Evaluating Box Splines with Reduced Complexity

Joshua Horacsek*, Usman Alim

Department of Computer Science, University of Calgary, 2500 University Dr NW, Calgary, AB T2N 1N4

Abstract

For the class of non-degenerate box splines, we present a set construction scheme that yields the explicit piecewise polynomial form for an arbitrary box spline. While it is possible to use the well known recursive formulation to obtain these polynomial pieces, this is quite expensive. Our construction is theoretically less expensive than the recursive formulation and allows us to evaluate box splines with more direction vectors than what would be feasible under the recursive scheme. Finally, using the explicit polynomials in each region of the box spline, we show how to create fast evaluation schemes using this explicit characterization and a spatial data structure.

1. Introduction

Box splines are a multivariate generalization of B-splines [2] that are particularly useful in visualization and signal processing applications. They allow practitioners to tailor approximation schemes (on regular grids) to specific computational and theoretical parameters. These parameters include *support size* (how many samples are needed to reconstruct a value), *smoothness* of reconstruction, and *order of approximation*. There is particular interest in using box splines to span approximation spaces on non-Cartesian lattices, since certain non-Cartesian lattices, like the Body Centered Cubic (BCC) lattice, allow more efficient sampling and reconstruction schemes [6].

Typically, to evaluate a box spline, one either relies on the recursive scheme [2, 1, 7], or one must derive an explicit representation of the spline which has so far been done on a case-by-case basis (see [3] as an example). In this paper, we derive an explicit form for a box spline's piecewise polynomial — this form is analogous to the well known piecewise polynomial expression for B-splines. We give a set construction recipe that yields the explicit truncated piecewise polynomial form for any box spline. From this, we derive the explicit polynomial within each region of evaluation. We note that this can also be accomplished by using the recursive form of evaluation restricted to a particular region of a spline. Once these have been computed, we construct a binary space partitioning tree that partitions the support of the box spline into regions of evaluation. Each leaf of the tree is then assigned an explicit polynomial. To evaluate the box spline, one must simply traverse the tree to determine the region of the mesh, then evaluate the polynomial in that region.

*Corresponding author

Email addresses: joshua.horacsek@ucalgary.ca (Joshua Horacsek), ualim@ucalgary.ca (Usman Alim)

When one wishes to evaluate a box spline, the task is not so trivial. The most widely known evaluation scheme is likely de Boor's recursive formula [2]. Unfortunately, this is quite unstable and quite expensive to evaluate, de Boor proposed a solution to this, but the solution is still not completely robust in dimensions higher than 2 [7, 5]. Kobbelt attempted to analyze and address these issues [1], yet there still appears to be numerical instabilities in high dimensions [5]. Moreover, these evaluation schemes tend to be fairly slow as they are recursive and rely on solving a system of equations at each step of the recursion. There are heuristics that may overcome some issues with this form, but it is generally accepted that creating some explicit representation within each region of the box spline's mesh, then using that explicit form of evaluation is much more computationally efficient.

In the work of Kim *et al.* [5], a fast and stable scheme is derived by creating an indexing system that quickly identifies each region of evaluation within the box spline's mesh. Within each region of evaluation, the Bernstein Bézier (BB) form of the piecewise polynomial is then derived from a stable form of the recursive evaluation formula. However, there are a few pitfalls to this method. Firstly, it relies on a stable evaluation scheme to derive the BB form of each polynomial. Secondly, its indexing scheme must be derived on a case-by-case basis and is only provided for particular box splines with Cartesian-like meshes. Finally, it relies on certain assumptions about the direction vectors that constitute a box spline, namely that they are rational.

The only assumptions we make in this work are that our direction vectors are real-valued, and must span the target space R^s. Additionally, the method proposed in this work is exact, our decomposition makes no approximations, and only relies on operations that are performed exactly within a machine. Additionally, our work may be used to augment other evaluation schemes. For example, we may use our results to speed up the precomputation step of the work by Kim *et al.* [5]. Conversely, if a box spline has appropriate structure, we may use said work of Kim *et al.* to derive a slightly more efficient indexing scheme of a box spline's mesh.

When compared to de Boor's well known recursive evaluation scheme [2, p. 17], we find that our decomposition often takes significantly fewer operations for more direction vectors. However, they work by different mechanisms — the recursive evaluation scheme uses the property that evaluation of a box spline is equivalently stated as a weighted sum of smaller box splines. Our method is to split the box spline into two spatial parts, the difference operator and Green's function.

The Fourier representations of these two parts of a box spline are known in closed form and are quite simple to characterize. Their spatial characterizations are less simple. However, in this work we are able to derive their spatial forms from a series of set constructions. The final result comes from a semi-discrete convolution between these two representations. The paper proceeds as follows. In Section 2, we provide a basic introduction to box splines and the notation we use to proceed with this work. In Section 3, we derive the spatial form for the difference operator via a simple set construction akin (i.e. distributionally equivalent) to the inverse discrete time Fourier transform of the difference operator. We proceed in Section 4 to construct the spatial form of the Green's function. At first glance, the Fourier form of the Green's function appears to be non-separable, which complicates obtaining a spatial representation. However, by constructing vectors from the kernel of the matrix that defines a box spline, we can always construct a distributionally equivalent form that *is* separable, and therefore has an inverse Fourier transform that is easy to

characterize. We then combine those forms via a semi-discrete convolution. In Section 5, we show how to use the difference operator and the Green's function to perform stable pointwise evaluation; we also provide bounds on the number of operations needed for evaluation at a single point. This form, while theoretically faster than the recursive form, is still quite expensive to evaluate pointwise. In Section 6, we show how to use the previous results to derive the explicit piecewise polynomial for each polynomial region; we then show how to build a fast tree structure to index these regions, which fulfills the promise of fast evaluation. Finally, in Section 7, we provide exemplary constructions for some known box splines.

2. Background

Following the established notation [2], we define a box spline as a generalized function $M : \mathbb{R}^s \to \mathbb{R}$ which is characterized by a list of n column vectors $\xi_1 \; \xi_2 \; \cdots \; \xi_n$, with each $\xi_i \in \mathbb{R}^s$, that are collected in an $s \times n$ matrix $\Xi := [\xi_1 \; \xi_2 \; \cdots \; \xi_n]$:

The function M is defined recursively by the convolution equation

$$M_{[\,]} := \delta \quad \text{and} \quad M_{\Xi} := \int_0^1 M_{\Xi \setminus \xi} (\cdot - t\,\xi)\, dt \tag{1}$$

where δ is the Dirac delta distribution. We say the box spline is non-degenerate if the dimension of the range of Ξ is s. When we speak of the kernel of a matrix, we speak of a specific basis for the null-space of that matrix. We explicitly choose the basis for the kernel so that it is in column echelon form. We use the notation $\ker X$ as shorthand for the null-space of X. For example, $\ker \Xi$ is the $n \times (n - s)$ matrix that forms a basis for the null-space of Ξ and is in column echelon form. We write $z \in \ker X$ if $Xz = 0$.

If a hyperplane defined by columns of Ξ forms an $s - 1$ dimensional subspace of $\mathbb{R}^s$, then we collect it in the set $\mathcal{H}(\Xi)$. Since our direction vectors are real valued, we define the mesh of Ξ as

$$\mathcal{Z}(\Xi) := \bigcup_{H \in \mathcal{H}} H + \mathbb{Z}^n. \tag{2}$$

We denote the Fourier equivalence between two functions with the symbol "$\widehat{\;}$". For example, the function $f(x)$ has the Fourier transform $\hat{f}(\omega)$, which we also write as $f(x) \widehat{\;} \hat{f}(\omega)$. We use the convention that the Fourier transform is defined distributionally as $\hat{f}(\omega) := \int_{\mathbb{R}^s} f(x) \exp(-i\,\omega \cdot x)\, dx$. Of pivotal importance to us is the Fourier representation of a box spline

$$\widehat{M_\Xi}(\omega) = \prod_{j=1}^{n} \frac{1 - \exp(-i\,\omega \cdot \xi_j)}{i\,\omega \cdot \xi_j} \tag{3}$$

which we split into two functions, the difference operator

$$\widehat{r_\Xi}(\omega) := \prod_{j=1}^{n} 1 - \exp(-i\,\omega \cdot \xi_j) \tag{4}$$

and the Green's function

$$\widehat{G}(\omega) := \prod_{j=1}^{n} (i\omega_j)^{-1} \tag{5}$$

which, when applied with $\widehat{D} := \prod_{j=1}^{n} i\omega_j$ produces the Dirac impulse.

