Evaluation of the Linear Box-Spline Filter from Trilinear Texture Samples: A Feasibility Study

Balázs Domonkos
Mediso Medical Imaging Systems, Hungary
balazs.domonkos@mediso.hu

Balázs Csébfalvi
Budapest University of Technology and Economics
cseb@iit.bme.hu

ABSTRACT

The major preference for applying B-spline filtering rather than non-separable box spline filtering on the BCC lattice is the fact that separable filtering can be performed more efficiently on current GPUs due to the utilization of the hardware-accelerated trilinear texture fetching. In order to make a fair comparison, a similar, efficient evaluation scheme is required that uses trilinear texture fetches instead of nearest-neighbor ones also for the box splines. Thus, in this paper, we propose an evaluation scheme for the linear BCC box spline built upon a trilinear B-spline basis. We compare our trilinearly evaluated linear box spline scheme to the latest method, that uses twice as many nearest neighbor fetches. Then we give a comparison to the major competitive methods: the BCC B-spline filtering and the BCC DC-spline filtering in terms of their performance.

Keywords: Volume Rendering, Filtering, Reconstruction.

1 INTRODUCTION

In many applications in engineering and computing science, a continuous phenomenon is represented by its discrete samples. In order to operate on the underlying continuous function, first it has to be accurately reconstructed from its discrete representation. Reconstruction filters have received attention also in image processing and volume visualization since appropriate reconstruction of multivariate functions is a key step of the processing pipeline [2, 3, 17, 18].

According to the most commonly-used sampling scheme in practice, volumetric data is often acquired on a uniform lattice by regular sampling, while reconstruction is performed by convolution filtering. An appropriate choice of both the sampling lattice and the reconstruction filter kernel is of crucial importance as they together directly determine the quality of the reproduced continuous function and the efficiency of the reconstruction.

Recent results advocate the benefits of non-Cartesian lattices for regular sampling. The application of Body-Centered Cubic (BCC) sampling received increased attention from the perspective of continuous signal reconstruction in the last decade [5, 6, 11, 12]. This lattice is optimal for sampling 3D signals of isotropic bandwidth [19, 21], unlike the commonly used Cartesian Cubic (CC) lattice along with tensor-product reconstruction. To perfectly reconstruct a signal of a spherically bounded spectrum from its discrete representation, roughly 30% fewer samples per unit volume have to be taken on a BCC lattice than on an equivalent CC lattice. In addition to the improved spectral isotropy, this directly translates into an explicit reduction of the storage cost.

A crucial question of BCC sampling is the way in which the original continuous signal is reconstructed from its discrete samples. Although due to the shift-invariant property of the sampling lattice, the reconstruction can be implemented by a simple convolution, the choice of the filter kernel has a direct impact on both numerical accuracy and visual quality. Generally, an appropriate filter is chosen by making a compromise between quality and efficiency.

Currently, three promising resampling techniques exist for the BCC lattice that provide high visual quality, numerical accuracy, and efficiency at the same time: the box splines [11], the BCC B-splines [8, 6], and the BCC DC-splines [10]. As only the latter two methods can exploit the hardware-accelerated trilinear filtering, it has not been possible to make a fair comparison so far. To remedy this problem, we propose an algorithm that uses trilinear fetches for the box spline filtering as well. Since these filters have already been compared in terms of visual quality and numerical accuracy [6, 10, 13], in this paper, we focus on a fair comparison of their performance.

2 RELATED WORK

One of the most important aspects of rendering sampled data is how to perform proper and efficient resampling depending on the applied lattice. For the CC lattice, reconstruction filters are usually designed in 1D, and then

Plzen, Czech Republic.

extended to the trivariate setting by a separable tensor-product extension. However, the BCC lattice is not separable itself, therefore the advantageous properties of a 1D filter are not necessarily inherited in 3D by a separable extension [21, 22, 15].

The first reconstruction filters tailored to the geometry of the non-Cartesian lattices were proposed by Entezari et al. [11]. They applied *box splines*, that offer a mathematically elegant toolbox for constructing a class of multidimensional elements with flexible shape and support. Box splines are often considered as a generalization of B-splines to multivariate setting. Theoretically, the computational complexity of a box spline is lower than that of an equivalent B-spline, since its support is more compact and its total polynomial degree is lower. To investigate this potential also in practice, several attempts were made. Although de Boor's recurrence relation [9] is the most commonly used technique for evaluating box splines at an arbitrary position, it is computationally inefficient and has numerical instabilities [14]. Addressing this issue, Entezari et al. [12] derived a piecewise-polynomial representation of the linear and quintic box splines for the BCC lattice. In a CPU-based implementation, due to the smaller support of the box spline kernels, the data access cost of discrete BCC samples turned out to be twice as low as for the equivalent B-spline filters on the CC lattice [12]. Following their work, Finkbeiner et al. proposed an algorithm to convolve the BCC samples with these box spline kernels [13]. Though they applied early selection of polynomial segments of the piecewise polynomial form that enabled them to avoid a full kernel evaluation for each affected sample point, the theoretical advantages of box splines could not be exploited on the GPUs, which are rather optimized for separable filtering.

Another family of non-separable filters is represented by the *Voronoi splines* [16] that inherit the geometry of a sampling lattice through its Voronoi cell. For Cartesian lattices, Voronoi splines coincide with tensor-product B-splines. For the 2D hexagonal lattice, Voronoi splines were originally proposed by Van de Ville et al. [23] as Hex-splines. For the BCC lattice Voronoi splines were derived as BCC-splines by Csébfalvi [5]. Recently, Mirzargar et al. [16] formulated the BCC-splines in terms of multi-box splines. In spite of their theoretical elegance, Voronoi splines are currently impractical, since their piecewise evaluation is not known yet.

