Injective hierarchical free-form deformations using THB-splines

João Pedro Duro Reis[a,*], Jiří Kosinka[b]

[a] *Universidade Federal do Rio Grande do Sul, Brazil*
[b] *University of Groningen, The Netherlands*

Abstract

The free-form deformation (FFD) method deforms geometry in n-dimensional space by employing an n-variate function to deform (parts of) the ambient space. The original method pioneered by Sederberg and Parry in 1986 uses trivariate tensor-product Bernstein polynomials in $\mathbb{R}^3$ and is controlled as a Bézier volume. We propose an extension based on truncated hierarchical B-splines (THB-splines). This offers hierarchical and local refinability, an efficient implementation due to reduced supports of THB-splines, and intuitive control point hiding during FFD interaction. Additionally, we address the issue of fold-overs by efficiently checking the injectivity of the hierarchical deformation in real-time.

Keywords: Free-form deformation, model deformation, truncated hierarchical B-spline, computer aided design, injective deformation

1. Introduction

One of the desired features of a modelling software is the capability of deforming an object in an efficient, precise, and smooth manner [3]. This can be achieved through the use of free-form deformation (FFD) techniques. The original FFD method was developed in [28]. It is based on Bernstein polynomials and intuitive control is provided through Bézier volumes. FFDs use the intuition that geometry can be deformed along with the space it is embedded in. This technique is highly flexible as it can be used globally or locally, with any degree of continuity, and even preserve volume [28].

Bernstein polynomials provide a versatile and simple basis for FFDs. However, they suffer from several limitations, most notably they have global support and fixed polynomial degree for a given number of freedoms in the corresponding control structure. This can be alleviated by employing B-splines instead [17]. Moreover, finer control is offered in the rational setting with weights attached to control points [18, 22].

FFDs have been further generalised to accommodate deformed control structures [5] and control structures of arbitrary topology [9, 24]. Additionally, special techniques have been developed to correctly deform polygonal meshes [8]. In our work, we remain in the structured setting and focus on hierarchical techniques.

Hierarchical splines were introduced in [11] to facilitate local refinement. However, the proposed method does not possess the partition of unity property in the hierarchical setting and thus finer level edits need to be maintained either independently of other levels, or via control vectors [11, 20] rather than control points. Hierarchical B-splines have been applied in the context of fitting and image registration [33, 34]. Since then, several methods which maintain partition of unity and support local refinement have been proposed. T-splines [29] allow for T-junctions in the control structure, LR-splines [6] rely on local splitting of B-splines, and THB splines [14] restore partition of unity of the hierarchical basis by a truncation mechanism.

Although T-splines have been extended, modified, and even applied in the context of FFDs, see [30, 32] and the references therein, their rational nature makes them less efficient than purely piecewise polynomial methods. While LR-splines may seem a good candidate for FFDs as they can be equipped with a control-point structure for the user to manipulate, such control structures typically lack a clear and intuitive hierarchy.

In contrast, THB-splines offer a clear hierarchical structure, which enables us to selectively hide control points from certain levels in the user interface [7, 14], and offer polynomial basis functions with reduced supports when compared to non-truncated alternatives [13]. Their locality and numerical efficiency are ideally suited for FEM-based non-rigid image registration [25, 26] and for performance-critical scenarios such as real-time FFDs performed on dense meshes. This also allows the use of more demanding techniques such as self-intersection detection and prevention [12] without sacrificing interactivity. Further advantages of B-spline hierarchies are nicely summarised in [7, 13]. An example FFD with THB-splines is shown in Figure 1.

Our main contributions are:

- a real-time FFD method based on THB-splines,
- intuitive user interface with features such as control

*Corresponding author
Email addresses: jpduroreis94@gmail.com (João Pedro Duro Reis), J.Kosinka@rug.nl (Jiří Kosinka)

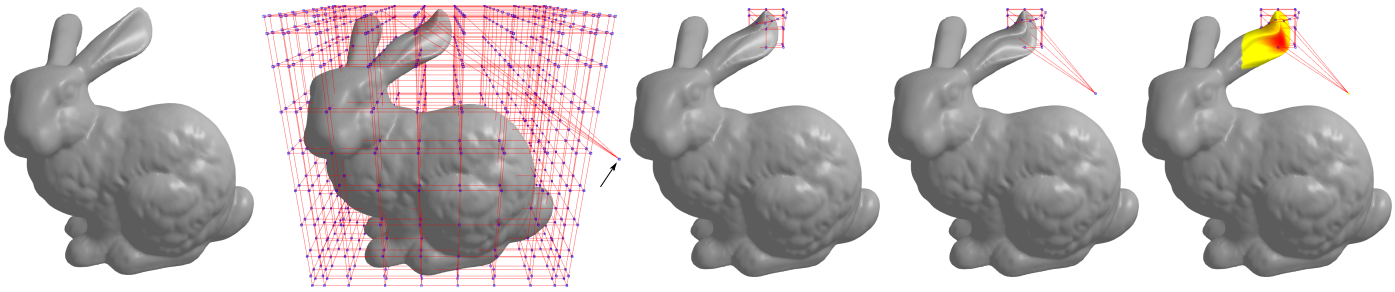

Figure 1: Far left: The input Stanford Bunny model (35K vertices, 70K faces). Left: The model embedded in a tri-cubic B-spline volume with $10 \times 10 \times 10$ control points (level zero), one of which (pointed to by the black arrow) has been moved to adjust the shape of its left ear. Middle: Control structure of level one as requested by the user near the left ear consisting of $3 \times 3 \times 3$ control points of level one. All control points of level zero have been on the user's request hidden to avoid visual clutter. Right: One of the control points of level one has been moved to finely adjust the local shape of the ear. Far right: The same situation, but this time with the region of influence of the control point being moved highlighted; yellow indicates small influence and red large influence, as determined by the associated basis function.

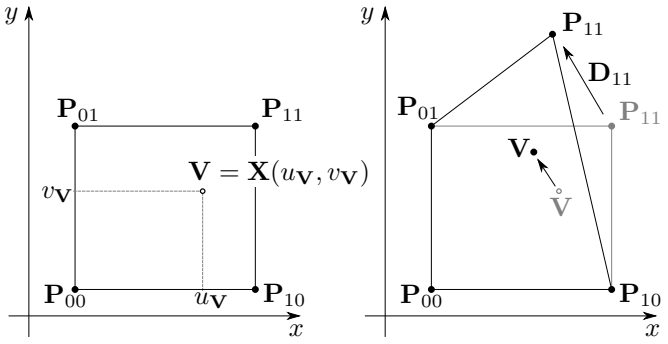

Figure 2: The FFD concept illustrated on a simple 2D example. Four control points $\mathbf{P}_{ij}$ define the control structure of a bi-linear FFD (left). A single vertex $\mathbf{V}$ with parametric coordinates $(u_\mathbf{V}, u_\mathbf{V})$ is deformed to its new position when $\mathbf{P}_{11}$ is displaced by $\mathbf{D}_{11}$ (right).

point hiding and region of influence highlighting,

- efficient injectivity checking resulting in interactive fold-over prevention.