To index an element of a vector we write (j) where j is an integer index. We also define the n-dimensional vector w as $w(j) := i\omega_j$. For the -power function, we use the definition $x^{\alpha} := \prod_{j=1}^{m} x(j)^{\alpha(j)}$ where x is an m-dimensional variable. It should be understood that by x^{-} we explicitly mean $1/x$. With these notations, the Green's function (5) can be written more succinctly as $\widehat{G}(\omega) = w^{-1}$.

We also adopt the notations

$$[\![x]\!] := \prod_{j=1}^{m} \frac{x(j)^{\alpha(j)}}{\alpha(j)!}; \tag{6}$$

$$[\![x]\!]_{sgn} := \prod_{j=1}^{m} \frac{(x(j))_{sgn}^{\alpha(j)}}{\alpha(j)!}; \tag{7}$$

$$[\![x]\!]_{+} := \prod_{j=1}^{m} \frac{(x(j))_{+}^{\alpha(j)}}{\alpha(j)!}; \tag{8}$$

for the normalized, normalized signed and normalized one-sided -power functions respectively. In the above, $(x)_{sgn}^{k} := x^k sgn(x)/2$ and $(x)_{+}^{k} := x^k H(x)$ where $H(x)$ is the heaviside distribution. We write $\|\alpha\|_0$ to count the non-zero elements within . If is an n-dimensional vector, we can write the $s \|\alpha\|_0$ matrix , which denotes the subset of that are selected by the non-zero $\alpha(j)$. Finally we use $0; 1$ to denote vectors consisting of all 0 or 1 respectively, and the dimension of the vector should be inferred from context in which it is being used.

The crux of this work is establishing explicit representations of the spatial forms of the difference operator and the Green's function of the box spline. The final piecewise polynomial form will result from the semi-discrete convolution of these two forms.

3. Difference Operator

The difference operator has a simple characterization in the spatial domain which can be constructed with the help of the following definition.

Definition 3.1. *For an s n matrix with full rank, define the sequence of multi-sets S by the rules*

$$S_{[\,]} := \{(0; 0)\} \text{ and } S_{[\alpha]} := \{(c; \beta + p) : (c; p) \in S_{[\beta]}\} \cup S_{[\beta]} : \tag{9}$$

For each tuple in each multi-set of this sequence, the first element of the tuple is a scalar value and the second element is a vector in $\mathbb{R}^s$.

From this definition we derive the following lemma.

Lemma 3.2. *Let be an s n matrix with full rank, then the difference operator r for M is given by the distribution*

$$r = \sum_{(c;p)\in S} c (p) \tag{10}$$

where is the Dirac delta distribution.

Proof. We show this inductively on the number of columns of . As a base case, assume contains a single non-zero direction vector, . By definition, we have

$$S = f(1;0);(1;)g:$$

We also have $\widehat{r} = 1 \exp(i !)$ which has the spatial form $r = ( )$ which is by definition

$$r = \sum_{(c;p)\in S} c (p): \tag{11}$$

For the general case, pick some with $n > 1$ columns, and suppose that for any $'$ with $i < n$ columns this result holds. Then split into an s by $(n 1)$ matrix $'$ and column vector , and consider the difference operator $r_{\cup}$. From the Fourier expression we have

$$\widehat{r} = r \setminus_{\cup} = \prod_{\in \cup} (1 \exp(i !)) = (1 \exp(i !)) \widehat{r}_{};$$

which is equivalently stated in the spatial domain as

$$r = r_{\cup} = r_{} r_{}( ): \tag{12}$$

The inductive hypothesis then gives

$$r_{\cup} = \sum_{(c;p)\in S_{}} c (p) + \sum_{(c;p)\in S_{}} c ((p +)) \tag{13}$$

and it follows, by definition of S, that

$$r = r_{\cup} = \sum_{(c;p)\in S} c (p) \tag{14}$$

which completes the induction. □

From this, the following corollary is clear.

Corollary 3.2.1. *For any S, if we there exists two tuples $(a; p)$ and $(b; q)$ 2 S with the property $p = q$ then those two tuples may be removed and be replaced by $(a + b; p)$, leaving Definition 3.1 unaffected.*

4. Green's Function

The Green's function of a box spline allows us to characterize the polynomial within each region of the box spline's mesh via semi-discrete convolution with the box spline's difference operator in the spatial domain. To do this, we need the spatial form of the Green's function. At first this appears quite difficult as the convolutional definition gives no information regarding the Green's function of a box spline. Turning to the Fourier form appears to be more tractable, but the non-separability of the Fourier form seems prohibitive. The approach is to look at the non-separable Green's function and apply a series of transformations so that $\widehat{G}(!) = w^{-1}$ becomes separable. More explicitly, we hope to split w^{-1} into a sum of weighted terms $w^{-1} = c_1 w^{-\ 1} + \cdots + c_j w^{-\ j}$ where each $\ _i$ has the property $jj\ \ _i jj_0 = s$ and is thus separable. This is an approach taken in the scientific visualization literature to derive fast evaluation schemes for certain box splines [3], and we generalize it here. The observation to make is that if we choose any vector $v \in \ker$, then $v \cdot w = 0$. This allows us to write $w^{-1} = w^{-1}\frac{v(i)w(i)-v\cdot w}{v(i)w(i)}$, provided $v(i) \neq 0$. This reduces the zero-norm of the exponents in the resultant expression by one. The question then becomes, how do we choose vectors from the kernel so that, after repeated applications of this process in general, we decrease the zero-norm of every exponent to s? This is formalized in the following definition.

Definition 4.1. *Take to be an $s \times n$ matrix with full rank, and take any with dimension n and $s < jj\ \ jj_0 \leq n$. Let $b'_1 \cdots b'_n$ be the rows of* ker *, then construct an $(n - jj\ \ jj_0) \times (n - s)$ matrix C where b'_j is included in this matrix iff $(j) = 0$. Let t be the first column vector of* ker C *, then define* := ker t *. If $jj\ \ jj_0 = n$ then such a vector does not necessarily exist, in this case we take as the first column vector of* ker .

The vectors are the exact vectors we need to incrementally decompose w^{-1}, we first note that each has the following property by construction.

Proposition 4.2. *For and as above, the vector has the property that if $(j) = 0$ then $(j) = 0$.*

Proof. This is immediate if $jj\ \ jj_0 = n$: So suppose that $s < jj\ \ jj_0 < n$, and suppose that $(j) = 0$ and $(j) \neq 0$. But then we have $b'_j\ \ t \neq 0$ so it must be that $t \notin \ker C$: □

We now define a sequence of sets P_k that will use Definition 4.1 to recursively decompose the Green's function. The elements of P_k will represent the terms $c_1 w^{-\ 1} + \cdots + c_j w^{-\ j} = G(!)$ in the previous discussion — we will eventually prove that, as k gets larger, the zero norm of the exponents in P_k decreases. The terms in this sum are encoded as the tuples $(c_i;\ \ _i)$ in the following definition.

Definition 4.3. *Let be an $s \times n$ matrix with full rank, define the sequence of sets P recursively by the rules*

$$P_0 := \{(1; 1)\} \tag{15}$$

$$P_k := \left\{ \left(c\frac{(j)}{(m\ )}; r_j(\ \ ) \right) : (c;\ \ ) \in P_{k-1}; m\ \ < j \leq n; \quad (j) \neq 0 \right\} \tag{16}$$

with $m := \min\{i : (i) \neq 0\}$ *and the vector function* r_j *defined as*

$$r_j : \mathbb{Z}^n \to \mathbb{Z}^n : (i) \mapsto \begin{cases} (i) + 1 & ; \text{ if } i = m \\ (i) - 1 & ; \text{ if } i = j \\ (i) & ; \text{ otherwise} \end{cases} \tag{17}$$

and $k \leq n$ *s: Here, each* P_i *is a set of tuples where the first element of each tuple is a real scalar value, and the second element is a vector in* $\mathbb{R}^n$.