Csébfalvi recommended a *prefiltered Gaussian reconstruction scheme* [4] adapting the principle of generalized interpolation [1] to the BCC lattice. According to this approach, first a non-separable discrete prefiltering is performed as a preprocessing step, and afterwards a fast separable Gaussian filtering is used for continuous resampling on the fly. This method was extended also to the *B-spline family of filters* [8]. An efficient GPU implementation was proposed exploiting the fact that the BCC lattice consists of two interleaved CC lattices, where the second CC lattice is translated by half of the grid spacing. The reconstruction can be performed separately for these two CC lattices in the given sample position by using a standard trilinear or tricubic B-spline resampling, and then the contributions are averaged [8]. BCC B-splines reconstruction was reported to be four to five times faster on an NVIDIA GeForce 6800 graphics card than a non-separable box spline reconstruction of the same approximation power [6], since the B-splines can utilize the hardware-accelerated trilinear texture fetching [20].

Recently, Domonkos et al. [10] proposed a *discrete/-continuous filter family* generated by the impulse response of the BCC trilinear kernel. This technique is theoretically equivalent to the discrete upsampling of the BCC-sampled volume on a higher resolution CC lattice, where the standard trilinear interpolation is used for resampling. In practice, however, the missing CC samples are calculated on the fly and not in a preprocessing. Using an optimized GPU implementation, the linear DC-spline was reported to be slightly faster than the linear box spline.

3 SPLINE RECONSTRUCTION FOR THE BCC LATTICE

In the following, we briefly review the main properties of the BCC lattice, as well as the box spline, B-spline, and DC-spline family of filters, as they are applied for reconstruction on the BCC lattice.

3.1 BCC Lattice

The BCC lattice Λ_{BCC} is a discrete subgroup of $\mathbb{R}^3$ generated by integer linear combinations of the following basis vectors:

$$\begin{aligned} \Xi_{BCC} &= [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3] = \frac{1}{2}\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} \\ \Lambda_{BCC} &= \left\{ \Xi_{BCC}\mathbf{i} \,:\, \mathbf{i} \in \mathbb{Z}^3 \right\} \subset \mathbb{R}^3. \end{aligned}$$

Besides, the BCC lattice points are located on a CC lattice with an additional sample placed in the center of each cube. Thus, the BCC lattice can also be considered as two interleaved CC lattices Λ_{CC_A} and Λ_{CC_B}. By shifting the secondary CC lattice Λ_{CC_B} by half of the grid spacing, the vertices of the secondary CC lattice are moved to the centers of the primary CC cells:

$$\begin{aligned} \Lambda_{BCC} &= \Lambda_{CC_A} \cup \Lambda_{CC_B} \\ \Lambda_{CC_A} &= \left\{ \mathbf{i} \,:\, \mathbf{i} \in \mathbb{Z}^3 \right\} \\ \Lambda_{CC_B} &= \left\{ \mathbf{i} + \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} \,:\, \mathbf{i} \in \mathbb{Z}^3 \right\}. \end{aligned} \tag{1}$$

On the other hand, the BCC lattice can be obtained also from a dense CC lattice by keeping only the lattice points whose coordinates have identical parity:

$$\Lambda_{BCC} = \left\{ \frac{1}{2} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \quad : \; i \equiv j \equiv k \pmod 2 \atop i,j,k \in \mathbb{Z} \right\}. \quad (2)$$

3.2 Box Splines for the BCC Lattice

A box spline M_{Ξ} is the shadow of a unit-hypercube in $\mathbb{R}^n$ projected to $\mathbb{R}^s, s \leq n$ where the projection is characterized by $\Xi = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_n] \in \mathbb{R}^{s \times n}$, $\boldsymbol{\xi}_i \in \mathbb{R}^s \backslash \mathbf{0}$ [9]. The shape, the continuity order, and the approximation power of a given box spline M_{Ξ} is determined by Ξ. The simplest box spline is constructed when $s = n$ as a normalized characteristic function of its support:

$$M_{\Xi}(\mathbf{x}) = \begin{cases} \frac{1}{\det \Xi} & \text{if } \Xi^{-1}\mathbf{x} \in [0,1)^n \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

When adding a further direction vector $\boldsymbol{\xi} \in \mathbb{R}^s$ to Ξ, $s < n$, the box spline $M_{[\Xi,\boldsymbol{\xi}]}$ is given by the convolution:

$$M_{[\Xi,\boldsymbol{\xi}]}(\mathbf{x}) = \int_0^1 M_{\Xi}(\mathbf{x} - t\boldsymbol{\xi})\, dt. \quad (4)$$

The linear box spline $M_{\Xi^1_{BCC}} \in C^0$ for the BCC lattice is constructed as a 3D shadow of a tesseract along its antipodal axis, resulting a function with a rhombic dodecahedron support, which is the first neighbors cell of the BCC lattice [12]:

$$\Xi^1_{BCC} = \begin{bmatrix} & & 1/2 \\ \Xi_{BCC} & & 1/2 \\ & & 1/2 \end{bmatrix}. \quad (5)$$

$M_{\Xi^1_{BCC}}$ has its maximum value at the center, and has a linear falloff towards the 14 first-neighbor vertices:

$$M_{\Xi^1_{BCC}}(\mathbf{x}) = \max(1 - x - y,\, 0), \quad (6)$$

where x is the largest and y is the second largest component of the absolute coordinates of $\mathbf{x}$ [12].