We start by reviewing FFDs and THB-splines and show how to integrate these two frameworks (Section 2). Then we present our approach to ensuring injectivity of FFDs as well as normal updating (Section 3). Implementation details are presented in Section 4 and results are discussed in Section 5. Finally, we conclude the paper and point to future work (Section 6).

2. Free-from deformations and THB-splines

In this section we recall the basic concepts regarding FFDs and THB-splines.

2.1. Free-form deformations

FFD is a method for deforming objects by moving the control points of a control structure encapsulating (parts of) these objects. There are many approaches to specifying these structures, from regular axis-aligned grids [28] to unstructured and arbitrarily oriented meshes [24]. Once the control structure has been specified, parametric coordinates of each vertex of the deformed object(s), which we assume is a (dense) triangular mesh, are computed. This is in general a difficult problem.

In our work, we assume that the control structure forms a regular tensor-product axis-aligned grid (which can later be refined in a hierarchical manner; see Section 2.2) of control points $\mathbf{P}_{ijk}$ corresponding to a tensor-product B-spline volume of a certain tri-degree. The individual control points are geometrically positioned based on their associated Greville abscissae $\boldsymbol{\xi}_{ijk}$ [15]. In that case, the parametric coordinates $(u_\mathbf{V}, v_\mathbf{V}, w_\mathbf{V})$ of each to-be-deformed vertex $\mathbf{V}$ are easily calculated using a linear transformation [28].

Assume that the associated B-spline volume is given by

$$\mathbf{X}(u, v, w) = \sum_{ijk} \beta_{ijk}(u, v, w)\mathbf{P}_{ijk}, \tag{1}$$

where β_{ijk} are the tri-variate tensor-product B-splines defined over (typically open-uniform) knot vectors whose size is specified by the number of desired control points in each parametric direction and the spline degree. Then indeed $\mathbf{X}(u, v, w) = (u, v, w)$ when $\mathbf{P}_{ijk} = \boldsymbol{\xi}_{ijk}$ for all i, j, k and thus the parametric coordinates $(u_\mathbf{V}, v_\mathbf{V}, w_\mathbf{V})$ follow from a linear transform. Namely, let $(x_\mathbf{V}, y_\mathbf{V}, z_\mathbf{V})$ be the original coordinates of vertex $\mathbf{V}$, and let $(x_{\min}, y_{\min}, z_{\min})$ and $(x_{\max}, y_{\max}, z_{\max})$ be the minimal and maximal Cartesian coordinates in $\mathbb{R}^3$ among all $\mathbf{P}_{ijk}$, respectively. The parametric coordinates $(u_\mathbf{V}, v_\mathbf{V}, w_\mathbf{V})$ of $\mathbf{V}$ are then

$$u_\mathbf{V} = \frac{x_\mathbf{V} - x_{\min}}{x_{\max} - x_{\min}},$$
$$v_\mathbf{V} = \frac{y_\mathbf{V} - y_{\min}}{y_{\max} - y_{\min}},$$
$$w_\mathbf{V} = \frac{z_\mathbf{V} - z_{\min}}{z_{\max} - z_{\min}}.$$

Once the parametric coordinates of each vertex are known, each $\mathbf{V}$ is mapped (deformed) to $\mathbf{X}(u_\mathbf{V}, v_\mathbf{V}, w_\mathbf{V})$

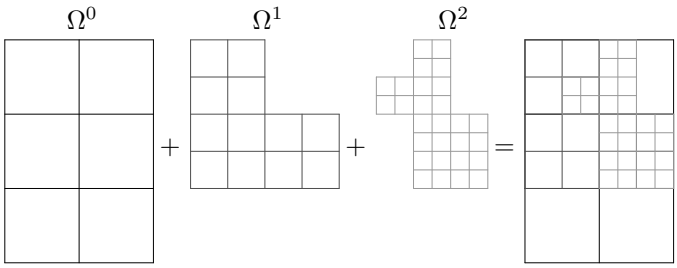

Figure 3: A locally refined domain (far right) is constructed using several domains from different levels of refinement. The initial domain Ω^0 (far left) supports tensor-product bi-linear B-splines.

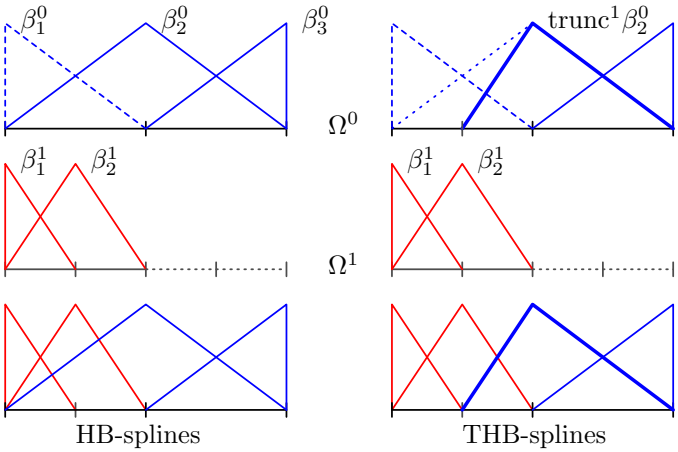

Figure 4: Hierarchical and truncated hierarchical B-splines of degree one shown over the hierarchy of two levels. Only the left half of the original domain at level 0 is selected for refinement to level 1. Top: Basis functions of level 0 on Ω^0. Replaced functions are shown dashed. The truncated function, $\text{trunc}^1\beta_2^0$, is shown in bold. Middle: Basis functions of level 1 on Ω^1. Bottom: The full hierarchical bases.

as the control points $\mathbf{P}_{ijk}$ are moved by the user. An illustration of this concept is shown in Figure 2.

While this provides a simple and efficient FFD system, local deformations are not possible due to the global tensor-product structure of the involved tri-variate B-splines. In order to support granular deformations in a hierarchical setting, we employ THB-splines.

2.2. Truncated hierarchical B-splines

Truncated hierarchical B-splines [14] provide, as discussed in Section 1, a number of advantages over other hierarchical techniques based on B-splines. We now recall some of their basic properties; further details can be found in [13, 14].

Starting from the familiar setting of tensor-product B-splines (1), hierarchical B-splines (HB-splines) first build a hierarchy of these spanning several levels based on a user-specified hierarchy of N nested domains

$$\Omega = \Omega^0 \supseteq \Omega^1 \supseteq \ldots \supseteq \Omega^{N-1} \supseteq \Omega^N = \emptyset. \tag{2}$$

The auxiliary (empty) set Ω^N is defined to simplify notation below. The level zero domain $\Omega^0 = \Omega$ is the domain of the original tri-variate tensor-product B-splines of (1). An example is shown in Figure 3.