Before we use this set to decompose the Green's function of M, we need some supplementary results.

Proposition 4.4. *If* $(c, \ ) \in P_k$, *then for all* l *such that* $m < l \leq n$; *we have*

$$(l) \in \{0, 1\}. \tag{18}$$

Proof. If $k = 0$, then this is true by definition. Next, for the inductive case, suppose that this is true for all j such that $0 \leq j < k$, and pick any $(c, \ ) \in P_k$. Since $k > 0$, there exists a $(d, \ ) \in P_{k-1}$ such that $c = d\frac{(l)}{(m)}$; and $= r_l(\ )$; for some l and m. If $m = m$; then $(i) \in \{0, 1\}$, for all i such that $m < i \leq n$, but by construction in Equation (16), we know $(i) \in \{0, 1\}$ for all $m < i \leq n$. But then $(i) \in \{0, 1\}$ for all $m < i \leq n$, which completes the proof. So suppose that $m < m$. Recall by construction, we have $(m) \neq 0$ and $(m) \neq 0$. Combine this fact with the definition $:= \ker t$, $:= \ker t$, and with the fact that $\ker$ is in column echelon form, we have

$$\min\{i : t(i) \neq 0\} < \min\{i : t(i) \neq 0\}. \tag{19}$$

Since $t \in \ker C$ forces certain elements of to be zero, and t forces the same elements of to be zero (and perhaps more), we know $t \in \ker C$ as well. But then t could not have been the first column of $\ker C$, since there exists a column with a lower-indexed, non zero row (by equation (19)), and $\ker C$ is in column echelon form.

There is however, one case left to show, since for $k = 1$, no such t exists. Thus if $k = 1$, choose any $(c, \ ) \in P_1$, and let $(d, \ ) \in P_0$ (chosen as above). Again, by the same logic as above, if $m = m$; then the proof is completed. So suppose that $m < m$. But since both $\in \ker$ and $\in \ker$, $(m) \neq 0$ and $(m) \neq 0$ and $\ker$ is in column echelon form, this contradicts our choice of since it could not have been the first column of $\ker$. □

Proposition 4.5. *If* $(d, \ ) \in P_k$ *for* $0 \leq k < n \leq s$, *and* $(c, \ ) \in P_{k+1}$, *with*

$$c = d\frac{(j)}{(m)}; \text{ and } = r_j(\ ) \tag{20}$$

for some $m < j \leq n$, *then*

$$\| - \|_0 = \| \|_0 - 1. \tag{21}$$

Proof. We show this directly. Choose any $(d, \ ) \in P_k$ and $(c, \ ) \in P_{k+1}$ as stated in the proposition. Then, by Proposition (4.4), we know that $(i) \in \{0, 1\}$ for all i such that $m < i \leq n$, by construction in Equation (16) exactly one of these non-zero (i) will be decremented (and (m) will be incremented) to become , thus $\| - \|_0 = \| \|_0 - 1$. □

Proposition 4.6. *If* $(c; \ ) \ 2 \ P_k; 0 \ \ k \ \ n$*, then* *has full rank.*

Proof. For P_0 this is holds by assumption. Next, assume this holds for all $(d; \ ) \ 2 \ P_j$ where $0 \ \ j < k$. Pick any $(c; \ ) \ 2 \ P_k$: There exists some $(d; \ ) \ 2 \ P_{k-1}$, such that $= r_l(\ )$ with $m < l \ \ n$, for some m and l. By construction of in Equation (16), we have $\not\sim$. But by Proposition 4.5, these differ by exactly one column vector, specifically Equation (16) tells us that we have $_l \ 2$, and $_l \ \not\in$. It remains to show that $_l$ can be written as a linear combination of vectors only in (not including $_l$ itself). But we also know

$$\sum_{i=1}^{n} \ _i \ \ (i) = 0 \tag{22}$$

since $2 \ker$. By Proposition 4.2, we can write this as

$$\sum_{i=1; i \neq l}^{n} \ _i \ \ (i) = \ \ (l) \ _l: \tag{23}$$

with $(l) \not\sim 0$ by construction in Definition (4.3). Note that all the non-zero elements of correspond to column vectors of thus we can write $_l$ as the weighted sum of vectors from : Since had full rank, has full rank, completing the induction. □

Lemma 4.7. *Let* *be an* $s \ \ n$ *matrix with full rank, then*

$$\widehat{G}(!) = \sum_{(c; \ ) \in P_k} c \ w^{-} \tag{24}$$

for $0 \ \ k \ \ n \ \ s$*:*

Proof. When $k = 0$ the result is true by definition. Suppose the result holds for all j such that for $0 \ \ j < k$. By assumption we have

$$\widehat{G}(!) = \sum_{(c; \ ) \in P_{k \ 1}} c \ w^{-} \ : \tag{25}$$

Next, since $2 \ker$ we have $w = 0$, re-arranged this is

$$\frac{(m \ )w(m \ ) \qquad w}{(m \ )w(m \ )} = 1: \tag{26}$$

So we certainly have

$$\widehat{G}(!) \ = \sum_{(c; \ ) \in P_{k \ 1}} (c \ w^{-} \ ) \frac{(m \ )w(m \ ) \qquad w}{(m \ )w(m \ )} \tag{27}$$

$$= \sum_{(c; \ ) \in P_{k \ 1}} \sum_{l=m \ +1}^{n} c \frac{(l)}{(m \ )} w^{-r_l(\ )} \tag{28}$$

which is, by definition,

$$\sum_{(c;\ )\in P_k} c\, w^{-}\ ; \tag{29}$$

completing the induction. □

Similar to Corollary 3.2.1, it follows that we can collect like terms at each iteration of this set construction, which is summarized in the following Corollary.

Corollary 4.7.1. *If we have two tuples* $(a;\ );(b;\ ) \in P_k$ *with* $=$ *, then we may replace them both by a single tuple* $(a+b;\ )$.

The next definition helps us "squish" n-dimensional vectors into appropriate s-dimensional vectors, which will aid in the transformation from Fourier form to the final spatial form.

Definition 4.8. *For any* $$ *of dimension* n *with* $jj\ \ jj_0 = s$ *define* $$ *as the* s*-dimensional vector obtained by retaining the non-zero elements of* $$*. Define also the corresponding vector* $$ *that indexes the non-zero elements of* $$*.*

For example, if $s = 2$ and $= (0;2;0;1)$, then $= (2;1)$ and $= (2;4)$. With this, we now define a helpful auxiliary function.

Definition 4.9. *For any* $$ *with dimension* n *and* $jj\ \ jj_0 = s$*, define the function*

$$T\ (\ !\ ) := \prod_{j=1}^{s} \frac{1}{i\ !\ (j)\ \ ^{(j)}}:$$

Proposition 4.10. *For any* $$ *with dimension* n *and* $jj\ \ jj_0 = s$ *then*

$$T\ (\ !\ )\ \ \ !\ [\![x]\!]_{sgn}^{-1}:$$

Proof. Using the Fourier equivalence

$$\frac{1}{(i\ !\ )^k}\ \ \ !\ \frac{(x)_{sgn}^{k-1}}{(k\ \ 1)!};\ k\ 2\ \mathbb{Z}^{+};\tag{30}$$

we write

$$T\ (\ !\ ) = \prod_{j=1}^{s} \frac{1}{i\ !\ (j)\ \ ^{(j)}}\ \ \ !\ \prod_{j=1}^{s} \frac{(x(j))_{sgn}^{(j)-1}}{(\ \ (j)\ \ 1)!} = [\![x]\!]_{sgn}^{-1}$$

by definition. □

Proposition 4.11. *For any* $s\ \ n$ *matrix* $$ *with full rank, and any* $$ *with dimension* n *and* $jj\ \ jj_0 = s$ *then*

$$w^{-}\ \ \ !\ \frac{[\![\ ^{-1}x]\!]_{sgn}^{-1}}{j\det\ \ j}:$$

Proof. Since $jj \quad jj_0 = s$ we can write

$$w^{-} = \prod_{j=1}^{s} \frac{1}{w(\quad(j))^{\quad(j)}} \tag{31}$$

$$= \prod_{j=1}^{s} (i^{\quad(j)} \quad !)^{-\quad(j)} \tag{32}$$

$$= T(\quad^{T}!); \tag{33}$$

and

$$T(\quad^{T}!) \quad ! \; j\det \quad^{-1}j[\![\quad^{-1}x]\!]_{sgn}^{-1}: \tag{34}$$