3.3 B-Splines for the BCC Lattice

The B-spline of order zero is defined as a box filter:

$$\beta^0(t) = \begin{cases} 1 & \text{if } |t| < \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Generally, the B-spline filter of order n is derived by successively convolving $\beta^0(t)$ n times with itself. The first-order B-spline is the linear interpolation filter or tent filter:

$$\beta^1(t) = \beta^0(t) * \beta^0(t) = \begin{cases} 1 - |t| & \text{if } |t| \leq 1 \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

The 1D B-splines can be extended to the 3D CC lattice by a tensor product extension. BCC B-spline resampling exploits the decomposition property of the BCC lattice (Eq. 1). The reconstruction is performed separately for the two CC sub-lattices in the given sample position by using a standard separable CC B-spline resampling, and then the contributions are simply averaged [8, 6]. This evaluation is equivalent to the convolution of the BCC samples with a B-spline kernel.

3.4 DC-Splines for the BCC Lattice

The BCC lattice can be obtained from a CC lattice by removing the lattice points whose coordinates have different parity (Eq. 2). The BCC trilinear interpolation reproduces these "missing CC samples" by interpolating between the available BCC samples on the fly using a discrete filter. The resultant impulse response χ^1_{BCC} of the linear BCC DC-spline is obtained by convolving this discrete filter with a scaled trilinear kernel $\beta^1(2\mathbf{x})$:

$$\chi^1_{BCC}(\mathbf{x}) = \beta^1(2\mathbf{x}) + \frac{1}{2}\sum_{k=1}^{6} \beta^1(2(\mathbf{x} - \nu_{\mathbf{k}})) \quad (9)$$

$$[\nu_{\mathbf{1}\ldots\mathbf{6}}] = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

4 EVALUATION OF THE LINEAR BOX SPLINE FROM TRILINEAR TEXTURE SAMPLES

The major preference for applying the BCC B-spline filtering over the non-separable box spline filtering is the fact that separable filtering can be performed significantly faster on current GPUs due to the utilization of the hardware-accelerated trilinear texture fetching [20].

In order to make a fair comparison, an efficient evaluation scheme is required that uses trilinear texture fetches instead of nearest neighbor ones also for the box splines. In the following, we propose an algorithm for evaluation of the linear BCC box spline built upon a trilinear B-spline basis.

According to Eq. 6, the support of $M_{\Xi^1_{BCC}}$ covers four BCC samples that form a tetrahedron, thus the B-form of resampling is [11]:

$$f(\mathbf{r}) = \sum_{i=1}^{4} s(\mathbf{r}_i) M_{\Xi^1_{BCC}}(\mathbf{r_i} - \mathbf{r}), \quad (10)$$

where $\mathbf{r}$ is an arbitrary resampling point and s is a 3D array of the discrete BCC samples. Direct implementation of this B-form is rather inefficient, since a full kernel evaluation is performed for each $\mathbf{r_i}$ sample point [13].

A more efficient piecewise-polynomial evaluation scheme can be set up, since it is possible to evaluate the ordering of the absolute coordinates of $\mathbf{r_i} - \mathbf{r}$ in advance for each $\mathbf{r_i}$ lattice points [13].

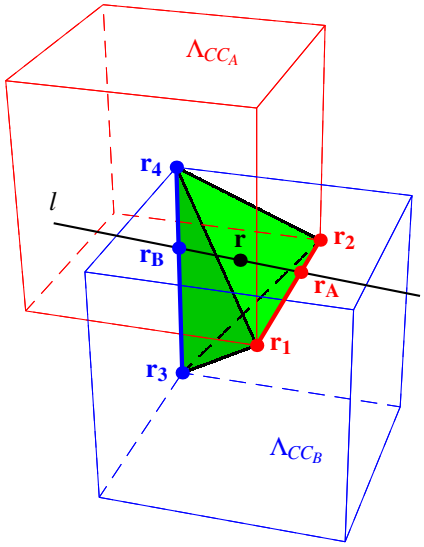


Figure 1: Trilinear evaluation scheme. For an arbitrary point $\mathbf{r}$, interpolation is performed within the green tetrahedron formed by the nearest points $\mathbf{r_1}, \mathbf{r_2} \in \Lambda_{CC_A}$ of the red CC lattice and the nearest points $\mathbf{r_3}, \mathbf{r_4} \in \Lambda_{CC_B}$ of the blue CC lattice. When $\mathbf{r}$ is an internal point, that is, $\mathbf{r} \notin \mathbf{r_{1,2}}$ and $\mathbf{r} \notin \mathbf{r_{3,4}}$, there is exactly one line l that intersects $\mathbf{r}$, and edges $\mathbf{r_{1,2}}$ and $\mathbf{r_{3,4}}$.

4.1 Trilinear Evaluation Scheme

The key point of the derivation lies in the fact that the linear box spline constitutes a linear interpolator on the BCC lattice [11]. This enables us to evaluate the linear interpolation within the tetrahedron more efficiently than a direct evaluation of Eq. 10.