The support of a basis function (of any level) is defined with respect to this base domain. In particular

$$\operatorname{supp}\beta = \overline{\{(u,v,w) \in \Omega : \beta(u,v,w) \neq 0\}},$$

i.e., the support of β is the closure of the set of all points in Ω where β does not vanish. Additionally, let

$$V^0 \subset V^1 \subset \ldots \subset V^{N-1} \tag{3}$$

be a sequence of nested tensor-product B-spline spaces defined on Ω. While these spaces can be defined in a more general manner, we assume that V^{l+1} is obtained from V^l by uniform dyadic knot-vector refinement.

Collecting all B-splines from all levels would lead to a linearly dependent generating system. Instead, only some basis functions are selected. At any level $l \in \{0, \ldots, N-1\}$, the set of selected functions is

$$H^l = \{\beta \in V^l : \operatorname{supp}\beta \subseteq \overline{\Omega^l} \wedge \operatorname{supp}\beta \nsubseteq \overline{\Omega^{l+1}}\}. \tag{4}$$

The hierarchical system H, forming a basis, is then obtained [21] as the union of all selected functions from all levels

$$H = \bigcup_{l=0}^{N-1} H^l.$$

A simple but illustrative example is shown in Figure 4, left, where the hierarchy comprises two levels. Note that β_1^0 of level 0 is not selected since its support is fully contained in Ω_1, and is thus replaced by β_1^1 and β_2^1 of level 1.

While H forms a basis of the hierarchical space, it does not, in general, form a partition of unity. This can be seen in the example in Figure 4. In the case of univariate linear B-splines, it is easy to see that

$$\beta_1^0 = 1 \cdot \beta_1^1 + \frac{1}{2} \cdot \beta_2^1.$$

This means that the hierarchical basis in this example does not form a partition unity as it is in excess by $\frac{1}{2} \cdot \beta_2^1$. This can be fixed with the truncation mechanism presented in [14]. We simply need to truncate β_2^0 by subtracting $\frac{1}{2} \cdot \beta_2^1$ from it. This leads to the situation shown in Figure 4, right, where partition of unity is restored.

We now formalise this concept. Further details and examples can be found in [13, 14].

As the spline spaces (3) are nested, it follows that any function β_i^l from V^l can be obtained as a linear combination of (some of the) functions β_j^{l+1} from the finer-level space V^{l+1}:

$$\beta_i^l = \sum_j c_{i,j}^{l+1} \beta_j^{l+1}. \tag{5}$$

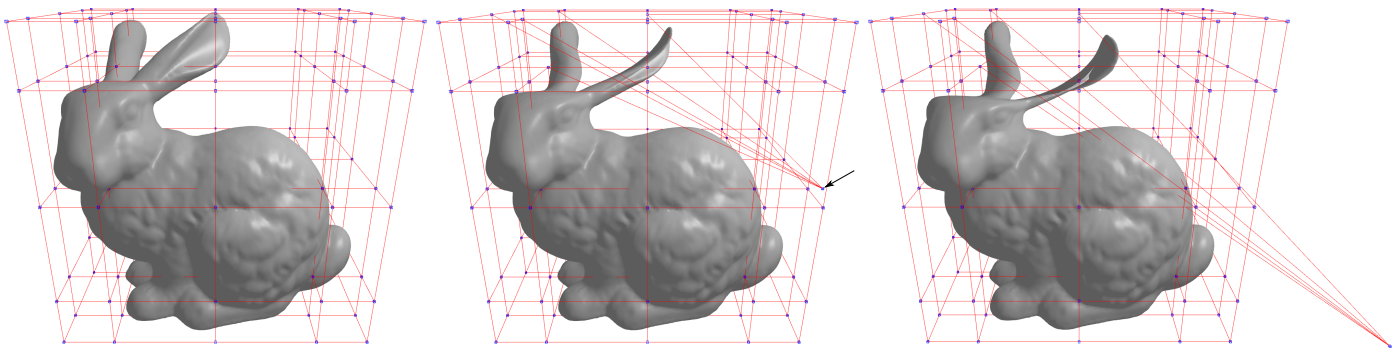

Figure 5: Left: The Stanford Bunny model in a tri-cubic $5 \times 5 \times 5$ B-spline volume. Middle: Our system ensures that user cannot introduce self-intersections in the model by moving a control point 'too far'. Right: When injectivity checking is disabled, fold-overs can occur (note the folded left ear). See the accompanying video for a real-time demonstration.

The truncation of β_i^l with respect to V^{l+1} and Ω^{l+1} is defined by

$$\mathrm{trunc}^{l+1}\beta_i^l = \sum_{j,\,\mathrm{supp}\,\beta_j^{l+1} \not\subseteq \Omega^{l+1}} c_{i,j}^{l+1}\beta_j^{l+1}. \tag{6}$$

This truncation mechanism is applied recursively to hierarchical B-splines across all levels in (4). More precisely, for every B-spline $\beta_i^l \in H^l$ its truncated version is defined as

$$\tau_i^l = \mathrm{trunc}^N(\mathrm{trunc}^{N-1}(\cdots(\mathrm{trunc}^{l+1}\beta_i^l)\cdots)) \tag{7}$$

for all $l \in \{0, \ldots, N-1\}$. These are then collected per level into T^l and the truncated hierarchical basis is then

$$T = \bigcup_{l=0}^{N-1} T^l.$$

This leads to a basis of the hierarchical space which partitions unity [14], and thus can be directly used in our context of FFDs.

As a result, (1) is generalised to

$$\mathbf{X}(u,v,w) = \sum_l \sum_{I^l} \tau_{I^l}^l(u,v,w)\mathbf{P}_{I^l}^l, \tag{8}$$

where l stands for the level in the hierarchy and I^l is a suitable index set which collects the indices of all selected (and possibly truncated) functions at level l. Recall that level $l = 0$ corresponds to the original tensor-product scenario and its domain $\Omega^0 = \Omega$ spans the full parameter space of the initial B-spline volume. Finer levels are defined via nested domains (2) and associated (refined) knot vectors; see Figure 3. In our setting, we assume that the knot vectors are open-uniform and are thus fully specified by their respective (sub-)domains via dyadic uniform refinement. This shields the user from the underlying technicalities of the method.

Additionally, the initialisation of the hierarchical grid is still as simple as in the non-hierarchical setting. This is due to the so-called coefficient preservation property [13, 14]. This effectively means that the Greville abscissae of a basis function and its truncated version are the same. Control points at finer levels are computed using (local) knot insertion to ensure that the current deformation does not change when a new level is added to the hierarchy. In our prototype application (see the accompanying video), we make use of the G+smo library [4], which offers, among other methods, an implementation of THB-splines in C++, including knot insertion and various operations on the hierarchical structure.

3. Injective FFDs with THB-splines

When manipulating an FFD (hierarchical) control grid, it is possible to introduce self-intersections in the deformed object. This is often undesirable as it does not correspond to a physical deformation. An example is shown in Figure 5. Self-intersections occur when the underlying deformation is not injective. It is known that injectivity is difficult to achieve and only few solutions that result in provably injective maps exist [10, 19, 27].