□

Lemma 4.12. *Let be an* $s \quad n$ *matrix with full rank, then*

$$G(x) = \sum_{(c;\ )\in P_n\ \ s} \frac{c}{j\det \quad j}[\![\quad^{-1}x]\!]_{sgn}^{(\quad^{-1})}: \tag{35}$$

Proof. Starting with

$$\widehat{G}(!) = \sum_{(c;\ )\in P_n\ \ s} cw^{-}: \tag{36}$$

then using Propositions 4.5 and 4.11 we obtain the form stated in the lemma, the only thing left to check is that this is well defined, .i.e we must verify that $\det \quad \neq 0$, but this is true by Proposition 4.6. □

Theorem 4.13. *Let be an* $s \quad n$ *matrix that corresponds to a non-degenerate box spline, then the box spline is given by*

$$M(x) = \sum_{(b;p)\in S} \sum_{(c;\ )\in P_n\ \ s} \frac{b \quad c}{j\det \quad j}[\![(\quad^{-1}x) \quad p]\!]_{+}^{(\quad^{-1})}: \tag{37}$$

Proof. This is a direct consequence of Lemma 3.2 and 4.12, combined with the fact that the difference operator annihilates all polynomials of degree lower than the order of the spline [2, I.32]. □

One can think of this as a generalized version of the one dimensional expression for B-splines

$$^{n}(x) = \sum_{j=0}^{n+1} (\quad 1)^{j} \binom{n+1}{j} [\![x \quad j]\!]_{+}^{n} \tag{38}$$

where n is the order of the un-centered B-spline.

5. Evaluation at a Point

While the form in Equation (37) is explicit, it is important to note that it is a distributional equality. In general, it may be that the term $[\![(\ ^{-1}x)\quad p]\!]_+^{(\ \ -1)}$, which, even though we perform all arithmetic exactly within a computer algebra system, involves computing a discontinuous step function, which is in general not computable [1]. This is also an issue when using the recursive evaluation scheme, given by

$$(n\quad s)M\ (x) = \sum_{\in} t\ M\ _{\setminus\{\ \}}(x)\quad (1\quad t\ )M\ _{\setminus\{\ \}}(x\quad\ ) \tag{39}$$

with $t\ =\ \ ^T(\ \ ^T)^{-1}x$ [2, p. 17]. When the evaluation point is on a knot plane, one way to overcome numerical instabilities is to evaluate portions of (37) away from the knot plane. This effectively pushes the evaluation into the interior of one of the mesh's regions. That is, if we have any point $x\ 2\ H$, then we choose some vector with sufficiently small length such that $+x\ 2\ H$. We then note that we may split

$$[\![(\ ^{-1}x)\quad p]\!]_+^{(\ \ -1)} = [\![(\ ^{-1}x)\quad p]\!]_+^0[\![(\ ^{-1}x)\quad p]\!]^{(\ \ -1)};$$

Where $[\![(\ ^{-1}x)\quad p]\!]_+^0$ is a discontinuous "indicator" (i.e. a step function) and $[\![(\ ^{-1}x)\quad p]\!]^{(\ \ -1)}$ is the polynomial part. In fact, when $jj\ \ jj_2$ is small enough, we may write

$$[\![(\ ^{-1}x)\quad p]\!]_+^{(\ \ -1)} = [\![(\ ^{-1}(x\quad\ ))\quad p]\!]_+^0[\![(\ ^{-1}x)\quad p]\!]^{(\ \ -1)}$$

this effectively pushes the evaluation of $[\![(\ ^{-1}x)\quad p]\!]_+^0$ away from a knot into one of the surrounding regions, however the term $[\![(\ ^{-1}x)\quad p]\!]^{(\ \ -1)}$ remains on the plane, which is valid because it is a polynomial and therefore continuous; only, of course, provided that the box spline is continuous at x. This is similar to the method used by de Boor to increase the stability of the recursive evaluation scheme [1]. This gives the following scheme

$$M\ (x) = \sum_{(b;p)\in S}\ \sum_{(c;\ )\in P_{n\ \ s}} \frac{b\quad c}{j\det\quad j}[\![\ ^{-1}(x\quad\ (x))\quad p]\!]_+^0[\![(\ ^{-1}x)\quad p]\!]^{(\ \ -1)} \tag{40}$$

with

$$(x) := \begin{cases} \quad & \text{if } x\ 2\ H \\ 0 & \text{otherwise.} \end{cases} \tag{41}$$

Again, we push the evaluation of the step function in Equation (37) into the interior of a region where it is computable, but we leave the value of x untouched in the evaluation of the polynomial. We dislike the need for a "fudge" factor , we would prefer a method that is general and stable regardless. Moreover, we find that the sets P_{n-s} and S can get quite large, so evaluation of Equation (40) can get quite expensive. We will return to the question of fast evaluation in the next section, where we use (37) to derive the explicit piecewise polynomials within each region of

the box spline mesh, in conjunction with a binary space partitioning tree to efficiently index these polynomials.

It is important to note that, even with the trick of evaluating parts of Equation (37) away from the knot planes, evaluating Equation (40) with floating point arithmetic may propagate floating point round-off error worse than the recursive scheme. To this end, it is better to use Equation (37) with to derive the explicit polynomial for that region, then use a Horner scheme to efficiently evaluate this polynomial with reduced error [8, 9]. We now proceed to bound the size of the sets P_{n-s} and S , which will allow us to bound the complexity of (40).

Proposition 5.1. *For the sets S and P_j we have the following bounds on their sizes*

$$jS\ j \quad j \quad f0;1g^n j \text{ and } jP_j j \quad p(n;n \quad j)\binom{n \quad 1}{n \quad j \quad 1} \tag{42}$$

where $p(n;k)$ is the number of partitions of the integer n of size k and the set $f0;1g^n$ is the set of distinct points obtained by projecting the vertices of the unit n-dimensional cube into s-dimensional space using the matrix.

Proof. To see the first inequality, we note that each tuple $(a;p)\ 2\ S$ has the property that p is the sum of some subset of the vectors in . To see this we notice that p corresponds to some term in $\prod_{j=1}^{n}(1 \quad \exp(\ i\ _j\ !))$. Thus, we know there is at least one element in the pre-image $x\ 2\ f0;1g^n$ such that $x = p$. However, not every $x\ 2\ f0;1g^n$ maps to some such p, this establishes the inequality. A simple counting argument gives the second bound — consider the elements of P_j, they are of the form $(c;\ )$. Note that we have $\sum\ (i) = n$. Moreover, we also know that the non zero elements of are ordered and that $jj\ \ jj_0 = n \quad j$, thus we conclude that, if we remove the zeros from we have a partition of n with size $n \quad j$. Thus, since P_j has no elements with duplicate , if we remove the zeros from each we end up with at most $p(n;n \quad j)$ unique elements. To arrive at a bound on $jP_j j$ we must add back the zeros. If we take any partition of n of size $n \quad j$, we must add exactly $n \quad j$ zeros. There are exactly j positions to insert the zeros in the partition (Definition 4.3 disallows an insertion at the start of the vector, all other locations are valid), this can be done in $\binom{n-1}{n-j-1}$ ways, giving the second bound. □

Corollary 5.1.1. *Evaluating Equation (40) has complexity $O(C_e(n;s))$ where*

$$C_e(n;s) := j \quad f0;1g^n j\, p(n;s)\binom{n \quad 1}{s \quad 1}:$$

We find it interesting to compare this to the cost of the recursive evaluation, which has cost $C_r(n;s) := 2^{n-s}\frac{n!}{s!}$ at a single point [1]. Our bound, however, is dependent on the structure of the box spline and not just the number of direction vectors n. This is convenient in deriving tighter bounds for specific , but makes analysis difficult for general n. We can, however, create a more conservative general bound by noting that the worst case size of $j \quad f0;1g^n j$ is 2^n, thus we have the looser bound

$$\bar{C}_e(n;s) := 2^n p(n;s)\binom{n \quad 1}{s \quad 1}$$

with $C_e(n;s) \quad \bar{C}_e(n;s)$*:* We have calculated this bound and presented it for $s = 2$ and 3 in Table 1 and Table 2 respectively. It is important to note that this is a bound on the number of operations required and not a strict cost, as in the case of the recursive form. In practice we have found that there are often less elements in the sets P_{n-s} and S but this is highly dependent on the choice of . In the same vein, it is also likely possible to use memoization techniques to slightly reduce the cost of recursive evaluation, however, this is already somehow "pre-baked" into our method — no memoization is needed.