The first observation we make is that the tetrahedron is composed of four congruent isosceles triangles (see Fig. 1):

1. $\mathbf{r_{1,2,3}}$ 2. $\mathbf{r_{1,2,4}}$ 3. $\mathbf{r_{3,4,1}}$ 4. $\mathbf{r_{3,4,2}}$

Four edges of the tetrahedron are formed by the equal sides of these triangles with the length of $\frac{\sqrt{3}}{2}$ while the remaining two edges of the tetrahedron are formed by the sides $\mathbf{r_{1,2}}$ and $\mathbf{r_{3,4}}$ of the triangles with the length of 1:

$$\begin{aligned} \frac{\sqrt{3}}{2} &= |\mathbf{r_{1,3}}| = |\mathbf{r_{2,3}}| = |\mathbf{r_{1,4}}| = |\mathbf{r_{2,4}}| \\ 1 &= |\mathbf{r_{1,2}}| = |\mathbf{r_{3,4}}| \end{aligned}$$

The edges $\mathbf{r_{1,2}}$ and $\mathbf{r_{3,4}}$ overlap the edges of the BCC lattice. Moreover, when the BCC lattice is considered as two interleaved CC lattices (Eq. 1), edge $\mathbf{r_{1,2}}$ is contained by the first CC lattice Λ_{CC_A}, while edge $\mathbf{r_{3,4}}$ is contained by the second CC lattice Λ_{CC_B}.

This enables us to rewrite the tetrahedral interpolation as the compound of three linear interpolations using the following scheme:

1. First, we define line l that contains $\mathbf{r}$ and intersects both $\mathbf{r_{1,2}} \in \Lambda_{CC_A}$ and $\mathbf{r_{3,4}} \in \Lambda_{CC_B}$ (see Fig. 1). The intersection points with edges $\mathbf{r_{1,2}}$ and $\mathbf{r_{3,4}}$ are $\mathbf{r_A}$ and $\mathbf{r_B}$, respectively. This decouples the BCC resampling problem into resamplings of two separate CC lattices, to Λ_{CC_A} and Λ_{CC_B}.

2. Next, the discrete data is resampled in $\mathbf{r_A}$ for Λ_{CC_A} and $\mathbf{r_B}$ for Λ_{CC_B} using a simple linear kernel:

$$\begin{aligned} f_A &= s_A(\mathbf{r_1} + |\mathbf{r_{1,A}}|\mathbf{r_{1,2}}) \\ f_B &= s_B(\mathbf{r_3} + |\mathbf{r_{3,B}}|\mathbf{r_{3,4}}), \end{aligned} \tag{11}$$

   where s_A and s_B are linearly addressable 3D arrays of the discrete CC samples corresponding to Λ_{CC_A} and Λ_{CC_B}, respectively.

3. Finally, the linear combination of the two CC samples is calculated:

$$f(\mathbf{r}) = f_A + \frac{|\mathbf{r} - \mathbf{r_A}|}{|\mathbf{r_{A,B}}|}(f_B - f_A). \tag{12}$$

The clear advantage of this evaluation scheme is that Step 2 can be performed by only two trilinear fetches on the GPU instead of four nearest neighbor fetches. Actually, these trilinear fetches involve in fact only 1D linear interpolations since $\mathbf{r_A}$ and $\mathbf{r_B}$ lie on a lattice edge. Regarding the storage scheme, the consequence is that the BCC samples need to be stored in two separate CC lattices, i.e. conventional 3D textures, to be able to exploit the trilinear fetching capability of the GPU just like in case of the BCC B-spline and the BCC DC-spline.

4.2 Orientation Cases

Addressing $\mathbf{r_1}$, $\mathbf{r_2}$, $\mathbf{r_3}$, and $\mathbf{r_4}$ for an arbitrary $\mathbf{r}$ is required in Step 1 which needs some further explanation. Let $\mathbf{r_{base}} = \text{round}(\mathbf{r})$ be the nearest lattice point in Λ_{CC_A} and let $\mathbf{d} = \mathbf{r} - \mathbf{r_{base}}$ be the relative resampling coordinates with their absolute values $\mathbf{a} = [|d_x|, |d_y|, |d_z|]^T \in [0, \frac{1}{2})^3$ and their signs $\mathbf{s} = [\text{sgn}(d_x), \text{sgn}(d_y), \text{sgn}(d_z)]^T$. Considering the symmetries of the rhombic dodecahedral support of $M_{\Xi^1_{BCC}}$, six different orientations of the resampling tetrahedron can be distinguished (see Fig. 2). These six cases are the 3! possible orderings of the absolute coordinates $\mathbf{a}$ in Eq. 6 as it was reported in [13].

Since using any control flow statement in the resampling implementation dramatically cuts the performance of the GPUs which have a SIMD architecture, it is advisable to avoid this six-fold branching. Descending order of three scalars can be calculated in a SIMD-aware manner as:

$$\begin{aligned} x &= max(a_x, a_y, a_z) \qquad z = min(a_x, a_y, a_z) \\ y &= a_x + a_y + a_z - x - z. \end{aligned} \tag{13}$$

On the other hand, based on the sort order of a_x, a_y, and a_z all the six orientations of the resampling tetrahedron can be transformed back to the first one (the green

