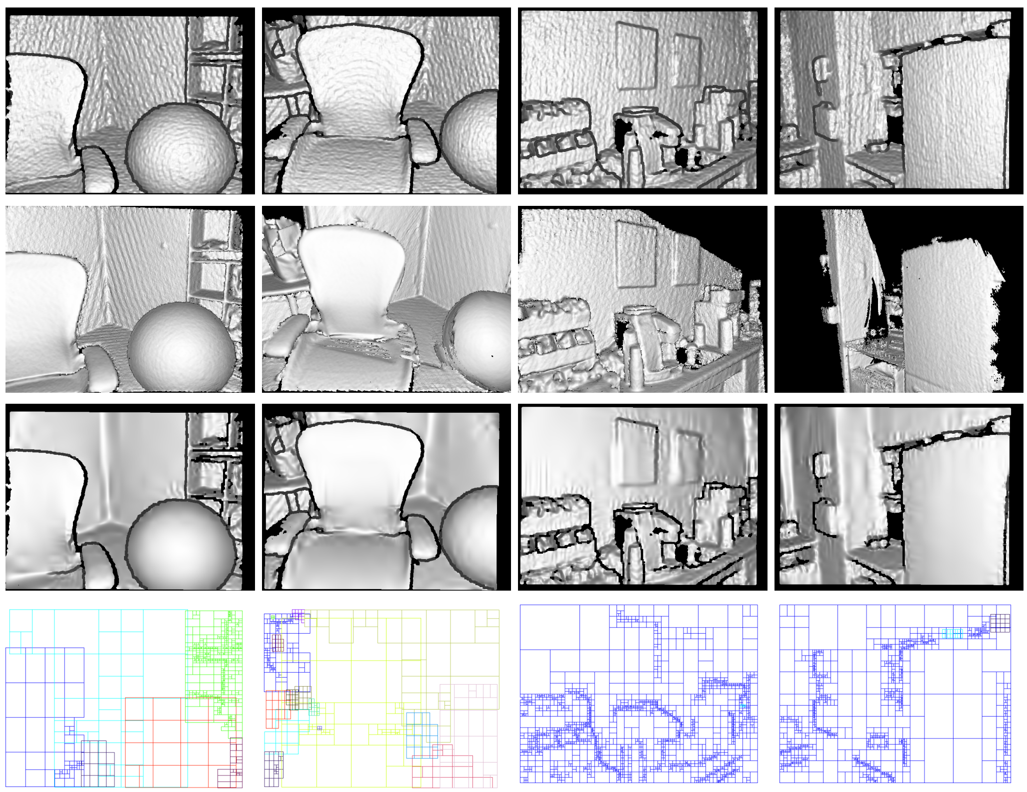


Figure 3: Qualitative comparisons between our algorithm and KinFu, an open-source implementation of KinectFusion, for several different scenes. First row: raw input point clouds. Second row: results obtained with KinFu by fusing multiple point clouds. Note that this requires accurate registration among the multiple frames, and the quality degrades if the registration is not accurate as in the second column. Third row: our results by fitting C^2 T-spline surfaces to different segments of the single input point cloud. Forth row: fitted T-meshes over point cloud segments shown in different colors. The surface reconstruction quality is comparable to that of KinFu assuming the accurate registration.

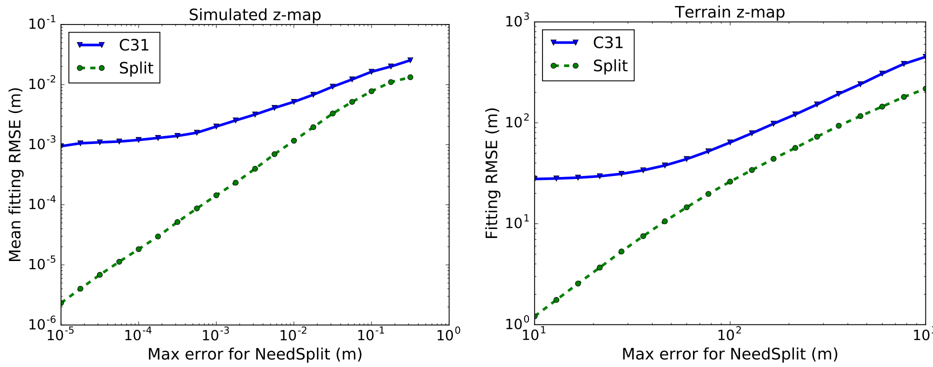
Table 3: Comparisons between SKI and FasTFit for simulated z-map fitting.

