효율적인 BCC 볼륨 데이터의 GPU 등가면 광선투사법

김민호° 김현준 Aaliya Sarfaraz

서울시립대학교 컴퓨터과학부

{minhokim, kamu1324}@uos.ac.kr, mohdilyas76@yahoo.com

Efficient GPU Isosurface Ray-casting of BCC Datasets

Minho Kim° Hyunjun Kim Aaliya Sarfaraz

Dept. of Computer Science, University of Seoul

요 약

본 연구에서는, 결과물의 우수함을 유지하면서도 기존 연구결과의 성능을 4–7배 향상시킨 실시간 GPU 등가면 볼륨 레이케스터를 제시한다. 이러한 성능향상은 효율적인 빈공간 생략법과 분석적 그레디언트 계산법을 도입함으로써 가능하다. 빈공간 생략법은 스플라인 조각들의 BB-형식으로부터 만들어진 최소/최대값 옥트리에 기반하고 있고, 분석적 그레디언트 계산법은 보다 정확한 렌더링 결과를 제공할 뿐 아니라 더욱 빠른 계산이 가능하도록 한다.

Abstract

This paper presents a real-time GPU (Graphics Processing Unit) isosurface ray-caster that improves the performance by 4–7 folds from our previous method, while keeping the superior visual quality. Such an improvement is achieved by incorporating an efficient empty-space skipping scheme and an analytic normal computation. The empty-space skipping scheme is done by building an min/max octree computed from the BB(Bernstein-Bézier)-form of spline pieces and the analytic normal formula provides not only a nice visual quality but also an improved evaluation performance.

키워드: GPU, 광선투사법, BCC 볼륨데이터, 박스-스플라인

Keywords: GPU, Ray-casting, BCC volume datasets, Box-splines

1 Introduction

Real-time volume rendering is one of the mostly investigated research area where the modern graphics hardware is playing an important role due to their powerful parallel processing feature. On the other hand, it has been known for a while that the most popular and hardware-friendly Cartesian lattice is not an efficient sampling lattice compared to other regular lattices such as BCC (Body-Centered Cubic) and FCC (Face-Centered Cubic) lattices. Specifically, it is shown that the BCC lattice is the optimal 3-dimensional sampling lattice and several reconstruction filters for BCC datasets have been proposed, one of which is the 7-direction quartic box-spline filter proposed by Kim [1]. This paper extends our previous work [2] to improve the performance of the GPU isosurface ray-caster by leveraging the analytic gradient computation and efficient empty space skipping method proposed by Kim [3].

2 Previous Work

Since it has been shown that the BCC lattice is the optimal 3-dimensional sampling lattice [4], several reconstruction filters for the BCC lattice have been proposed and box-spline filters showed the best results compared to others. Entezari *et al.* [5] proposed the 8-direction quintic box-spline, bcc8, that shows the superior reconstruction quality. Later Finkbeiner *et al.* [6] implemented a GPU isosurface ray-caster based on bcc8 leveraging an efficient spline evaluation scheme. Csébfalvi and Hadwiger [7] proposed the tricubic B-spline filter, bcc12, with a GPU isosurface ray-caster that shows extremely superior performance due to its hardware-friendly structure, but the reconstructed volume is oversmoothed. Recently, Kim [1] proposed the 7-direction quartic box-spline, bcc7, that outperforms the other two in both quality and performance when evaluated on the CPU. Note that all three box-spline reconstruction filters share the same approximation order 3. Kim and Lee [2] proposed a real-time GPU isosurface ray-caster leveraging an efficient evaluation of splines based on three optimizations; (i) constructing transformation matrices by encoding instead of lookup tables, (ii) stencil offset transformation by vector addition instead of an expensive matrix-vector multiplication, and (iii) an efficient partially-factored power formula with a low computational overhead.

For FCC datasets, Kim *et al.* [8] proposed the 6-direction box-spline filter and Kim [3] implemented the GPU isosurface ray-caster that accelerated the performance by adopting two optimizations; analytic gradient computation and an efficient empty-space skipping method, which are incorporated in our work.