Our method enhances the possibility of deforming fine details in an object while avoiding these self-intersections. Although our method works on arbitrary (polygonal) meshes, it offers best results when applied to sufficiently dense meshes. More specifically, the mesh should be sufficiently dense to capture the main features of the object and also to ensure that the control sub-grid of the finest level in the hierarchy is not finer than the mesh. If this condition is not met in an input mesh, the user can (repeatedly) subdivide the input mesh using e.g. a linear binary subdivision step as illustrated in Figure 7. This ensures that the input geometry is exactly preserved, but the mesh density criterion is met. (Alternatively, Loop subdivision [23] can be applied, but that changes the input geometry by smoothing it.) As we shown below in Section 5.2, our system can handle very dense meshes at interactive rates and thus allows for turning coarse geometries into sufficiently dense meshes when required.

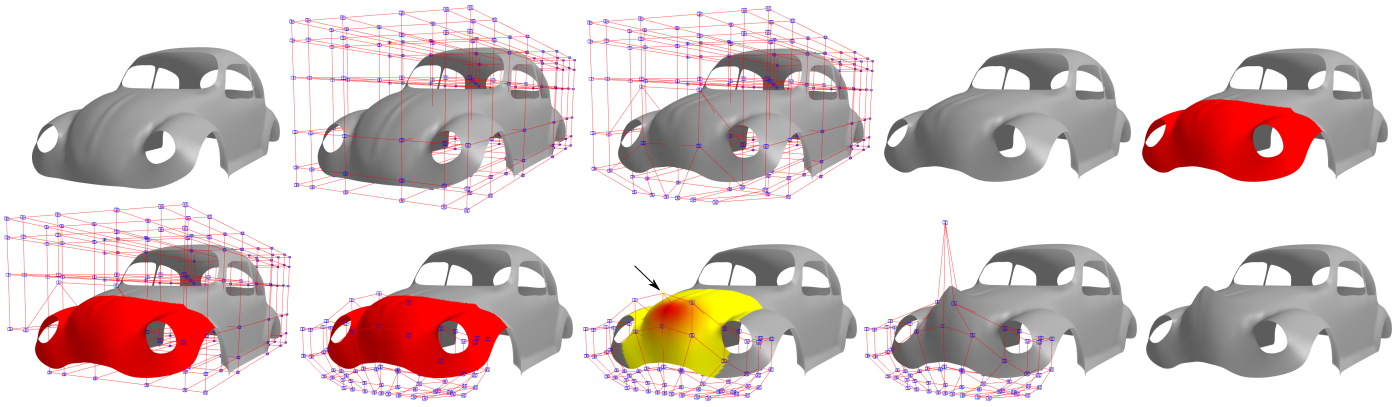

Figure 6: First row: A beetle model (far left; 61K vertices, 121K faces), embedded in a tri-cubic B-spline volume with $6 \times 6 \times 6$ control points (left), and deformed at level 0 (middle and right). When selecting the refined region for level 1, the user has the option to toggle highlighting of the to-be-affected portion of the model in red (far right). Second row: Control points of only level 0 (far left) and of only level 1 (left) are shown. A control point of level 1 is selected and its region of influence is highlighted (middle). The user performs a fine edit at level 1 (right), which results in the final deformed model (far right).

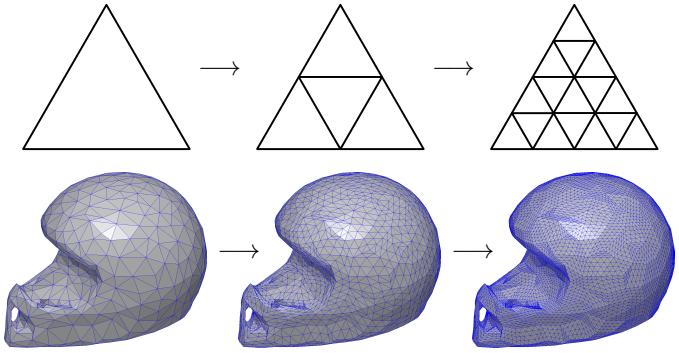

Figure 7: If an input triangular mesh is not sufficiently dense, it can be refined. Top row: A schematic representation of one and two steps of linear subdivision applied to a single triangle. Bottom row: Linear subdivision applied to a coarse model (Helmet, 496 vertices), which meets the sufficiently dense mesh criterion after refinement (with 7996 vertices after two steps).

Therefore, it is assumed that the object to be deformed is specified as a dense triangular manifold mesh with a normal at each vertex (for visualisation purposes) and that the FFD acts on the vertices and their normals, i.e., the faces of the mesh remain triangular after the deformation. While it is possible to deform triangles to curved ones [8], that comes at a greater computational expense. With the assumption that the input meshes are sufficiently dense, it is enough to deform only vertices, and, more crucially, to test injectivity of the deformation also only at the vertices. This allows us to focus on the issue of injectivity while maintaining interactive performance.

In the context of FFDs, there are two approaches [12] to checking injectivity and both rely on the Jacobian of the FFD map. One can either express the Jacobian in a spline form and leverage the convex-hull property to ensure that the Jacobian does not change sign, but this turns out to be prohibitively expensive as the Jacobian is of a relatively high degree and only sufficient conditions can be obtained. Alternatively, one can use numerical checking in combination with knot-insertion to obtain tighter bounds on the sign of the Jacobian.

In our approach, we only need to evaluate the Jacobian at the vertices of the object under deformation. Since the parametric coordinates of each vertex are known and all the Jacobian matrices are before the deformation initialised to 3×3 identity matrices, it is sufficient to only keep track of the updates of these matrices when the user moves a single control point (whose influence is typically local).

More precisely, let $J(\mathbf{X}(u,v,w))$ be the Jacobian matrix of $\mathbf{X}(u,v,w)$. Then $\mathbf{X}(u,v,w)$ is locally injective if $|J| > 0$. Note that local injectivity does not necessarily lead to global injectivity. As a counter-example, take the shape — and deform it smoothly into the shape ✂. Such deformation is locally injective everywhere, but it is not globally injective due to the self-intersection. Global injectivity can be guaranteed by also checking the injectivity of the map on the boundary of its parameter domain.

Assume that the control point $\mathbf{P}_i^l$ is moved by the user by a vector $\mathbf{D}_i^l$ acting as a displacement; see Figure 2. The displacement $\mathbf{D}_i^l$ can be seen as an increment $\Delta\mathbf{P}_i^l$ to the current position of $\mathbf{P}_i$. Turning back to (8), this update amounts to

$$\mathbf{X}(u,v,w) \leftarrow \mathbf{X}(u,v,w) + \tau_i^l \mathbf{D}_i^l, \tag{9}$$

which in turn leads to

$$J(\mathbf{X}(u,v,w)) \leftarrow J(\mathbf{X}(u,v,w)) + J(\tau_i^l \mathbf{D}_i^l). \tag{10}$$

Due to the locality of τ_i^l, we only need to update the Jacobian matrices of all vertices $\mathbf{V}$ whose parametric coordinates $(u_\mathbf{V}, v_\mathbf{V}, w_\mathbf{V})$ are contained in the support of τ_i^l.