Mean±Std	Time (ms)	RMSE (mm)	#ctrl
Without *FindKnotMultiplicity*			
C31	**87±24**	8.2±3.2	**246±123**
SKI-C31CTRL	1163±444	7.2±2.9	248±123
SKI-C31RMSE	1169±450	**6.8±2.9**	269±128
With *FindKnotMultiplicity*			
C31	**158±69**	6.9±3.3	**301±176**
SKI-C31CTRL	1192±548	6.6±2.8	304±175
SKI-C31RMSE	3243±9793	**5.8±2.8**	592±927

We also simulate a dataset of 41 different z-map point clouds (300×300 points in a $1 \times 1 \times 1.2m^3$ region) generated from different random Bézier surfaces of degree ranging from 10 to 50. Based on this, we further simulate three datasets by adding missing data points, discontinuities, and isotropic Gaussian noise (0.003m standard deviation). The comparison with SKI method on these simulated datasets (164 point clouds in total) are shown in Table 3. A typical fitting result is shown in Figure 6, which demonstrates FasTFit's fitting efficiency and quality, and especially the benefit of respecting data discontinuities through *FindKnotMultiplicity*.

4.2.1. Split Criteria vs. Fitting Error

One limitation of FasTFit is that we cannot directly control a final T-spline's fitting accuracy. To decrease such a final T-spline fitting error, we need to apply a more strict criteria in the *NeedSplit* function, e.g., to decrease the prescribed threshold of the L^∞ Bézier patch fitting error. Thus we studied the relationship between the final fitting error and the prescribed error threshold using both the terrain z-map and our simulated z-map dataset mentioned above. The results are shown in Figure 4. We can observe the effectiveness of such an indirect control of the final fitting accuracy. Note that the RMSEs of Bézier patches are always lower than the final RMSEs, since Bézier patches can be seen as a T-spline without enforcing surface continuities across patch boundaries. Also, as the error threshold decreases, the RMSEs of Bézier patches approach zero while the final RMSEs saturate to small values. This is due to the noises in data that cannot be fitted into a continuous T-spline surface.



(a) Simulated z-map (b) Terrain z-map

Figure 4: The relationship between the final T-spline fitting error and the prescribed Bézier patch fitting error threshold. X-axis shows different L^∞ Bézier patch fitting error thresholds. Y-axis shows the corresponding absolute RMSE. The green dashed lines are the RMSE of the Bézier patches output by Algorithm 2 before connecting into T-splines.