The bound $\bar{C}(n;s)$ has behaviour similar to that of Pascal's triangle — with n fixed, as s increases C_e increases, but when s passes a threshold, C_e decreases. The behaviour of this bound is slightly skewed and is shown in Figure 1.

n	2	3	4	5	6	7	8	9	10
$\tilde{C}_e(n;2)$	4	16	96	256	960	2,304	7,168	16,384	46,080
$C_r(n;2)$	1	6	48	480	5,760	80,640	1,290,240	23,224,320	464,486,400

Table 1: A comparison between the bound on the number of operations required to evaluate Equation 40 when $s = 2$ in the first row, and the number of operations required for the recursive evaluation, the second row.

n	3	4	5	6	7	8	9	10
$\tilde{C}_e(n;3)$	8	48	384	1,920	7,680	26,880	100,352	294,912
$C_r(n;3)$	1	8	80	960	13,440	215,040	3,870,720	77,414,400

Table 2: A comparison between the bound on the number of operations required to evaluate Equation 40 when $s = 3$ in the first row, and the number of operations required for the recursive evaluation, the second row. Notice that as s grows, the bound $\tilde{C}_e$ grows.

n = 1:						2					
n = 2:					4		4				
n = 3:				8		16		8			
n = 4:			16		96		48		16		
n = 5:		32		256		384		128		32	
n = 6:	64		960		1920		1280		320		64

Figure 1: The triangle of $C(\tilde{n}, s)$. The value of s varies from 1 to n as the values move left to right. The maximal values are found roughly in the center of the rows.

6. Fast Evaluation

While it is possible to use Equation (40) as a stable form of evaluation, we prefer a different approach. We prefer to modify Equation (37) to yield an explicit polynomial within each region of the box spline mesh. Similar to the above section, we may obtain the polynomial piece for some region R with the formula

$$P_R(x) := \sum_{(b;p)\in S} \sum_{(c;\ )\in P_{n\ s}} \frac{b\quad c}{j\det\quad j} [\![\ {}^{-1}c_R \quad p]\!]_+^0 [\![\ {}^{-1}x \quad p]\!]^{(\quad {}^{-1})} \tag{43}$$

where we choose c_R as the center of the region R, but it could be any interior point of R. To see why this is correct, we take any $x\ 2\ R^o$ and notice that if $[\![\ {}^{-1}x \quad p]\!]_+^0 = 1$ then we know that the polynomial $[\![\ {}^{-1}x \quad p]\!]^{(\quad {}^{-1})}$ contributes to the polynomial in region R. We sum all of these in Equation (43), which yields the polynomial for this region.

We note that it is also possible to obtain the explicit piecewise polynomial regions via a slight modification to the recursive evaluation scheme. This however, is also done in $O(2^{n-s}\frac{n!}{s!})$, per region which is quite expensive. Once the sets P_{n-s} and S have been computed, the cost of computing $P_R(x)$ for one region is $O(C_e(n;s))$: There is also a cost to computing P_{n-s} and S , which can be obtained via summing the magnitude of each set in each recurrence. This gives the following result

Proposition 6.1. *The sets P_{n-s} and S can be computed in*

$$O\left(p(n)\binom{n\quad 1}{b(n\quad 1)=2c}\right) \text{ and } O(nj\quad f0;1g^n j)$$

operations respectively, where $p(n)$ is the total number of partitions of the integer n.

The procedure for a fast and stable evaluation scheme can be outlined as follows. In a pre-computation step, compute the sets P_{n-s} and S . Next, compute each region of the box spline by slicing the support of the box spline by the knot-planes of the mesh. Now that we know which regions contain distinct polynomials, for each region compute the polynomial within that region using Equation (43). Finally, build a binary space partitioning (BSP) tree that uses the knot planes as internal nodes to partition space into each region. At evaluation time, one simply traverses this tree and uses the stored polynomial at a leaf to compute the value of the box spline. If k is the number of regions, this happens in $O(\log(k)\binom{d+s}{s})$ time, where d is the degree of the polynomial pieces of the box spline, assuming the most naïve evaluation scheme for a polynomial.

The precomputation step is outlined in Algorithm (1) and the evaluation at a point is outlined in Algorithm (2). In the following algorithms there are two object classes, $InternalNode()$ and $LeafNode()$ that inherit from a base $Node()$ class. Additionally, whenever we speak of some hyperplane h, we speak of it as the normal equation $h(x) = n \quad x \quad d$ for whatever vector n and d define that plane.

It is important to note that the choice of h in Algorithm 1 will affect the performance of evaluation; h should be chosen so as to balance the resulting binary tree. As a heuristic, we choose h

Algorithm 1 Precompute box spline pieces

```
1: procedure PRECOMPUTEBOXSPLINE( )
2:     /* Returns a tuple containing the polyhedron of the support of
3:     as the first element and the root Node object as the second*/
4:     Compute S  from Definition 3.1
5:     Compute P_{n-s} from Definition 4.3
6:     Calculate the polyhedron Q for supp(M )
7:     Compute knot-planes H s.t. if h 2 H then h \ supp(M )
8:     return (Q; Recurse(H; Q))
9: procedure RECURSE(H, Q)
10:    /* Returns a Node object */
11:    Choose h 2 H that splits Q
12:    if no such h exists then
13:        Then Q is a distinct region of the mesh, so use
14:        Equation (43) to derive a polynomial piece P_Q(x)
15:        return LeafNode(Q; P_Q(x))
16:    Split Q with h into left and right regions A; B
17:    /* Here, left means all points x that satisfy h(x)    0, right is the opposite */
18:    node    InternalNode()
19:    node:h    h
20:    node:left    Recurse(Hnfhg; A)
21:    node:right    Recurse(Hnfhg; B)
22:    return node
```

Algorithm 2 Evaluate box spline

```
1: procedure EVALBOXSPLINE(x, Q, root)
2:     if x 2 Q then
3:         return 0
4:     return RecurseEval(x; root)
5: procedure RECURSEEVAL(x, node)
6:     if node is a LeafNode then
7:         return evaluate node's polynomial at x
8:     h    node:h
9:     L    node:left
10:    R    node:right
11:    if h(x)    0 then
12:        return Recurse(x; L)
13:    return Recurse(x; R)
```

so that it splits the given polyhedron into polyhedra of roughly equivalent hyper-volumes. To do this, we iterate over each $h \ 2 \ H$. At each iteration we sum the volumes of the regions on the left of h, and sum the volumes of the regions on the right. We choose h that minimizes the difference between these two sums. On average, the traversal time is $O(log(k))$ where k is the total number of regions of the box spline. A variant on this algorithm is implemented in a SageMath [10] worksheet and is available as supplementary material.

7. Examples

Here we enumerate some examples of box splines. When feasible, we enumerate their polynomial regions, and P and S sets. When we enumerate the sets for these box splines, we enumerate simplified versions of the sets, combining elements that correspond to like terms in the final summation.