This leads to the following update for the affected Jacobian matrices

$$J_{\mathbf{V}} \leftarrow J_{\mathbf{V}} + \nabla\tau_i^l(u,v,w)|_{(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})} \times \mathbf{D}_i^l, \qquad (11)$$

all originally initialised to identity matrices. In expanded form, the update matrix $\nabla\tau_i^l(u,v,w)|_{(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})} \times \mathbf{D}_i^l$ reads

$$\begin{bmatrix} \frac{\partial\tau_i^l(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})D_i^l x}{\partial u} & \frac{\partial\tau_i^l(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})D_i^l x}{\partial v} & \frac{\partial\tau_i^l(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})D_i^l x}{\partial w} \\ \frac{\partial\tau_i^l(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})D_i^l y}{\partial u} & \frac{\partial\tau_i^l(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})D_i^l y}{\partial v} & \frac{\partial\tau_i^l(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})D_i^l y}{\partial w} \\ \frac{\partial\tau_i^l(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})D_i^l z}{\partial u} & \frac{\partial\tau_i^l(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})D_i^l z}{\partial v} & \frac{\partial\tau_i^l(u_{\mathbf{V}},v_{\mathbf{V}},w_{\mathbf{V}})D_i^l z}{\partial w} \end{bmatrix},$$

where $\mathbf{D}_i^l = (D_i^l x, D_i^l y, D_i^l z)$ and all the partial derivatives are evaluated at $(u_{\mathbf{V}}, v_{\mathbf{V}}, w_{\mathbf{V}})$.

Of course, an analogous local update procedure is applied to the vertex positions themselves. From (9) it directly follows that

$$\mathbf{V} \leftarrow \mathbf{V} + \tau_i^l(u_{\mathbf{V}}, v_{\mathbf{V}}, w_{\mathbf{V}})\mathbf{D}_i^l. \qquad (12)$$

This allows us to achieve interactive performance.

Even when injectivity checking is disabled, the Jacobian matrices are used to correctly map the normal $\mathbf{N}$ attached to a vertex $\mathbf{V}$. Since the FFD map is in general not a rigid body transformation or uniform scaling, the 'deformed' normal is updated by

$$\mathbf{N} \leftarrow (J_{\mathbf{V}}^{-1})^{\top}\mathbf{N}, \qquad (13)$$

i.e., using the transpose of the inverse of $J_{\mathbf{V}}$, which is known in the graphics community as the normal matrix.

4. Implementation

We now describe our prototype implementation of FFDs using THB-splines, including preprocessing and real-time injectivity testing.

We have built our tool in C++ and made use of the G+smo tool-kit [4]. G+smo implements THB-splines and provides the necessary functionality to handle the underlying hierarchy of (truncated) B-splines. This includes domain refinement, creation of new domains at finer levels, and evaluating basis functions at a given parametric position.

In our prototype tool, an OBJ file holding a triangular mesh can be loaded (and saved after being deformed). We then compute its axis-aligned bounding box and construct a tensor-product grid of points based on the user-specified number of points in each direction and tri-degree. This effectively constructs a B-spline volume based on Greville abscissae corresponding to open-uniform knot vectors and constitutes level 0 of the THB-spline hierarchy as explained in Section 2.2.

As a preprocessing step, all vertices $\mathbf{V}$ of the input mesh are assigned their parametric coordinates $(u_{\mathbf{V}}, v_{\mathbf{V}}, w_{\mathbf{V}})$ with respect to the B-spline volume. These coordinates never change throughout the FFD, even if further levels of THB-splines are introduced or control points of any level are adjusted. Further, the 3×3 Jacobian matrices of all vertices are set to identity matrices. Normals (used for Phong shading) are read from the input file. (If vertex normals are not provided, they are computed, only once, by adjacent face normal averaging and stored in the input OBJ file.)

When a control point of any level is moved, we loop over vertices in the interior of the support of the corresponding basis function. The updating of these vertices and their associated Jacobians and normals is done according to (11–13). Updating not only vertices but also their normals ensures that Phong shading produces correct results also after deformation. If injectivity checking is enabled, we start looping through all updated Jacobians. As soon as a negative determinant is encountered, we break out of the loop and keep the vertex at its last position. As these update computations are data independent, they can be trivially parallelised. We handle parallel vertex, normal, and Jacobian matrix updates via OpenMP [2].

Note that in the original state, the transformation is the identity map and the determinant of the Jacobian is identically equal to one over the whole domain. This means that there is a non-empty neighbourhood of each control point in which it can be moved while maintaining injectivity. However, determining these neighbourhoods is very complex and cannot be done in real-time. If the user wishes to deform an object further in a possibly non-injective manner, injectivity checking can be disabled; see Figure 9.

As mentioned above, we set all original knot-vectors to open-uniform. This means that the user is not bothered with setting individual knots. When a new level in the hierarchy is created, the corresponding local refinement is uniform, i.e., each affected non-zero knot-interval is split in its middle. The user can select a tri-variate region for refinement by simply selecting a sub-volume based on indices for the new level, again saving the user from specifying individual knot values. To aid the user in this process, the portion of the model affected by the to-be-created domain at a finer level can optionally be highlighted in red; see Figure 6. More precisely, the portion of the model in the combined support of the basis functions that would be created by the current settings of knot indices is highlighted in red. This directly shows the user where the new level in the hierarchy has the potential to deform the model.

For the deformation itself, our tool offers region highlighting by the function value of the basis function associated with the control point (of any level) being moved; see Figures 1 and 6.

Our experimental graphical user interface offers a CAD-like deformation experience as well as simple controls. It allows the possibility to switch between the basic vertex deformation update where only vertices are updated, the normal update in which also normals are updated, and the complete update option with self-intersection checking; see Section 3. THB-spline refinement is restricted to

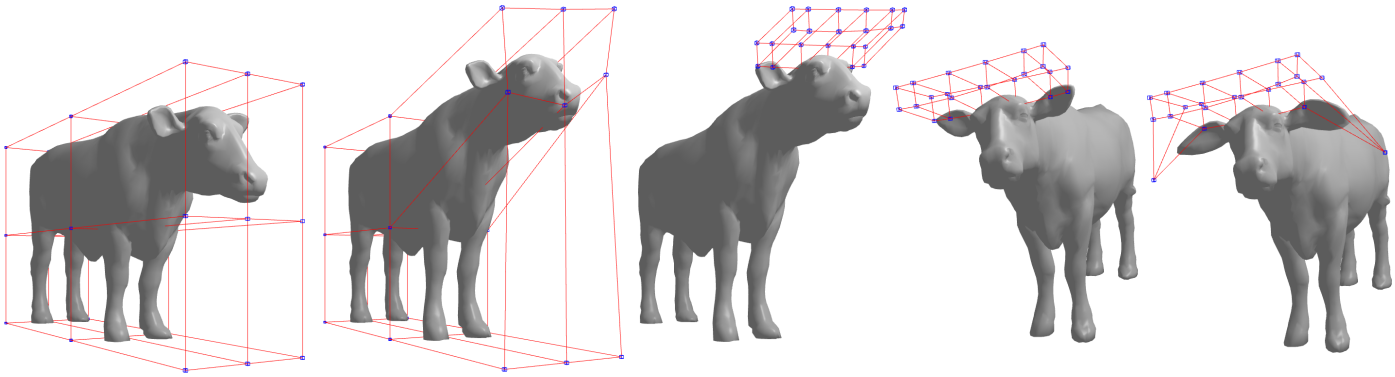

Figure 8: Far left: The input cow model (4K vertices, 7K faces), embedded in a tri-quadratic Bézier volume with $3 \times 3 \times 3$ control points (level zero). Left: Several vertices have been moved to adjust the global posture of the animal. Middle: Control structure of level two as requested by the user near the top of the head. Right: Same setting, but from a different viewpoint to prepare for local ear shape editing. Far right: Two control points of level two have been moved to adjust the shape of both ears.