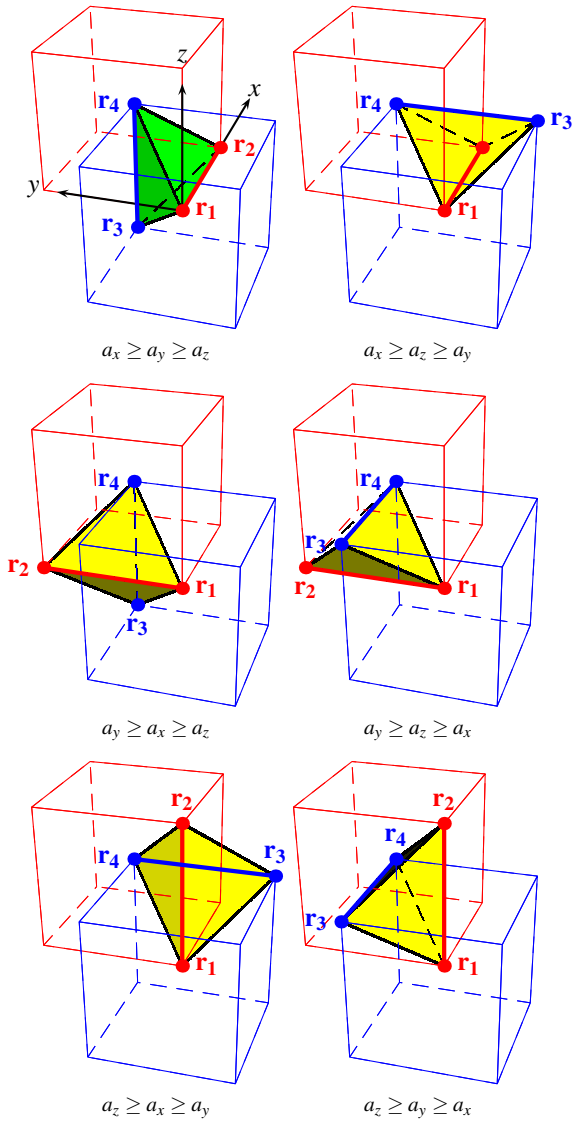


Figure 2: There are six orientation cases for ordering the coordinates of $\mathbf{a} \in [0, \frac{1}{2})^3$. The required resampling points $\mathbf{r_1}, \mathbf{r_2} \in \Lambda_{CC_A}$ and $\mathbf{r_3}, \mathbf{r_4} \in \Lambda_{CC_B}$ are determined by these six cases. These resampling points are indicated as red and blue dots for each orientation case.

tetrahedron for $a_x \geq a_y \geq a_z$ in Fig. 2). Thus, the resampling formula needs to be written only for the first orientation case, and the other cases can be retrieved by using this transformation. The transformation can be defined by a rotation matrix $\mathbf{\Pi}$ as

$$\Pi_{i,j} = s_i \cdot e_{\pi(j),i}, \tag{14}$$

where $\mathbf{e}_{\pi(1)}$, $\mathbf{e}_{\pi(2)}$, and $\mathbf{e}_{\pi(3)}$ are the unit vectors corresponding to x, y, and z, respectively (Eq. 13). As a compact notation, π represents the descending order of a_x, a_y, and a_z as a permutation. By using $\mathbf{\Pi}$, the lattice points can be addressed as

$$\mathbf{r_1} = \mathbf{r_{base}} + \mathbf{\Pi}\,[0\;0\;0]^T \qquad \mathbf{r_2} = \mathbf{r_{base}} + \mathbf{\Pi}\,[1\;0\;0]^T$$
$$\mathbf{r_3} = \mathbf{r_{base}} + \mathbf{\Pi}\left[\frac{1}{2}\;\frac{1}{2}\;-\frac{1}{2}\right]^T \qquad \mathbf{r_4} = \mathbf{r_{base}} + \mathbf{\Pi}\left[\frac{1}{2}\;\frac{1}{2}\;\frac{1}{2}\right]^T.$$

4.3 Formal Derivation

In the following, we also give a formal derivation of the proposed algorithm. The derivation is based on the rewriting of the tetrahedral interpolation in barycentric coordinates. Barycentric coordinates provide a convenient way for interpolation on a tetrahedral mesh:

$$f(\mathbf{r}) = \sum_{i=1}^{4} \lambda_i s(\mathbf{r_i}), \tag{15}$$

where scalars $\lambda_{1\ldots4}$ are barycentric coordinates of $\mathbf{r}$ with respect to the vertices of the tetrahedron $\mathbf{r_{1\ldots4}}$ under the constraint $\sum_{i=1}^{4} \lambda_i = 1$. The barycentric expansion of $\mathbf{r}$ is set up in terms of the vertices of the tetrahedron as:

$$\begin{aligned} \mathbf{T\lambda} &= \mathbf{r} - \mathbf{r_4} \\ \mathbf{T} &= [\mathbf{r_1} - \mathbf{r_4} \mid \mathbf{r_2} - \mathbf{r_4} \mid \mathbf{r_3} - \mathbf{r_4}] \\ \boldsymbol{\lambda} &= \left[\;\lambda_1 \quad \lambda_2 \quad \lambda_3\;\right]^T. \end{aligned} \tag{16}$$

The solution of this linear equation system is

$$\mathbf{T} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & -1 \end{bmatrix}, \quad \mathbf{T}^{-1} = \begin{bmatrix} -1 & -1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix},$$

$$\lambda_1 = 1 - x - y \qquad \lambda_2 = x - y$$
$$\lambda_3 = y - z \qquad \lambda_4 = y + z.$$

This enables us to write Eq. 10 as

$$f(\mathbf{r}) = \sum_{i=1}^{2} \lambda_i s_A(\mathbf{r_i}) + \sum_{i=3}^{4} \lambda_i s_B(\mathbf{r_i}). \tag{17}$$