Example 7.1. *Courant Element:* The first example we consider is the well known Courant element which is constructed by taking the the two Cartesian principle lattice directions and adding the diagonal vector. This is given by the direction matrix

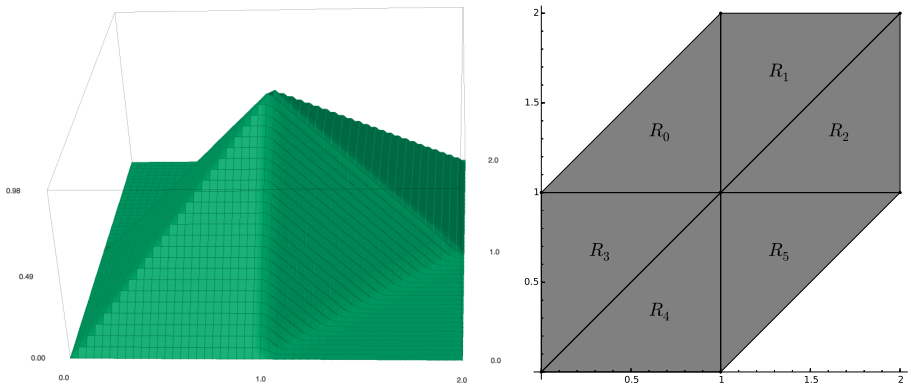

Figure 2: On the left is a plot of the Courant element, on the right is a plot of the different polynomial regions of the box spline.

Enumerating the sets for this direction matrix yields

$$S = f(1;(1;2));(\ 1;(0;1));(1;(0;0));(1;(2;1));(\ 1;(2;2));(\ 1;(1;0))g;$$
$$P_1 = f(1;(2;1;0));(\ 1;(2;0;1))g:$$

Example 7.2. *Zwart-Powell Element:* The Zwart-Powell element is similar to the Courant element but has an additional vector added along a complementary diagonal. We provide a more thorough example for the construction of P_2 for the ZP element. We first start by noting the ZP element's direction matrix and a basis for its null-space

$$= \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}; \ker = \begin{bmatrix} 1 & 0 \\ 1 & 2 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}: \tag{44}$$

We first construct P_1 by definition (4.3). Consider $(c; \ ) = (1; (1; 1; 1; 1)) \ 2 \ P_0$. Since this is the base case in definition (4.1), for this we have

$$= (1; 1; \ 1; 0); \text{ and } m = 1:$$

Definition (4.3) also dictates that this produces elements for all j with $m < j \ n$ such that $(j) \not< 0$. For $j = 2$ and $j = 3$ we have the tuples

$$\left(1\frac{1}{1}; (2; 0; 1; 1) \right) \text{ and } \left(1\frac{1}{1}; (2; 1; 0; 1) \right)$$

respectively. Thus we have $P_1 = f(\ 1; (2; 0; 1; 1)); (1; (2; 1; 0; 1)g$, and we apply the recursion to each tuple of this multiset to construct P_2. For $(c; \ ) = (\ 1; (2; 0; 1; 1)) \ 2 \ P_1$ we construct

$$C = \begin{bmatrix} 1 & 2 \end{bmatrix} \text{ which has } \ker C = \begin{bmatrix} 2 \\ 1 \end{bmatrix}:$$

Thus $t = (2; \ 1)$, and we have

$$= \ker \ t = (2; 0; \ 1; 1); \text{ and } m = 1;$$

which produces the elements

$$\left((\ 1)\frac{1}{2}; (3; 0; 0; 1) \right) \text{ and } \left(1\frac{1}{2}; (3; 0; 1; 0) \right)$$

for $j = 3$ and $j = 4$ respectively. Finally, we look at the case $(c; \ ) = (1; (2; 1; 0; 1)) \ 2 \ P_1$. Again, we construct

$$C = \begin{bmatrix} 1 & 1 \end{bmatrix} \text{ which has } \ker C = \begin{bmatrix} 1 \\ 1 \end{bmatrix}:$$

Which gives $t = (1; \ 1)$, and we have

$$= \ker \ t = (1; \ 1; 0; 1); \text{ and } m = 1;$$

producing the tuples

$$\left(1\frac{1}{1}; (3; 0; 0; 1) \right) \text{ and } \left(1\frac{1}{1}; (3; 1; 0; 0) \right):$$

So we have

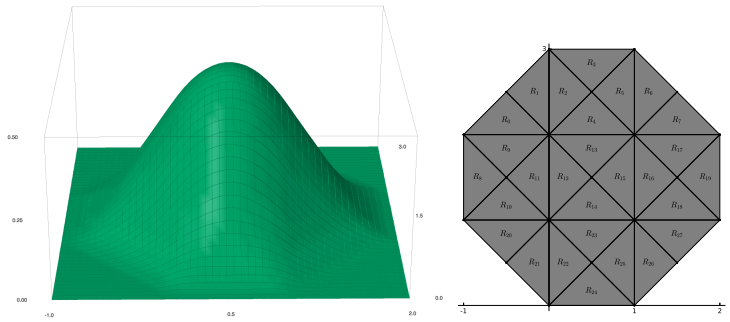

Figure 3: On the left is a plot of the ZP-element, on the right is a plot of the different polynomial regions of the box spline.

$$P_2 = f(\ \ 1=2;(3;0;0;1))\ ;(1=2;(3;0;1;0))\ ;(1;(3;0;0;1))\ ;(\ \ 1;(3;1;0;0))g$$

or, equivalently (by collecting like terms)

$$P_2 = f(1=2;(3;0;0;1))\ ;(1=2;(3;0;1;0))\ ;(\ \ 1;(3;1;0;0))g:$$

Applying the recursion in definition (3.1) also yields the set

$$\begin{aligned} S \quad = \quad & f(\ \ 1;(\ \ 1=2;3=2))\ ;(1;(\ \ 1=2;\ \ 3=2))\ ;(1;(\ \ 3=2;1=2))\ ;(1;(1=2;3=2))\ ; \\ & (\ \ 1;(1=2;\ \ 3=2))\ ;(\ \ 1;(\ \ 3=2;\ \ 1=2))\ ;(1;(3=2;\ \ 1=2))\ ;(\ \ 1;(3=2;1=2))g \end{aligned}$$

Using all of this applied to Equation (37) yields

$$
\begin{aligned}
M\ (x) \;=\;& \left[\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 3{=}2;\ \ 1{=}2)\right]\right]_+^{(3;1)} \quad \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 3{=}2;\ \ 1{=}2)\right]\right]_+^{(3;1)} \\
& \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 3{=}2;\ \ 1{=}2)\right]\right]_+^{(3;1)} \quad 1\left[\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}^{-1} x \quad (1{=}2;3{=}2)\right]\right]_+^{(3;1)} \\
+\ & \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (1{=}2;3{=}2)\right]\right]_+^{(3;1)} + \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (1{=}2;3{=}2)\right]\right]_+^{(3;1)} \\
& 1\left[\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 3{=}2;1{=}2)\right]\right]_+^{(3;1)} + \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 3{=}2;1{=}2)\right]\right]_+^{(3;1)} \\
+\ & \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 3{=}2;1{=}2)\right]\right]_+^{(3;1)} \quad 1\left[\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}^{-1} x \quad (3{=}2;\ \ 1{=}2)\right]\right]_+^{(3;1)} \\
+\ & \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (3{=}2;\ \ 1{=}2)\right]\right]_+^{(3;1)} + \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (3{=}2;\ \ 1{=}2)\right]\right]_+^{(3;1)} \\
& 1\left[\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 1{=}2;\ \ 3{=}2)\right]\right]_+^{(3;1)} + \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 1{=}2;\ \ 3{=}2)\right]\right]_+^{(3;1)} \\
+\ & \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 1{=}2;\ \ 3{=}2)\right]\right]_+^{(3;1)} + \left[\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}^{-1} x \quad (3{=}2;1{=}2)\right]\right]_+^{(3;1)} \\
& \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (3{=}2;1{=}2)\right]\right]_+^{(3;1)} \quad \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (3{=}2;1{=}2)\right]\right]_+^{(3;1)} \\
+\ & 1\left[\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}^{-1} x \quad (1{=}2;\ \ 3{=}2)\right]\right]_+^{(3;1)} \quad \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (1{=}2;\ \ 3{=}2)\right]\right]_+^{(3;1)} \\
& \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (1{=}2;\ \ 3{=}2)\right]\right]_+^{(3;1)} + \left[\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 1{=}2;3{=}2)\right]\right]_+^{(3;1)} \\
& \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 1{=}2;3{=}2)\right]\right]_+^{(3;1)} \quad \frac{1}{2}\left[\left[\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}^{-1} x \quad (\ 1{=}2;3{=}2)\right]\right]_+^{(3;1)} :
\end{aligned}
$$