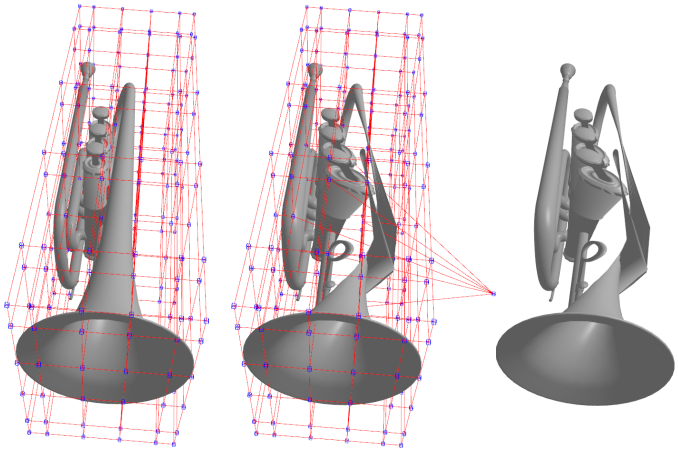

Figure 9: An example self-intersection. The trumpet model (obtained from [1]; 12K vertices, 23K faces) self-intersecting in consequence of the movement of control points. Our tool has the optional feature to prevent self-intersections when models are being deformed.

the maximum of four finer levels (on top of the initial level zero). In most scenarios, level four control point grids are sufficiently dense for precise manipulation. As levels are added to the grid, the possibility of hiding control points and their respective connecting lines of a certain level is given to the user, as shown in Figures 1 and 6.

5. Results and discussion

We now present further results obtained using our method and evaluate its effectiveness, especially from the point of view of interactive deformations with or without checking for self-intersections (Figures 5 and 9).

5.1. Control point hiding

As stated above, THB-splines provide a natural hierarchy in terms of basis functions from different refinement levels. A useful feature derived from this is control point hiding. As refinements are made, the user interface may become cluttered with control points. Therefore, our tool offers the possibility of showing only control points of certain levels. Examples of this are shown in Figures 1, 8 and 10. Note how this allows the user to focus only on a certain region with finer control over the deformation of models.

5.2. Performance evaluation

For an interactive tool based on FFDs, the efficiency of the evaluation of the underlying deformation function is of utmost importance. As the initial time for initialisation is often negligible and easily tolerated or even unnoticed by the user, the following performance analysis focuses on how efficient it is to update vertices, normals, and Jacobians as a control point is being manipulated. Naturally, as the grid receives more refinements, the control points of finer levels affect fewer vertices in the deformed model and therefore less time is required to update the model.

Figure 11 displays the region which is influenced by the selected control point on two models. The measured data (in milliseconds) are presented in Table 1, comparing two options: only vertex updates, and vertex and normal updates with self-intersection testing, computed in parallel.

Table 1: Performance table of our tests on two model situations; see Figure 11. All times are in milliseconds and averaged over many executions. FFD: updates only vertex positions. FFD+I: updates vertices, normals, and test for self-intersection is enabled. All cases make use of parallelism via OpenMP. Preprocessing times comprise computing parametric coordinates of vertices, THB-spline structure building, and initialising Jacobians.

Model	Suzanne	Violin case
All vertices	46K	527K
Active vertices	29K	485K
Preprocessing	97	532
FFD	4.0424	18.327
FFD+I	9.4882	90.228

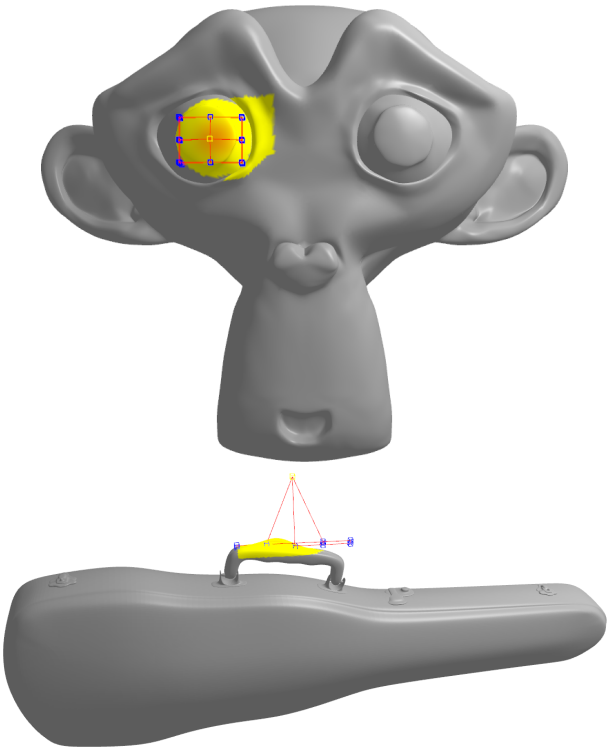

Figure 10: Level hiding example. The only control points being currently displayed are level three control points. All other control points are hidden to avoid visual clutter. The user can thus clearly focus on editing local details such as the left eye of Suzanne (top; Blender model) or the violin case handle (bottom).

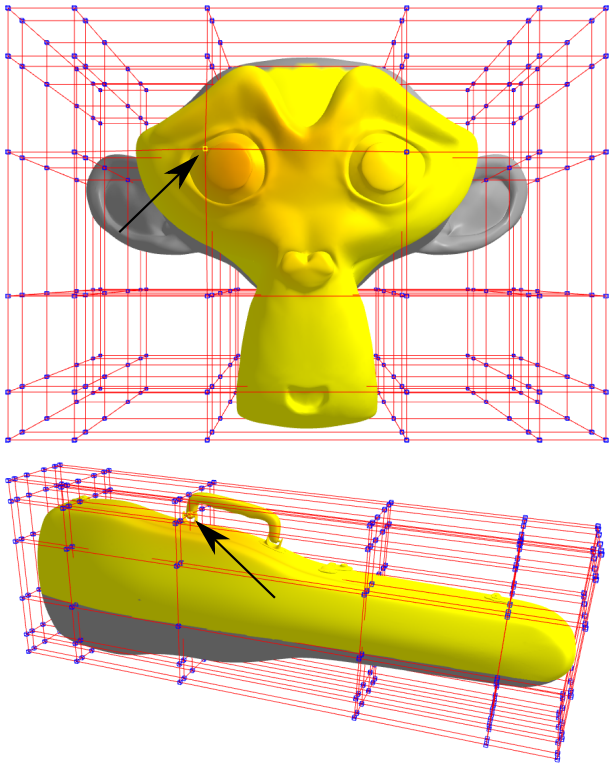

Figure 11: The two models used in our performance test. The highlighted regions show the active regions of the models when manipulating the selected control point (black arrow). Top: Suzanne (Blender). Number of active vertices: 29294 (out of 46590). Bottom: Violin case model. Number of active vertices: 485683 (out of 527252).