Using the separable trilinear technique of Sigg and Hadwiger [20], evaluation of Eq. 17 can be derived by two linear fetches instead of four nearest neighbor ones. In general, two nearest neighbor fetches can be replaced by a linear fetch as:

$$\begin{aligned} (1-t)\,f_i + t\,f_{i+1} &\;\Rightarrow\; f(i+t) \\ a\,f_i + b\,f_{i+1} &\;\Rightarrow\; (a+b)\,f\left(i + \frac{b}{a+b}\right), \end{aligned} \tag{18}$$

as long as $t \in [0,1]$ and $\frac{b}{a+b} \in [0,1]$. By combining both λ_1 with λ_2 and λ_3 with λ_4, the linear box spline can be evaluated by two linear texture fetches:

$$\sum_{i=1}^{2} \lambda_i s_A(\mathbf{r_i}) \;\Rightarrow\; (1-2y)\,s_A\left(\mathbf{r_1} + \frac{x-y}{1-2y}\mathbf{\Pi}\begin{bmatrix}1\\0\\0\end{bmatrix}\right)$$
$$\sum_{i=3}^{4} \lambda_i s_B(\mathbf{r_i}) \;\Rightarrow\; 2y\,s_B\left(\mathbf{r_3} + \frac{y+z}{2y}\mathbf{\Pi}\begin{bmatrix}0\\0\\1\end{bmatrix}\right)$$

Summing these terms up, we get

$$
\begin{aligned}
f_A &= s_A\left(\mathbf{r_{base}} + \frac{x-y}{1-2y}\,\mathbf{s}\circ\mathbf{e}_{\pi(1)}\right) \qquad (19)\\
f_B &= s_B\left(\mathbf{r_{base}} + \mathbf{s}\circ\left(\begin{bmatrix}1/2\\1/2\\1/2\end{bmatrix} + \frac{z-y}{2y}\mathbf{e}_{\pi(3)}\right)\right),\\
f(\mathbf{r}) &= (1-2y)\,f_A + 2y\,f_B
\end{aligned}
$$

where $\circ$ represents the element-wise product. This is exactly what was claimed in Step 2 and Step 3 of the proposed evaluation scheme.

5 GPU IMPLEMENTATION

We employed the proposed trilinear evaluation scheme formulated in Eq. 19 in a GPU-based first-hit ray-casting application by using ray marching with equidistant steps. At each sample position, a filter kernel was used to reconstruct the volume from the discrete BCC samples. To get a numerically stable formulation when the resampling point lies within a triangular face, on an edge, or coincides a vertex, the divisions in Eq. 19 are evaluated as $\lim_{\varepsilon\to 0}\varepsilon\cdot s_i(\frac{\text{constant}}{\varepsilon}) = 0$. This numerical safeguard was incorporated in the GPU implementation as well.

In our GPU implementation, the lattice samples are stored as textures. Function $s_A(\mathbf{r})$ fetches the sample set s_A at $\mathbf{r}+[\frac{1}{2},\frac{1}{2},\frac{1}{2}]^T$, while function $s_B(\mathbf{r})$ fetches the shifted sample set s_B at $\mathbf{r}$. Sample sets s_A and s_B can be implemented as two separate textures or as one texture with two channels. We have not found an appreciable difference between these two methods. We present the complete Cg source of the proposed linear box spline resampling algorithm in the appendix.

We compare the rendering speed of our trilinearly evaluated linear box spline scheme to the latest method, that uses twice as many nearest neighbor fetches [13]. We also give a comparison to the major competitive methods: to the BCC B-spline [8] and to the BCC DC-spline [10]. Comprehensive analysis of the numerical accuracy, and visual quality of these splines are out of the scope of this paper. We refer the interested reader to [6, 10, 13] for a more thorough overview.

The number of texture lookups and the arithmetic costs differ for each filter (see Table 1). The arithmetic cost of the trilinear B-spline filtering is practically negligible [6, 8], the DC-spline filtering has moderate addressing overhead [10], while the trilinear and nearest neighbor linear box spline schemes have the highest number of floating point operations [13]. Concerning the number of texture fetches, the trilinear B-spline and the trilinearly evaluated linear box spline are in the best position: they need only two lookups, while the linear box spline filtering needs four fetches, and the DC-spline filtering needs six lookups.

Filter	lookups	complexity
Lin. box spline (nearest)	4	high
Lin. box spline (linear)	2	high
Trilinear B-spline	2	low
Linear DC-spline	6	medium

Table 1: Number of texture lookups and the arithmetic cost of different reconstruction filters. These properties determine the rendering performance.

The skeleton of the ray caster application was the same for each filtering technique, only the filter kernels and the storage scheme of the BCC samples were altered. For the nearest neighbor box spline evaluation, the BCC samples are stored in a one-channel texture by shifting the samples of the second lattice by half a grid spacing in every dimension [13]. For the trilinear box spline scheme, for the BCC B-splines, and for the BCC DC-splines, the BCC samples were stored as two separate set of CC samples as a two-channel texture.