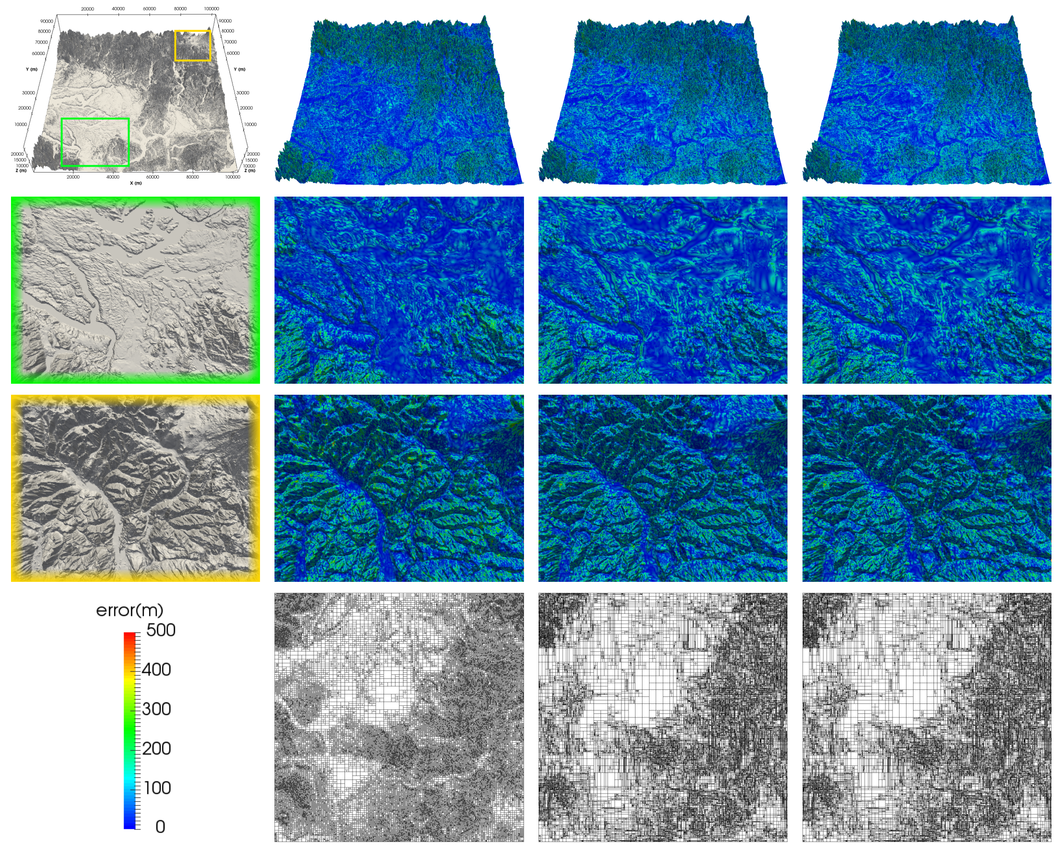
4.3. Unorganized Point Clouds

We further evaluate FasTFit on unorganized 3D point clouds captured by registering multiple down sampled Kinect point clouds using a SLAM system [34]. Here we manually segmented an object from the scanned scene, parameterized the object point cloud using a PCA-based method as described above, and fit a single T-spline surface. Some typical results are shown in Figure 7.

4.4. Image Data

In addition to the point cloud data, we fit a T-spline in RGB space for a color image, which is the main focus of [9]. The fitting result could help various image processing algorithms such as zooming and geometric transformations.

We performed such image fitting on two images used in [9]. The images reconstructed from the fitting results and corresponding T-mesh are shown in Figure 8. The differences between original images and fitted images from both FasTFit and SKI are hardly visible. Table 4 shows the fitting statistics. In the Lena case, *SKI-C31RMSE* has the same result as *SKI-C31CTRL* since it reaches the max iterations before reducing the RMSE below *SKI-C31CTRL* and *C31*. In all cases, FasTFit is significantly faster than SKI while producing similar image reconstruction quality. Note that both our FasTFit and the

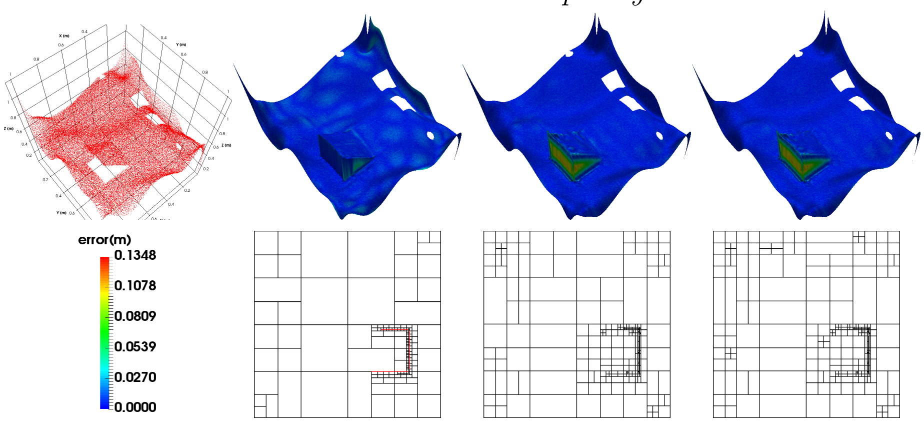


Raw mesh	*C31*	*SKI-C31CTRL*	*SKI-C31RMSE*
Time (s)	**4**	17882	17104
RMSE (m)	96.5	**94.3**	95.4
#ctrl	75807	75809	**74225**

Figure 5: Fitting a terrain z-map. Row 1: overview; Row 2 and 3: detailed views; Row 4: color legend and T-mesh. Mesh color indicates fitting error. FasTFit is 4 orders of magnitude faster and fits comparable surface comparing to the SKI method.