Preprocessing times are reported as well.

The first column (Suzanne) in Table 1 shows how negligible the performance impact is for about 29K vertices being updated at every frame. Using a parallel implementation of the self-intersection test and normal updating, we achieved the full update time of approximately 9.5 milliseconds, i.e., about 105 frames per second.

In contrast, the second column (violin case) shows a much greater performance impact of the self-intersection testing. With about 485K active vertices, the performance impact in the second column of Table 1 makes user interaction in real-time slightly challenging at about 11 frames per second. The tool was still responsive enough for slower manipulation of control points, which is generally sufficient in precise adjustments. On the other hand, the situation is rather extreme: the model is dense and most of the model (92%) is being actively deformed. This can often be avoided by the use of the THB-spline hierarchy.

THB-splines allow more precise deformations, just as in the case of modelling and fitting [14], with fewer control points than would be needed in a full tensor-product setting based on tri-variate B-splines. This typically reduces the active region and thus the number of active vertices, which in turn reduces the performance cost of manipulating the model, especially with self-intersection testing active.

The workstation used for all tests had 4 CPU cores at 2.6GHz and an Nvidia GTX970m graphics card.

5.3. Comparison to standard B-splines

As already mentioned, THB-splines provide a very suitable spline structure for FFDs. We now evaluate their performance with respect to the standard setting with B-splines.

In terms of preprocessing, most of the time is spent on calculating smooth normals and parametric coordinates of the vertices. Therefore, as also Table 1 indicates, the preprocessing time scales linearly with the object's complexity, i.e., its number of vertices. While a simpler and dedicated implementation of the standard B-spline structure might reduce preprocessing times, the amount of time spent on building the THB-spline structure is negligible in comparison with model loading and pre-computation of parametric coordinates. And as preprocessing speeds up user interaction, THB-splines are preferred.

In the same manner, memory usage is dominated by the amount of vertices. In our analysis, the amount of memory needed to retain the THB-spline structure was negligible in comparison to the general non-hierarchical structure. And so again, coupled with the fact that THB-splines allow for control structures with the same precision

Table 2: Control point (CP) influence for the Suzanne model (46590 vertices). The average numbers of active vertices (and percentages) at various levels l are listed. B-spline 6^3 is the tensor-product B-spline volume with $6 \times 6 \times 6$ control points, and similarly for B-spline 15^3. The last row corresponds to the THB-spline structure refined to level 3 in the left eye region of the Suzanne model, as shown in Figure 10, top.

B-spline setting	# CPs	$l = 0$	$l = 1$	$l = 2$	$l = 3$
B-spline 6^3	216	13803 (29.6%)	-	-	-
B-spline 15^3	3375	881 (1.8%)	-	-	-
THB-spline 6^3 fully refined to level 1	167	-	4082 (8.7%)	-	-
THB-spline 6^3 fully refined to level 2	3375	-	-	881 (1.8%)	-
THB-spline 6^3 locally refined to level 3	302	13907 (29.8%)	4263 (9.1%)	5892 (12.6%)	2134 (4.5%)

Table 3: Control points with no influence on the Suzanne model (46590 vertices). The numbers of control points (CPs) which do not affect any vertices are listed per level. The analysed (TH)B-splines are the same as those in Table 2. Note the efficiency of the hierarchical control structure in the last row.

B-spline setting	# CPs	$l = 0$	$l = 1$	$l = 2$	$l = 3$
B-spline 6^3	216	12	-	-	-
B-spline 15^3	3375	1396	-	-	-
THB-spline 6^3 fully refined to level 1	167	-	729	-	-
THB-spline 6^3 fully refined to level 2	3375	-	-	1396	-
THB-spline 6^3 locally refined to level 3	302	12	9	0	0

as B-splines but with a reduced number of control points, the hierarchical structure is preferred over the global structure from this perspective.

Another point of interest is the locality of deformations provided by THB-splines. In the general B-spline setting, adding more control points can hinder usability as the number of control points can easily grow to unmanageable levels if a certain deformation precision is needed.

Here, we investigate the two cases previously used in Section 5.1 and shown in Figure 10. We can obtain a reasonable estimate on how control points influence an object by averaging the number of vertices each control point influences. In Table 2, we compare THB-splines with sparse and dense B-splines. A B-spline volume controlled by $6 \times 6 \times 6$ control points fully refined to level 2 is equivalent to a B-spline volume with $15 \times 15 \times 15$ control points. However, the main advantage of THB-splines is the possibility of refining only where it is needed, such as in the last example in Table 2 (last row) refined to level 3 only in the region of the eye (Figure 10).

In Table 2, we can see that for the level 3 refined region (last row), the average number of active vertices is a bit higher than that of the denser B-spline. This is mainly due to the fact that in a dense B-spline, there is a considerable number of control points in areas with low vertex counts. The effect is more pronounced the less regular the shape of the object is. As reported in Table 3, the number of control points which do not affect the object increases considerably in a non-hierarchical scenario. These typically unwanted control points do not contribute to the deformation, but they still occupy memory and may impact processing time. Our implementation avoids this unneeded computation problem by using the incremental approach presented in Section 3.

In summary, our detailed analysis shows that THB-splines offer clear advantages over non-hierarchical B-splines in the context of FFDs.

5.4. Discussion

We have focused on the 3D setting, but all the employed concepts can be used in 2D or in higher dimensions, e.g., in the setting of surfaces or volumes embedded in $\mathbb{R}^4$ or on dynamic (time-dependent) meshes in $\mathbb{R}^{3+1}$. Also, while our current implementation supports only triangular meshes, more general polygonal meshes could be easily directly supported, although at a performance cost due to the use of more dynamic data-structures.

Further, the user could be given finer control over the refinement process. In principle, finer domains could be based on arbitrary knot values. We believe this would be overwhelming for users not closely familiar with non-uniform (hierarchical) B-splines and thus chose to use open-uniform knot-vectors, as also advocated in [16, Section 13].

We note that parallelism could be further exploited by employing the GPU. However, since our approach uses incremental updates it is not obvious how to perform this purely on the GPU. This remains an interesting avenue for future research.

All of the features shown and discussed above can be seen in action in the accompanying video showing an interactive session.