5.1 Rendering Speed

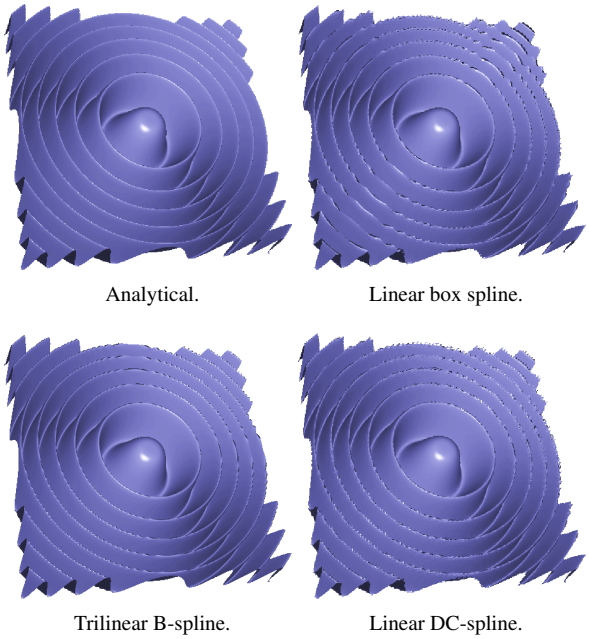
We rendered four data sets of different voxel counts at an image resolution of 512×512. The analytically defined Marschner-Lobb test signal was sampled at $64^3\times 2$ BCC resolution. The other three data sets are well-known CT scans reconstructed originally on a CC lattice. To get a BCC representation of them, we employed a frequency-domain upsampling [7].

The viewing rays were evaluated in front-to-back order, which enabled us to use early ray termination. The first-hit isosurfaces were shaded by the Blinn-Phong model using gradients calculated from central differences. The ray marching step and the central differencing step were adjusted to the voxel size of the data sets.

The renderings of the Marschner-Lobb test signal are illustrated in Figure 3. Note that the linear box spline introduces postaliasing artifacts along the diagonal directions, while the artifacts produced by the linear DC-spline or the trilinear B-spline are less apparent.

To get relevant rendering speeds, we chose the middle-aged NVIDIA Geforce 8700 GPU for our experiments. The observed frame rates are illustrated in Table 2. According to our prior expectations, the frame rates depended on the number of samples fetched, the algorithmic complexity of the filter kernel, the resolution of the volume, and the distance of the iso-surface from the image plane.

We can confirm the observation, that the frame rates get similar as the number of voxels increases with appropriately decreasing the sampling distance. Possibly, the texture fetches become the bottleneck of the rendering pipeline. This can be the reason why the DC-spline results in the lowest frame rates for the highest voxel counts.



Analytical. Linear box spline.

Trilinear B-spline. Linear DC-spline.

Figure 3: Renderings of the analytical Marschner-Lobb test signal and its sampled representations at $64^3 \times 2$ reconstructed by different resampling filters.

Data set	$M_{\Xi_{BCC}^{1,nearest}}$	$M_{\Xi_{BCC}^{1,linear}}$	β^1	χ_{BCC}^1
ML	21.03	22.90	53.64	21.75
Engine	16.28	17.18	41.82	16.42
Carp	9.22	10.00	25.07	9.19
Xmas Tree	5.77	6.19	6.57	4.61

Table 2: Frame rates in frames per second for different reconstruction filters and popular data sets: the Marschner-Lobb test signal sampled at $64^3 \times 2$, the "Engine Block" at $256^2 \times 110 \times 2$, the "Carp" at $256^2 \times 512 \times 2$, and the "Christmas Tree" at $512 \times 499 \times 512 \times 2$.

On the other hand, for low and moderate volume resolutions, the arithmetic complexity seems to be more important than the number of texture fetches. It is interesting to note that the concept of applying linear fetches instead of nearest neighbor ones [20] does not always pay off. We think that the texture cache operates very well for filters with a narrow support. This might explain that the nearest neighbor version and the linear version of the linear box spline filtering as well as the linear DC-spline filtering with even six samples attain similar frame rates, while the trilinear B-spline holds a towering lead in performance.

6 CONCLUSION AND FUTURE WORK

In this paper, we have proposed a GPU evaluation scheme for the linear BCC box spline filtering exploiting the hardwired trilinear texture fetching. This result enabled us to make a fair comparison of the linear box spline, the BCC B-spline, and the BCC DC-spline in terms of their performance. We found that, in general, the proposed linear evaluation scheme operates slightly faster than the evaluation scheme with nearest neighbor fetches [13]. However, using an optimized GPU implementation, the trilinear B-spline can still achieve the best performance, as it takes the minimum number of samples with the lowest arithmetic cost. Since the texture fetches become more expensive when the support of the filter gets wider or the resolution of the volume increases, we plan to develop a similar scheme for the quintic box spline for the BCC lattice.

ACKNOWLEDGMENTS

This project was supported by Mediso Medical Imaging Systems, the Hungarian National Office for Research and Technology (Project ID: TECH 08/A2), and the New Hungary Development Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002). This work is connected to the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" project.

REFERENCES

[1] T. Blu, P. Thévenaz, and M. Unser. Generalized interpolation: Higher quality at no additional cost. In *Proceedings of IEEE International Conference on Image Processing*, pages 667–671, 1999.

[2] Dutta Roy S. C. and Kumar B. *Handbook of Statistics*, volume 10, chapter Digital Differentiators, pages 159–205. Elsevier Science Publishers B. V., N. Holland, 1993.

[3] I. Carlbom. Optimal filter design for volume reconstruction and visualization. In *Proceedings of the 4th conference on Visualization*, pages 54–61. IEEE Computer Society, 1993.

[4] B. Csébfalvi. Prefiltered gaussian reconstruction for high-quality rendering of volumetric data sampled on a body-centered cubic grid. In *Proceedings of IEEE Visualization*, pages 311–318, 2005.

[5] B. Csébfalvi. BCC-splines: Generalization of B-splines for the body-centered cubic lattice. *Journal of WSCG 16, 1–3 (2008)*, pages 81–88, 2008.