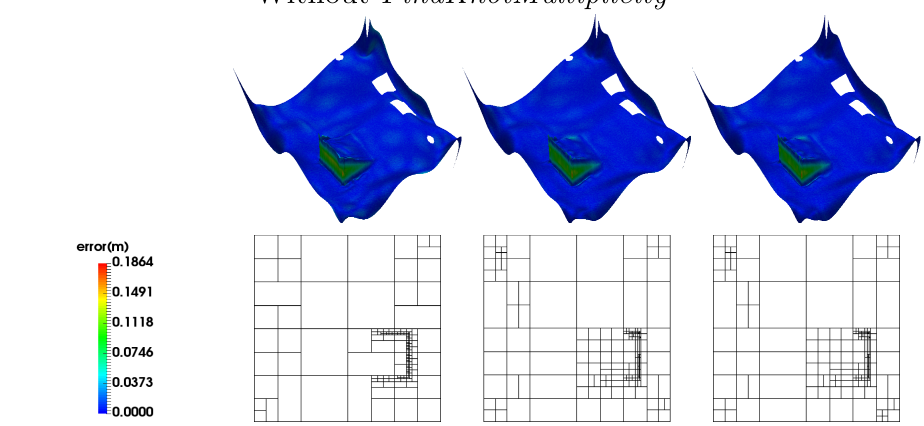
Raw mesh	*C31*	*SKI-C31CTRL*	*SKI-C31RMSE*

With *FindKnotMultiplicity*



	C31	*SKI-C31CTRL*	*SKI-C31RMSE*
Time (ms)	**146**	1532	1574
RMSE (mm)	**8.3**	8.4	**8.3**
#ctrl	**389**	393	434

Without *FindKnotMultiplicity*



	C31	*SKI-C31CTRL*	*SKI-C31RMSE*
Time (ms)	**81**	1268	1255
RMSE (mm)	10.3	**9.5**	**9.5**
#ctrl	**291**	296	302

Figure 6: Comparison on fitting a simulated z-map (generated from a 25-degree Bézier surface with Guassian noise). Mesh color indicates fitting error. The red lines in T-mesh indicate data discontinuities detected in *FindKnotMultiplicity*. FasTFit is 10 times faster and fits better surface than the SKI method.