Example 7.3. *Skewed Element:* As an example of a box spline that has not appeared in the literature, we introduce the four direction skewed element, defined by the matrix

$$= \begin{bmatrix}1 & 0 & 1 & 2\\ 0 & 1 & 1 & 1\end{bmatrix}. \tag{45}$$

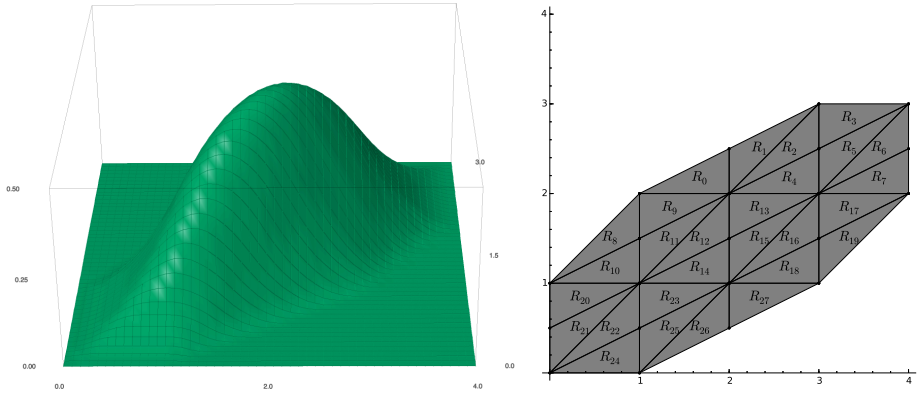

Figure 4: On the left is a plot of the Skewed element, on the right is a plot of the different polynomial regions of the box spline.

$$
\begin{aligned}
S &= f(1;(2;3=2));(\ 1;(2;1=2));(\ 1;(\ 2;\ 1=2));(1;(\ 2;\ 3=2));\\
&\quad (1;(\ 1;1=2));(1;(1;\ 1=2));(\ 1;(1;3=2));(\ 1;(\ 1;\ 3=2))g\\
P_2 &= f(1=2;(3;0;0;1));(\ 1;(3;0;1;0));(\ 1=2;(3;1;0;0))g
\end{aligned}
$$

Example 7.4. *FCC Cubic Spline:* Finally, as an example of a higher dimensional box spline, we enumerate the sets for the FCC Cubic Spline of Kim et al [4].

$$
= \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{46}
$$

$$
\begin{aligned}
S &= f(1;(\ 1=2;3=2;\ 3=2));(\ 1;(1=2;\ 3=2;\ 1=2));(\ 1;(3=2;\ 3=2;1=2));\\
&\quad (\ 1;(\ 1=2;3=2;1=2));(\ 1;(\ 3=2;1=2;3=2));(\ 1;(\ 3=2;3=2;\ 1=2));\\
&\quad (\ 1;(\ 1=2;1=2;\ 3=2));(1;(3=2;\ 1=2;1=2));(1;(\ 3=2;\ 1=2;3=2));\\
&\quad (\ 1;(1=2;3=2;\ 3=2));(\ 1;(3=2;\ 1=2;\ 3=2));(\ 1;(1=2;\ 1=2;3=2));\\
&\quad (1;(1=2;\ 1=2;\ 3=2));(1;(\ 1=2;1=2;3=2));(1;(1=2;\ 3=2;3=2));\\
&\quad (\ 1;(\ 3=2;\ 1=2;1=2));(1;(1=2;3=2;\ 1=2));(\ 1;(\ 1=2;\ 3=2;3=2));\\
&\quad (1;(\ 3=2;1=2;\ 1=2));(1;(\ 3=2;3=2;1=2));(1;(\ 1=2;\ 3=2;1=2));\\
&\quad (1;(3=2;1=2;\ 3=2));(1;(3=2;\ 3=2;\ 1=2));(\ 1;(3=2;1=2;\ 1=2))g\\
P_3 &= f(\ 1;(4;1;1;0;0;0));(\ 1;(4;1;0;0;1;0));(\ 1;(4;0;1;1;0;0));\\
&\quad (\ 1;(3;2;1;0;0;0));(1;(3;2;0;0;1;0));(\ 1;(4;0;0;0;1;1));\\
&\quad (1;(4;1;0;0;0;1));(\ 1;(4;1;0;1;0;0));(\ 1;(3;2;0;0;0;1));\\
&\quad (\ 1;(3;2;0;1;0;0))g:
\end{aligned}
$$

As this box spline has many regions, we omit the enumeration of polynomial pieces. The following figure shows the numerical stability of this form by plotting several iso-contours of this box spline.

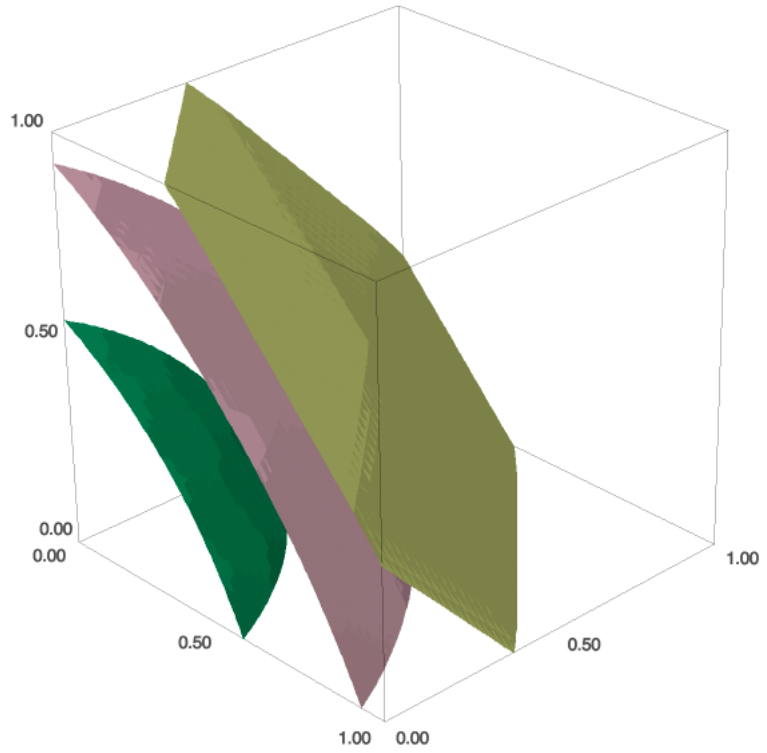

Figure 5: Iso-surfaces for the level-sets 1/4, 1/16, 1/256, pictured in green, pink and yellow respectively.

8. Conclusion

In this work, we provided an explicit decomposition scheme for the piecewise polynomial form of any non-degenerate box spline and provided bounds on the complexity of this procedure. While there do exist other evaluation schemes for evaluating box splines, they all typically depend on recursively evaluating a box spline, or they are ad-hoc in their construction procedure. Our method is general and works for all non-degenerate box-splines with real valued direction matrices. However, we only provide a characterization of the splines themselves, when used in an approximation space, it is possible to derive more efficient evaluation schemes for the semi-discreet convolution sum between a box-spline and lattice data. This is an avenue for future work.

We also provide, as a SageMath [10] worksheet, the code to derive the piecewise polynomial form of any box-spline, as well as the code to generate BSP trees that represent the evaluation schemes, and code to generate C++ code from those BSPs.

Acknowledgements

We would like to thank the reviewers for their helpful comments, suggestions and corrections. We would also like to thank NSERC for providing funding for this work.