3 Background

In this section, we first describe the BCC lattice followed by a brief review of the box-splines. Refer to [9] for more details about box-splines. Lastly we introduce the seven-direction box-spline filter we adopted in this work.

In this paper, vectors and matrices are typeset as bold lower and upper case letters, *e.g.*, $\boldsymbol{x}$ and $\boldsymbol{M}$, respectively. $\boldsymbol{x}(j)$ denotes the j-th component of the vector $\boldsymbol{x}$.

3.1 The BCC Lattice

A three-dimensional lattice $\mathcal{L}$ is defined as the set of all the points generated by integer linear combinations of a non-singular *generator matrix* $\boldsymbol{G}$:

$$\mathcal{L} := \{\boldsymbol{G}\boldsymbol{j} : \boldsymbol{j} \in \mathbb{Z}^3, \boldsymbol{G} \in \mathbb{R}^{3\times 3}, \det \boldsymbol{G} \neq 0\}. \tag{1}$$

The BCC lattice is defined as a subset of the three-dimensional Cartesian lattice $\mathbb{Z}^3$ where the three coordinates are all even or all odd, *i.e.*,

$$\mathbb{Z}_{\text{bcc}} := \{\boldsymbol{j} \in \mathbb{Z}^3 : \boldsymbol{j}(1), \boldsymbol{j}(2), \text{ and } \boldsymbol{j}(3) \text{ are all even or all odd}\}. \tag{2}$$

The same set of points can be obtained by the generator matrix

$$\boldsymbol{G}_{\text{bcc}} := \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}. \tag{3}$$

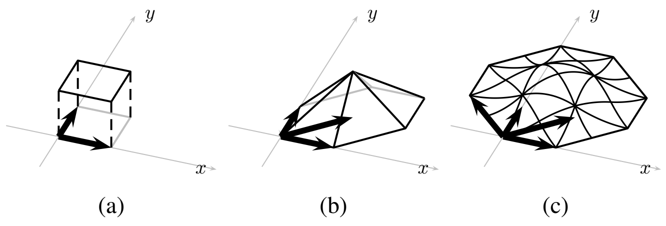

Figure 1: Construction of the box-splines defined by the direction matrices (a) $\left[\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right]$, (b) $\left[\begin{smallmatrix}1&0&1\\0&1&1\end{smallmatrix}\right]$, and (c) $\left[\begin{smallmatrix}1&0&1&-1\\0&1&1&1\end{smallmatrix}\right]$ via successive directional convolutions.

Since the dual of the BCC lattice, the FCC lattice, is the densest sphere packing lattice, the BCC lattice is the optimal three-dimensional sampling lattice for band-limited and isotropic signals [4].

3.2 Box-Splines

In the following, depending on the context, a matrix may also denote a set of column vectors allowing multiplicity.

A box-spline is a piecewise polynomial with a finite support and certain continuity defined by a *direction matrix*. Given an $n \times m$ (usually $n < m$) direction matrix Ξ, a box-spline M_Ξ is constructed by a successive directional convolutions along the directions in Ξ. In other words, with the base case ($n = m$)

$$M_\Xi(\boldsymbol{x}) := \frac{1}{|\det \Xi|}\chi_\Xi(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^n \tag{4}$$

where Ξ is non-singular and χ_Ξ is the characteristic function on the half-open parallelepiped $\Xi[0,1)^m$, a box-spline defined by the direction matrix $\Xi \cup \{\boldsymbol{\xi}\}$ is recursively defined as (Figure 1)

$$M_{\Xi\cup\{\boldsymbol{\xi}\}}(\boldsymbol{x}) := \int_0^1 M_\Xi(\boldsymbol{x} - t\boldsymbol{\xi})dt, \quad \boldsymbol{\xi} \in \mathbb{R}^n. \tag{5}$$