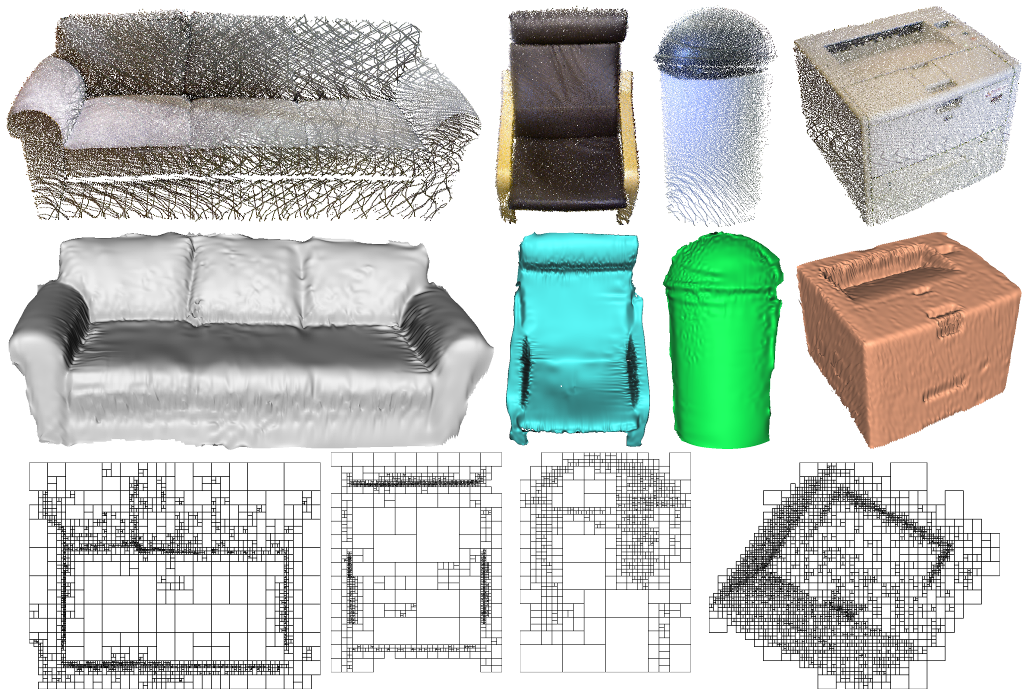


Figure 7: Fitting unorganized point clouds. From top to bottom: raw points, fitted T-spline, T-mesh.

original SKI implementation in [9] use OpenMP for parallelization. Even though FasTFit's PSNR is slightly smaller than that of SKI, and sometimes FasTFit has much more control points, it is as expected since images tend to have more data discontinuities than Kinect-like point clouds, and trade-offs have been made in FasTFit to favor computation speed.

5. Conclusions and Future Work

We presented a novel T-spline fitting strategy that can efficiently and accurately model large point clouds such as VGA-sized Kinect data. By adaptively dividing the input point cloud into smaller parts until each of them can be faithfully represented by an independent Bézier patch, a proper T-mesh is efficiently discovered without iterative knot refinement and control point adjustments. Then through different patch connection options that respect the surface continuity across patch boundaries revealed in the input data, the local knot vectors for all control points are inferred on the T-

Table 4: Comparisons between SKI and FasTFit for image fitting.

	Time (s)	PSNR	#ctrl
Lena (512 × 512)			
C31	**1.18**	**32.5276**	30533
SKI-C31CTRL	1592.09	32.2693	**23895**
SKI-C31RMSE	1629.95	32.2693	**23895**
Landscape (1600 × 1200)			
C31	**6.07**	37.2496	**43038**
SKI-C31CTRL	5388.24	36.9977	43039
SKI-C31RMSE	8799.40	**37.6410**	54145

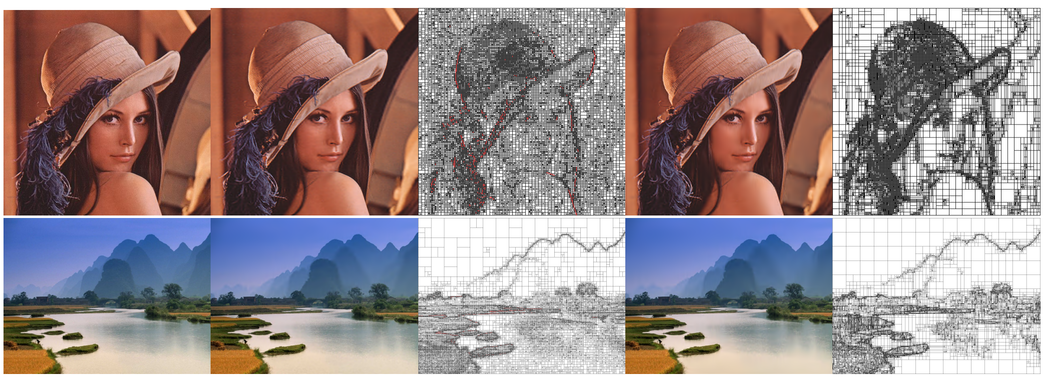


Figure 8: Comparison on fitting images. Column 1: input images; Column 2: *C31* results; Column 3: *C31* T-mesh (red lines indicates detected data discontinuities); Column 4: *SKI-C31CTRL* results; Column 5: *SKI-C31CTRL* T-mesh.

mesh. Finally through a heuristic initialization of all control points based on the fitted Bézier patches, a control point refinement is performed efficiently, leading to our fitted T-spline model of the input point cloud. Our results show comparable or sometimes even better surface reconstruction on single frame Kinect point cloud data, compared with results generated by KinectFusion after fusing multiple frames.

To the best of our knowledge, no real-time or near real-time T-spline fitting has been proposed for VGA-sized point clouds before. Our algorithm was shown to have near real-time performance on VGA-sized Kinect data with less than 600 ms processing time per frame on average. For the image data fitting, our algorithm achieved at least 2 orders of magnitude faster processing time than the published state-of-the-art result. We believe such fast processing speed could benefit many downstream applications.