6. Conclusion

We have shown that THB-splines provide a natural and efficient hierarchical spline framework that is very well-

suited for use in free-form deformations. This combination provides a powerful and interactive deformation tool with features such as local and hierarchical deformations, control point hiding, and injectivity checking and fold-over prevention.

Our tool facilitates seamless manipulation of complex meshes. Interactive behaviour and self-intersection testing can be achieved simultaneously and interactively at the expense of pre-calculations prior to deformations. Finally, we have shown that CPU parallelism can be leveraged in creating an efficient deformation environment based on THB-splines and incremental updating.

It would be interesting to investigate how and to what extend our approach can be generalised to more versatile control structures such as those based on subdivision [24] or even truncated hierarchical subdivision methods [31].

Acknowledgement. This work is based on the first author's internship at the University of Groningen and BSc thesis at Universidade Federal do Rio Grande do Sul.

References

[1] Obj files. `http://people.sc.fsu.edu/~jburkardt/data/obj/obj.html`, 2017.

[2] OpenMP. `http://www.openmp.org/`, 2017.

[3] Mario Botsch, Pierre Alliez, Leif Kobbelt, Mark Pauly, and Bruno Lvy. *Polygon mesh processing*. CRC Press, 2010.

[4] Angelos Mantzaflaris (Coordinator). G+smo. `https://gs.jku.at/gismo`, 0.8.1 Alpha, August 2015.

[5] Sabine Coquillart. Extended free-form deformation: A sculpturing tool for 3D geometric modeling. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, pages 187–196, New York, NY, USA, 1990. ACM.

[6] Tor Dokken, Tom Lyche, and Kjell Fredrik Pettersen. Polynomial splines over locally refined box-partitions. *Computer Aided Geometric Design*, 30(3):331–356, 2013.

[7] E.J. Evans, M.A. Scott, X. Li, and D.C. Thomas. Hierarchical T-splines: Analysis-suitability, Bézier extraction, and application as an adaptive basis for isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 284:1 – 20, 2015.

[8] Jieqing Feng, Pheng-Ann Heng, and Tien-Tsin Wong. Accurate B-spline free-form deformation of polygonal objects. *Journal of Graphics Tools*, 3(3):11–27, 1998.

[9] Jieqing Feng, Jin Shao, Xiaogang Jin, Qunsheng Peng, and A. Robin Forrest. Multiresolution free-form deformation with subdivision surface of arbitrary topology. *The Visual Computer*, 22(1):28–42, Jan 2006.

[10] Michael S. Floater and Jiří Kosinka. On the injectivity of Wachspress and mean value mappings between convex polygons. *Advances in Computational Mathematics*, 32(2):163–174, Feb 2010.

[11] David R. Forsey and Richard H. Bartels. Hierarchical B-spline Refinement. *SIGGRAPH Comput. Graph.*, 22(4):205–212, June 1988.

[12] James E. Gain and Neil A. Dodgson. Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):289–298, October 2001.

[13] Carlotta Giannelli, Bert Jüttler, Stefan K. Kleiss, Angelos Mantzaflaris, Bernd Simeon, and Jaka Špeh. THB-splines: An effective mathematical technology for adaptive refinement in geometric design and isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 299:337–365, 2016.

[14] Carlotta Giannelli, Bert Jüttler, and Hendrik Speleers. THB-splines: The truncated basis for hierarchical splines. *Computer Aided Geometric Design*, 29(7):485–498, 2012.

[15] W. J. Gordon and R. Riesefeld. B-spline curves and surfaces. *Computer Aided Geometric Design*, (4):95–126, oct 1974.

[16] William J Gordon and Richard F Riesenfeld. *B-spline curves and surfaces*, pages 95–126. Academic Press, New York, 1974.

[17] Josef Griessmair and Werner Purgathofer. Deformation of Solids with Trivariate B-Splines. In *EG 1989-Technical Papers*, pages 137–148. Eurographics Association, 1989.

[18] Prem Kalra, Angelo Mangili, Nadia Magnenat Thalmann, and Daniel Thalmann. Simulation of facial muscle actions based on rational free form deformations. *Computer Graphics Forum*, 11(3):59–69, 1992.

[19] Jiří Kosinka and Michael Bartoň. Injective shape deformations using cube-like cages. *Computer-Aided Design and Applications*, 7(3):309–318, 2010.

[20] Jiří Kosinka, Malcolm A. Sabin, and Neil A. Dodgson. Control vectors for splines. *Computer-Aided Design*, 58:173–178, 2015. Solid and Physical Modeling 2014.

[21] R. Kraft. *Adaptive und linear unabhängige Multilevel B-Splines und ihre Anwendungen*. PhD thesis, Universität Stuttgart, 1998.

[22] Henry J. Lamousin and Warren N. Waggenspack Jr. NURBS-based free-form deformations. *IEEE Comput. Graph. Appl.*, 14(6):59–65, November 1994.

[23] Charles Teorell Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Dept. of Mathematics, University of Utah, 1987.

[24] Ron MacCracken and Kenneth I. Joy. Free-form deformations with lattices of arbitrary topology. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 181–188, New York, NY, USA, 1996. ACM.

[25] Aishwarya Pawar, Yongjie Zhang, Cosmin Anitescu, Yue Jia, and Timon Rabczuk. DTHB3DReg: Dynamic truncated hierarchical B-spline based 3D nonrigid image registration. *Communications in Computational Physics*, 23(3):877–898, 2018.

[26] Aishwarya Pawar, Yongjie Zhang, Yue Jia, Xiaodong Wei, Timon Rabczuk, Chiu Ling Chan, and Cosmin Anitescu. Adaptive FEM-based nonrigid image registration using truncated hierarchical B-splines. *Computers & Mathematics with Applications*, 72(8):2028–2040, 2016.

[27] Roi Poranne and Yaron Lipman. Provably good planar mappings. *ACM Trans. Graph.*, 33(4):76:1–76:11, July 2014.

[28] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH '86 Computer Graphics*, 20(4):151–160, aug 1986.

[29] T.W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCs. *ACM Transactions on Graphics*, 22(3):477–484, 2003.

[30] Wenhao Song and Xunnian Yang. Free-form deformation with weighted T-spline. *The Visual Computer*, 21(3):139–151, Apr 2005.

[31] Xiaodong Wei, Yongjie Zhang, Thomas J.R. Hughes, and Michael A. Scott. Truncated hierarchical Catmull-Clark subdivision with local refinement. *Computer Methods in Applied Mechanics and Engineering*, 291:1–20, 2015.

[32] Xiaodong Wei, Yongjie Zhang, Lei Liu, and Thomas J.R. Hughes. Truncated T-splines: Fundamentals and methods. *Computer Methods in Applied Mechanics and Engineering*, 316:349–372, 2016.

[33] Zhiyong Xie and Gerald E. Farin. Mathematical methods for curves and surfaces. chapter Deformation with Hierarchical B-splines, pages 545–554. Vanderbilt University, Nashville, TN, USA, 2001.

[34] Zhiyong Xie and Gerald E. Farin. Image registration using hierarchical B-splines. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):85–94, January 2004.