A box-spline can be used as a reconstruction filter for a discrete dataset sampled on the Cartesian lattice; given a discrete dataset $V : \mathbb{Z}^n \to \mathbb{R}$, a (mostly continuous) spline $s(\boldsymbol{x})$ can be constructed by taking a convolution with a box-spline filter M_Ξ:

$$s(\boldsymbol{x}) := (V * M_\Xi)(\boldsymbol{x}) := \sum_{\boldsymbol{j}\in\mathbb{Z}^n} V(\boldsymbol{j})M_\Xi(\boldsymbol{x} - \boldsymbol{j}). \tag{6}$$

But we can obtain a better reconstruction by applying a discrete quasi-interpolation prefilter q on the dataset beforehand:

$$(V \star q) * M_\Xi, \tag{7}$$

where $\star$ denotes the discrete convolution defined as

$$(V \star q)(\boldsymbol{k}) := \sum_{\boldsymbol{j}\in\mathbb{Z}^n} V(\boldsymbol{j})q(\boldsymbol{k}-\boldsymbol{j}). \tag{8}$$

For a direction $\boldsymbol{\xi} \in \Xi$, the directional derivative along $\boldsymbol{\xi}$, $D_{\boldsymbol{\xi}} M_{\Xi}$, can be expressed as a backward difference of the box-spline defined by $\Xi\backslash\{\boldsymbol{\xi}\}$ [9]:

$$D_{\boldsymbol{\xi}} M_{\Xi} = \nabla_{\boldsymbol{\xi}} M_{\Xi\backslash\{\boldsymbol{\xi}\}} \tag{9}$$

where $\nabla_{\boldsymbol{\xi}}$ denotes the backward difference operator along $\boldsymbol{\xi}$. Since $D_{\boldsymbol{\xi}}$ is a linear operator, the directional derivative of a spline (reconstructed by M_{Ξ}) along $\boldsymbol{\xi} \in \Xi$ can be represented as a spline reconstructed by $\nabla_{\boldsymbol{\xi}} M_{\Xi\backslash\{\boldsymbol{\xi}\}}$:

$$\begin{aligned} D_{\boldsymbol{\xi}}(V * M_{\Xi}) &= V * (D_{\boldsymbol{\xi}} M_{\Xi}) = V * (\nabla_{\boldsymbol{\xi}} M_{\Xi\backslash\{\boldsymbol{\xi}\}}) \\ &= V * (M_{\Xi\backslash\{\boldsymbol{\xi}\}} - M_{\Xi\backslash\{\boldsymbol{\xi}\}}(\cdot - \boldsymbol{\xi})). \end{aligned} \tag{10}$$

3.3 The Seven-Direction Quartic Box-Spline on the BCC Lattice

The seven-direction quartic box-spline M_7 is the 'centered' and 'scaled' version of the box-spline defined by the direction matrix Ξ_7 [1]:

$$M_7(\boldsymbol{x}) := |\det \boldsymbol{G}_{\mathrm{bcc}}| M_{\Xi_7}(\boldsymbol{x} + \boldsymbol{\xi}_c), \tag{11}$$

where $|\det \boldsymbol{G}_{\mathrm{bcc}}| = 4$,

$$\Xi_7 := \begin{bmatrix} 2 & 0 & 0 & -1 & 1 & 1 & -1 \\ 0 & 2 & 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & 2 & 1 & 1 & -1 & -1 \end{bmatrix}, \tag{12}$$

and

$$\boldsymbol{\xi}_c := \frac{1}{2}\sum_{\boldsymbol{\xi}\in\Xi_7} \boldsymbol{\xi} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{13}$$

Its total polynomial degree is $7-3=4$ and its support has the shape of a truncated rhombic dodecahedron. The optimal approximation order, 3, can be obtained by the discrete quasi-interpolation prefilter [1]

$$q_7(\boldsymbol{j}) := \begin{cases} 19/12 & \boldsymbol{j} = \boldsymbol{0} \\ -1/24 & \boldsymbol{j} \in \{\pm\boldsymbol{\xi} : \boldsymbol{\xi} \in \Xi_7\} \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