While our fitting results shown in the paper and supplementary video are already visually satisfactory, there are still rooms to improve the FasTFit algorithm in the future. First is about the wiggling artifacts presented in some final fitting results under certain situations. Although they are still good fit and are mainly caused by both the structured noise in input Kinect data and locally high degree of freedom, we hope such visually less pleasing artifacts can be reduced by either adding regularization and smoothing terms in the final fitting equation, similar to [15, 35], or perform some local knot removal to reduce the degree of freedom. Second is about improving the fitting error metric so that we can avoid some redundant patch split, such as using the squared-distance-minimization (SDM) [36]. This will help further reduce the number of control points and further accelerate overall computation. Last but not least, similar to [24], we would like to incorporate sharp feature preservation to our algorithm to more faithfully represent scenes with many corners or other C^0 continuity surfaces.

Acknowledgments

We would like to thank the anonymous reviewers, Teng-Yok Lee, Srikumar Ramalingam, Jay Thornton, Alan Sullivan, Andrew Knyazev, Arvind U. Raghunathan, and Katsuyuki Kamei for their helpful comments. This work was supported by Mitsubishi Electric Research Laboratories (MERL).

References

[1] L. Piegl, W. Tiller, The NURBS Book (2Nd Ed.), Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[2] T. W. Sederberg, J. Zheng, A. Bakenov, A. Nasri, T-splines and T-NURCCs, ACM Trans. Graphics 22 (3) (2003) 477–484.

[3] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, T. Lyche, T-spline simplification and local refinement, ACM Trans. Graphics 23 (3) (2004) 276–283.

[4] Y. He, K. Wang, H. Wang, X. Gu, H. Qin, Manifold T-spline, in: Geometric Modeling and Processing, Springer, 2006, pp. 409–422.

[5] J. Deng, F. Chen, X. Li, C. Hu, W. Tong, Z. Yang, Y. Feng, Polynomial splines over hierarchical T-meshes, Graphical Models 70 (4) (2008) 76–86.

[6] H. Kang, F. Chen, J. Deng, Modified T-splines, Computer Aided Geometric Design 30 (9) (2013) 827–843.

[7] Y. Bazilevs, V. M. Calo, J. A. Cottrell, J. A. Evans, T. Hughes, S. Lipton, M. Scott, T. Sederberg, Isogeometric analysis using T-splines, Computer Methods in Applied Mechanics and Engineering 199 (5) (2010) 229–263.

[8] T. Mörwald, J. Balzer, M. Vincze, Direct optimization of T-splines based on multiview stereo, in: Int'l Conf. 3D Vision (3DV), Vol. 1, 2014, pp. 20–27.

[9] H. Lin, Z. Zhang, An efficient method for fitting large data sets using T-splines, SIAM Journal on Scientific Computing 35 (6) (2013) A3052–A3068.

[10] Y. Wang, J. Zheng, Curvature-guided adaptive T-spline surface fitting, Computer-Aided Design 45 (8) (2013) 1095–1107.

[11] J. Zheng, Y. Wang, H. S. Seah, Adaptive T-spline surface fitting to z-map models, in: Proc. Int'l Conf. on Computer Graphics and Interactive Techniques in Australasia and South East Asia, 2005, pp. 405–411.

[12] M. S. Floater, Mean value coordinates, Computer Aided Geometric Design 20 (1) (2003) 19–27.

[13] J. Barhak, A. Fischer, Parameterization and reconstruction from 3d scattered points based on neural network and PDE techniques, IEEE Transactions on Visualization and Computer Graphics 7 (1) (2001) 1–16.

[14] Z. Liu, F. Cohen, Z. Zhang, Fitting B-splines to scattered data new and old parameterization, in: Int'l Conf. on Multimedia Computing and Systems (ICMCS), 2014, pp. 75–80.

[15] H. Pottmann, S. Leopoldseder, A concept for parametric surface fitting which avoids the parametrization problem, Computer Aided Geometric Design 20 (6) (2003) 343–362.