[6] B. Csébfalvi. An evaluation of prefiltered B-spline reconstruction for quasi-interpolation on the body-centered cubic lattice. *IEEE Transactions on Visualization and Computer Graphics 16, 3 (2010)*, pages 499–512, 2010.

[7] B. Csébfalvi and B. Domonkos. Frequency-domain upsampling on a body-centered cubic lattice for efficient and high-quality volume rendering. In *Proceedings of Vision, Modeling, and Visualization*, pages 225–232, 2009.

[8] B. Csébfalvi and M. Hadwiger. Prefiltered B-spline reconstruction for hardware-accelerated rendering of optimally sampled volumetric data. In *Proceedings of VMV*, pages 325–332, 2006.

[9] C. de Boor, K. Höllig, and S. Riemenschneider. *Box Splines*, volume 98. Springer-Verlag, 1993.

[10] B. Domonkos and B. Csébfalvi. DC-splines: Revisiting the trilinear interpolation on the body-centered cubic lattice. In *Proceedings of VMV*, pages 275–282, 2010.

[11] A. Entezari. *Optimal Sampling Lattices and Trivariate Box Splines*. PhD thesis, Simon Fraser University, Vancouver, Canada, July 2007.

[12] A. Entezari, D. Van De Ville, and T. Möller. Practical box splines for reconstruction on the body centered cubic lattice. *IEEE Trans. on Vis. and Computer Graphics*, 14(2):313–328, 2008.

[13] B. Finkbeiner, A. Entezari, D. Van De Ville, and T. Möller. Efficient volume rendering on the body centered cubic lattice using box splines. *Computers and Graphics*, 34(4):409–423, 2010.

[14] L. Kobbelt. Stable evaluation of box splines. *Numerical Algorithms*, 14, 1996.

[15] O. Mattausch. Practical reconstruction schemes and hardware-accelerated direct volume rendering on bodycentered cubic grids. Master's thesis, Vienna University of Technology, 2003.

[16] M. Mirzargar and A. Entezari. Voronoi splines. *IEEE Transactions on Signal Processing*, 58:4572–4582, Sept 2010.

[17] T. Möller, R. Machiraju, K. Mueller, and R. Yagel. A comparison of normal estimation schemes. In *Proceedings of the 8th conference on Visualization*, pages 19–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[18] T. Möller, K. Mueller, Y. Kurzion, R. Machiraju, and R. Yagel. Design of accurate and smooth filters for function and derivative reconstruction. In *Proceedings of the IEEE symposium on Volume visualization*, pages 143–151, New York, NY, USA, 1998. ACM.

[19] D. P. Petersen and D. Middleton. Sampling and reconstruction of wave-number-limited functions in n-dimensional euclidean spaces. *Information and Control*, 5(4):279–323, 1962.

[20] C. Sigg and M. Hadwiger. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, pages 313–329. Addison- Wesley, 2005.

[21] T. Theußl, T. Möller, and M. E. Gröller. Optimal regular volume sampling. In *Proceedings of the conference on Visualization*, pages 91–98. IEEE Computer Society, 2001.

[22] T. Theußl, T. Möller, and M. E. Gröller. Reconstruction schemes for high quality raycasting of the body-centered cubic grid. Technical Report TR-186-2-02-11, Institute of Computer Graphics and Algorithms, TU Vienna, Dec 2002.

[23] D. Van De Ville, T. Blu, M. Unser, W. Philips, I. Lemahieu, and R. Van de Walle. Hex-Splines: A novel spline family for hexagonal lattices. *IEEE Transactions on Image Processing*, 13(6):758–772, 2004.

CG SHADER CODE

```
uniform float3    Size;
uniform sampler3D Volume;

// Handle removable singularity
#define DIV( A, B ) \
  ( abs(B) ? ( A ) / ( B ) : 0.0 )

// I^th coordinate of unit vector e_π(1)
#define E_PI_1( I ) ( a.I == x )

// I^th coordinate of unit vector e_π(3)
#define E_PI_3( I ) \
  ( a.I == z && a.I != x && a.I != y )

// Unit vector e_π(J)
#define E_PI( J ) float3( E_PI_ ## J( x ), \
  E_PI_ ## J( y ), E_PI_ ## J( z ) )

// Fetching a trilinear sample from Λ_CC_A at R
#define S_A( R ) \
  tex3D(Volume, (r_base + (R) + 0.5) / Size).r

// Fetching a trilinear sample from Λ_CC_B at R
#define S_B( R ) \
  tex3D( Volume, ( r_base + (R) ) / Size ).a

float linearBoxSpline( float3 texCoords ) {
  // Resampling point r
  float3 r = texCoords * Size - 0.5;

  // Nearest lattice point of Λ_CC_A
  float3 r_base = round( r );

  // Relative coordinates d,
  // their absolute values a, and signs s
  float3 d = r - r_base;
  float3 a = abs( d );
  float3 s = sign( d );

  // Sorting a by its components
  float x = max( a.x, max( a.y, a.z ) );
  float z = min( a.x, min( a.y, a.z ) );
  float y = a.x + a.y + a.z - x - z;

  // Fetching from sample sets Λ_CC_A and Λ_CC_B
  float two_y = 2.0 * y;
  float tA = DIV( x - y, 1.0 - two_y );
  float tB = DIV( z - y, two_y );
  float fA = S_A( tA * s * E_PI(1) );
  float fB = S_B(s * ( 0.5 + tB * E_PI(3) ));

  // Linear interpolation of the two samples
  return lerp( fA, fB, two_y );
}
```