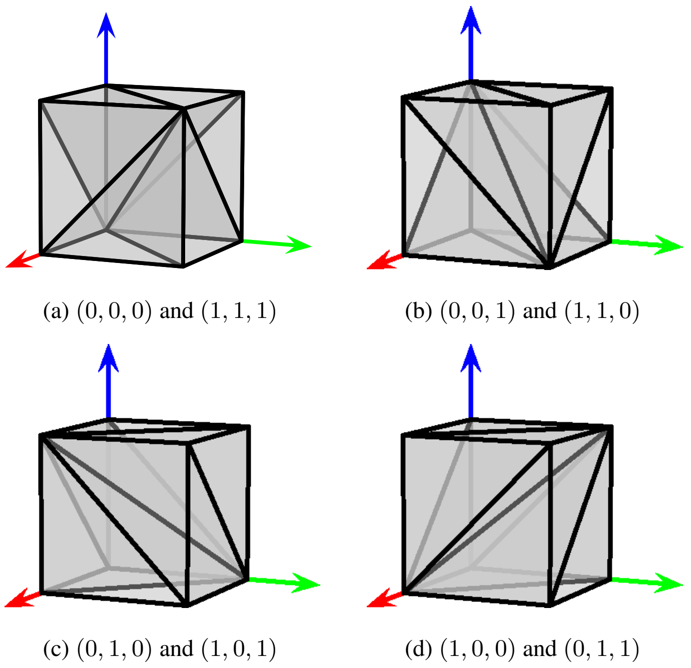
Figure 2: Four types of shift-invariant cubes. The three tuples denote the parity of the lower corner of each cube.

(a) $(0,0,0)$ and $(1,1,1)$ (b) $(0,0,1)$ and $(1,1,0)$ (c) $(0,1,0)$ and $(1,0,1)$ (d) $(1,0,0)$ and $(0,1,1)$

4 GPU Isosurface Ray-casting of BCC Datasets

In this section, we first briefly review the evaluation scheme of splines generated by M_7, which is the basis of our previous work [2]. Then we describe the analytic gradient computation scheme that results in a nice visual quality and improves the performance of isosurface ray-casting. We complete this section with the efficient empty-space skipping method that further improves the overall performance.

4.1 Evaluation of Splines

To evaluate a spline generated by M_7, we need to analyze the polynomial structure. There are total nine shift-invariant knot planes and three axis-aligned ones of them delineates a spline into four types of shift-invariant cubes (Figure 2). We choose the cube in Figure 2a as the *reference cube*. Note that other three types of cubes can be transformed to the reference one by a proper translation followed by a reflection. Each cube is further decomposed into six tetrahedral polynomial pieces by three knot planes intersecting each cube. Figure 3 shows the six types of tetrahedra for the reference cube. Among them we choose the tetrahedron in Figure 3a as the *reference tetrahedron*. Note again that other five types of tetrahedra can be transformed to the reference one by a proper permutation transformation.

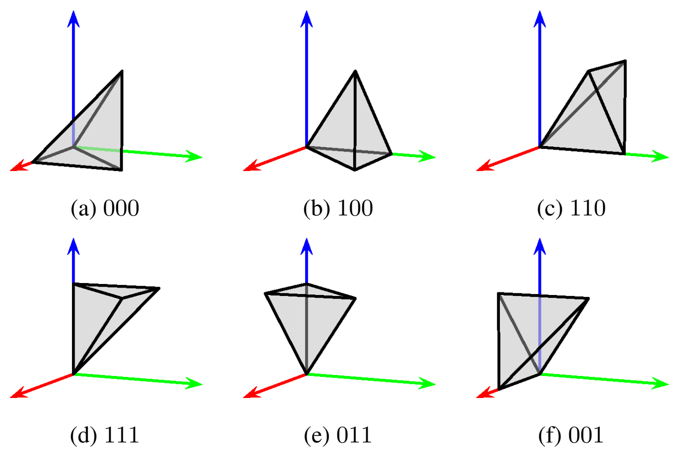

Figure 3: Six types of shift-invariant tetrahedral polynomial pieces for the reference cube. The code for each tetrahedron is computed by the membership test against three knot planes intersecting the cube.