[1] de Boor, C.: On the evaluation of box splines. Numerical Algorithms **5**(1), 5–23 (1993). DOI 10.1007/BF02109280

[2] de Boor, C., Höllig, K., Riemenschneider, S.: Box Splines. Springer-Verlag New York, Inc., New York, NY, USA (1993). DOI 10.1007/978-1-4757-2244-4

[3] Entezari, A., Van De Ville, D., Moller, T.: Practical box splines for reconstruction on the Body Centered Cubic lattice. Visualization and Computer Graphics, IEEE Transactions on **14**(2), 313–328 (2008). DOI 10.1109/TVCG.2007.70429

[4] Kim, M., Entezari, A., Peters, J.: Box spline reconstruction on the Face-Centered Cubic lattice. Visualization and Computer Graphics, IEEE Transactions on **14**(6), 1523–1530 (2008)

[5] Kim, M., Peters, J.: Fast and stable evaluation of box-splines via the BB-form. Numerical Algorithms **50**(4), 381–399 (2009). DOI 10.1007/s11075-008-9231-6

[6] Kim, M., Peters, J.: Symmetric box-splines on the A*N lattice. Journal of Approximation Theory **162**(9), 1607–1630 (2010). DOI 10.1016/j.jat.2010.04.007

[7] Kobbelt, L.: Stable evaluation of box splines. Numerical Algorithms **14**(4), 377–382 (1997). DOI 10.1023/A:1019133501773

[8] Peña, J.M.: On the multivariate Horner scheme. SIAM Journal on Numerical Analysis **37**(4), 1186–1197 (2000)

[9] Peña, J.M., Sauer, T.: On the multivariate Horner scheme II: Running error analysis. Computing **65**(4), 313–322 (2000)

[10] William, A.S., et al.: Sage Mathematics Software (Version 8.0.0) (2017). `http://www.sagemath.org`

Region	Polynomial Piece
R_0	$x \quad y+1$
R_1	$y+2$
R_2	$x+2$
R_3	x
R_4	y
R_5	$x+y+1$

Table 3: Polynomial regions of the Courant element.

Region	Polynomial Piece
$R_0; R_1$	$\frac{1}{4}x^2 \quad \frac{1}{2}xy+\frac{1}{4}y^2+\frac{3}{2}x \quad \frac{3}{2}y+\frac{9}{4}$
R_2	$\frac{1}{4}x^2 \quad \frac{1}{2}xy+\frac{1}{4}y^2+\frac{3}{2}x \quad \frac{3}{2}y+\frac{9}{4}$
R_3	$\frac{1}{2}y^2 \quad 3y+\frac{9}{2}$
R_4	$\frac{1}{2}x^2+\frac{1}{2}x \quad \frac{1}{2}y+\frac{5}{4}$
R_5	$\frac{1}{4}x^2+\frac{1}{2}xy+\frac{1}{4}y^2 \quad x \quad 2y+\frac{7}{2}$
$R_6; R_7$	$\frac{1}{4}x^2+\frac{1}{2}xy+\frac{1}{4}y^2 \quad 2x \quad 2y+4$
R_8	$\frac{1}{2}x^2+x+\frac{1}{2}$
R_9	$\frac{1}{4}x^2 \quad \frac{1}{2}xy \quad \frac{1}{4}y^2+\frac{3}{2}x+\frac{1}{2}y+\frac{1}{4}$
R_{10}	$\frac{1}{4}x^2+\frac{1}{2}xy \quad \frac{1}{4}y^2+y \quad \frac{1}{2}$
R_{11}	$\frac{1}{2}y^2+\frac{1}{2}x+\frac{3}{2}y \quad \frac{3}{4}$
$R_{12}; R_{13}; R_{14}; R_{15}$	$\frac{1}{2}x^2 \quad \frac{1}{2}y^2+\frac{1}{2}x+\frac{3}{2}y \quad \frac{3}{4}$
R_{16}	$\frac{1}{2}y^2 \quad \frac{1}{2}x+\frac{3}{2}y \quad \frac{1}{4}$
R_{17}	$\frac{1}{4}x^2+\frac{1}{2}xy \quad \frac{1}{4}y^2 \quad 2x+2$
R_{18}	$\frac{1}{4}x^2 \quad \frac{1}{2}xy \quad \frac{1}{4}y^2 \quad \frac{1}{2}x+\frac{3}{2}y \quad \frac{1}{4}$
R_{19}	$\frac{1}{2}x^2 \quad 2x+2$
$R_{20}; R_{21}$	$\frac{1}{4}x^2+\frac{1}{2}xy+\frac{1}{4}y^2$
R_{22}	$\frac{1}{4}x^2+\frac{1}{2}xy+\frac{1}{4}y^2$
R_{23}	$\frac{1}{2}x^2+\frac{1}{2}x+\frac{1}{2}y \quad \frac{1}{4}$
R_{24}	$\frac{1}{2}y^2$
R_{25}	$\frac{1}{4}x^2 \quad \frac{1}{2}xy+\frac{1}{4}y^2+\frac{1}{2}x+\frac{1}{2}y \quad \frac{1}{4}$
$R_{26}; R_{27}$	$\frac{1}{4}x^2 \quad \frac{1}{2}xy+\frac{1}{4}y^2 \quad \frac{1}{2}x+\frac{1}{2}y+\frac{1}{4}$

Table 4: Polynomial regions of the ZP-element.

Region	Polynomial Piece
R_0	$\frac{1}{2}x^2 \quad xy + \frac{1}{2}y^2 + 2x \quad 2y + 2$
R_1	$\frac{1}{2}x^2 + \frac{1}{4}y^2 + 2x \quad 2y + 2$
$R_2; R_3$	$\frac{1}{4}y^2 \quad 2y + 4$
R_4	$\frac{1}{2}x^2 + xy \quad \frac{1}{4}y^2 \quad x \quad y + \frac{7}{2}$
R_5	$\frac{1}{2}x^2 \quad 3x + \frac{9}{2}$
R_6	$x^2 \quad xy + \frac{1}{4}y^2 + x \quad \frac{1}{2}y + \frac{1}{4}$
R_7	$\frac{1}{2}x^2 \quad xy + \frac{1}{4}y^2 + 2x \quad \frac{1}{2}y \quad \frac{1}{4}$
R_8	$\frac{1}{2}x^2 + 2x \quad \frac{1}{2}y \quad \frac{1}{4}$
$R_9; R_{10}$	$x^2 + xy \quad \frac{1}{2}y^2 + x + \frac{1}{2}y \quad \frac{3}{4}$
R_{11}	$\frac{1}{2}x^2 + xy \quad \frac{1}{2}y^2 \quad x + \frac{1}{2}y + \frac{5}{4}$
R_{12}	$\frac{1}{2}x^2 \quad \frac{1}{4}y^2 \quad 3x + \frac{3}{2}y + \frac{9}{4}$
R_{13}	$x^2 \quad xy + \frac{1}{4}y^2 \quad 3x + \frac{3}{2}y + \frac{9}{4}$
R_{14}	$x^2 \quad xy + \frac{1}{4}y^2 + x \quad \frac{1}{2}y + \frac{1}{4}$
R_{15}	$\frac{1}{2}x^2 \quad \frac{1}{4}y^2 + \frac{1}{2}y \quad \frac{1}{4}$
R_{16}	$\frac{1}{2}x^2 + xy \quad \frac{1}{2}y^2 + \frac{1}{2}y \quad \frac{1}{4}$
$R_{17}; R_{18}$	$x^2 + xy \quad \frac{1}{2}y^2 + x + \frac{1}{2}y \quad \frac{3}{4}$
R_{19}	$\frac{1}{2}x^2 + x + \frac{1}{2}y \quad \frac{3}{4}$
R_{20}	$\frac{1}{2}x^2 \quad xy + \frac{1}{4}y^2 \quad x + \frac{3}{2}y + \frac{1}{4}$
R_{21}	$x^2 \quad xy + \frac{1}{4}y^2 \quad 3x + \frac{3}{2}y + \frac{9}{4}$
R_{22}	$\frac{1}{2}x^2$
R_{23}	$\frac{1}{2}x^2 + xy \quad \frac{1}{4}y^2$
$R_{24}; R_{25}$	$\frac{1}{4}y^2$
R_{26}	$\frac{1}{2}x^2 + \frac{1}{4}y^2 + x \quad \frac{1}{2}$
R_{27}	$\frac{1}{2}x^2 \quad xy + \frac{1}{2}y^2 \quad x + y + \frac{1}{2}$

Table 5: Polynomial regions of the Skewed element.