[16] T. Mörwald, J. Balzer, M. Vincze, Modeling connected regions in arbitrary planar point clouds by robust B-spline approximation, Robotics and Autonomous Systems 76 (2015) 141–151.

[17] D. Brujic, I. Ainsworth, M. Ristic, Fast and accurate NURBS fitting for reverse engineering, The International Journal of Advanced Manufacturing Technology 54 (5-8) (2011) 691–700.

[18] S. Lee, G. Wolberg, S. Y. Shin, Scattered data interpolation with multilevel B-splines, IEEE Transactions on Visualization and Computer Graphics 3 (3) (1997) 228–244.

[19] B. F. Gregorski, B. Hamann, K. Joy, et al., Reconstruction of B-spline surfaces from scattered data points, in: Proc. Computer Graphics International, IEEE, 2000, pp. 163–170.

[20] M. Bertram, X. Tricoche, H. Hagen, Adaptive smooth scattered-data approximation for large-scale terrain visualization, in: Proc. Eurographics Symposium on Data Visualisation, 2003, pp. 177–184.

[21] H. Yoshihara, T. Yoshii, T. Shibutani, T. Maekawa, Topologically robust B-spline surface reconstruction from point clouds using level set methods and iterative geometric fitting algorithms, Computer Aided Geometric Design 29 (7) (2012) 422–434.

[22] W. Xiao, Y. Liu, R. Li, W. Wang, J. Zheng, G. Zhao, Reconsideration of t-spline data models and their exchanges using {STEP}, Computer-Aided Design 79 (2016) 36 – 47.

[23] W. C. Li, N. Ray, B. Lévy, Automatic and interactive mesh to T-spline conversion, in: Proc. the Fourth Eurographics Symposium on Geometry Processing, SGP '06, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2006, pp. 191–200.

[24] W. Wang, Y. Zhang, M. A. Scott, T. J. Hughes, Converting an unstructured quadrilateral mesh to a standard T-spline surface, Computational Mechanics 48 (4) (2011) 477–498.

[25] J. Zheng, Y. Wang, Periodic T-splines and tubular surface fitting, in: Curves and Surfaces, Springer, 2012, pp. 731–746.

[26] X. Yang, J. Zheng, Approximate T-spline surface skinning, Computer-Aided Design 44 (12) (2012) 1269–1276.

[27] H. Lin, Adaptive data fitting by the progressive-iterative approximation, Computer Aided Geometric Design 29 (7) (2012) 463–473.

[28] K. Khoshelham, S. O. Elberink, Accuracy and resolution of Kinect depth data for indoor mapping applications, Sensors 12 (2) (2012) 1437–1454.

[29] C. V. Nguyen, S. Izadi, D. Lovell, Modeling Kinect sensor noise for improved 3D reconstruction and tracking, in: Proc. Int'l Conf. 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012, pp. 524–530.

[30] A. Buffa, D. Cho, G. Sangalli, Linear independence of the T-spline blending functions associated with some particular T-meshes, Computer Methods in Applied Mechanics and Engineering 199 (23) (2010) 1437–1445.

[31] H. Lin, Y. Cai, S. Gao, Extended T-mesh and data structure for the easy computation of T-spline, J Inf Comput Sci 9 (3) (2012) 583–593.

[32] X. Li, J. Zheng, T. W. Sederberg, T. J. Hughes, M. A. Scott, On linear independence of T-spline blending functions, Computer Aided Geometric Design 29 (1) (2012) 63–76.

[33] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, A. Fitzgibbon, KinectFusion: Real-time dense surface mapping and tracking, in: Proc. Int'l Symposium on Mixed and augmented reality (ISMAR), 2011, pp. 127–136.

[34] Y. Taguchi, Y.-D. Jian, S. Ramalingam, C. Feng, Point-plane slam for hand-held 3D sensors, in: Int'l Conf. on Robotics and Automation (ICRA), IEEE, 2013, pp. 5182–5189.

[35] D. Brujic, M. Ristic, I. Ainsworth, Measurement-based modification of NURBS surfaces, Computer-Aided Design 34 (3) (2002) 173–183.

[36] W. Wang, H. Pottmann, Y. Liu, Fitting B-spline curves to point clouds by curvature-based squared distance minimization, ACM Trans. Graphics 25 (2) (2006) 214–238.