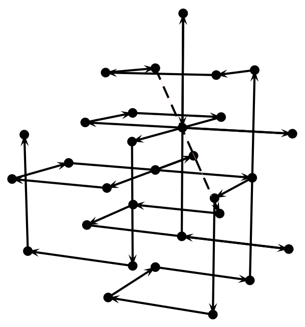

Figure 4: The order of 30 data fetches.

While there are total 24 shift-invariant tetrahedral polynomial pieces, due to the nice symmetry of M_7, we only need to consider evaluating the spline for the reference tetrahedron, since all the other tetrahedra can be transformed to the reference one by a proper matrix. But multiplying a matrix for each data fetch induces a computational overhead. We can reduce such an overhead since the matrices are all transform the coordinates system axis-aligned. Specifically, we can reorder the fetch order such that adjacent data fetch is done along axis-aligned direction as in Figure 4. Then the matrix-vector multiplication can be reduced to a simpler vector addition.

After fetching all data required, the polynomial is efficiently evaluated using a partially-factored form [1].

4.2 Analytic Gradient Computation

Based on (10), we can define a *directional derivative kernel* $M_{\boldsymbol{\xi}}$, $\boldsymbol{\xi} \in \Xi_7$, as follows:

$$M_{\boldsymbol{\xi}} := \nabla_{\boldsymbol{\xi}} M_7 = 4M_{\Xi_7\setminus\{\boldsymbol{\xi}\}}(\cdot + \boldsymbol{\xi}_c) - 4M_{\Xi_7\setminus\{\boldsymbol{\xi}\}}(\cdot - \boldsymbol{\xi} + \boldsymbol{\xi}_c). \quad (15)$$

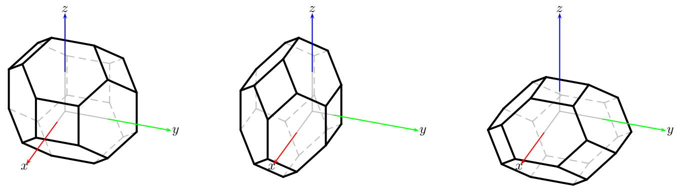

Figure 5: The supports of three box-splines (a) $M_{\Xi_7\setminus\{\boldsymbol{\xi}_1\}}(\cdot + \boldsymbol{\xi}_c)$ (b) $M_{\Xi_7\setminus\{\boldsymbol{\xi}_2\}}(\cdot + \boldsymbol{\xi}_c)$ and (c) $M_{\Xi_7\setminus\{\boldsymbol{\xi}_3\}}(\cdot + \boldsymbol{\xi}_c)$.

Let

$$\boldsymbol{\xi}_1 := \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{\xi}_2 := \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}, \text{ and } \boldsymbol{\xi}_3 := \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}. \quad (16)$$

Figure 5 shows the supports of three box-splines $M_{\Xi\setminus\{\boldsymbol{\xi}_1\}}$, $M_{\Xi\setminus\{\boldsymbol{\xi}_2\}}$, and $M_{\Xi\setminus\{\boldsymbol{\xi}_3\}}$ and the backward differences along corresponding directions of them are the directional derivative kernels $M_{\boldsymbol{\xi}_1}$, $M_{\boldsymbol{\xi}_2}$, and $M_{\boldsymbol{\xi}_3}$, respectively. Note that the supports of three directional derivative kernels are subsets of the support of M_7, therefore we do not need to fetch any additional data to evaluate the gradient, but can re-use those data already fetched for spline evaluation.

To evaluate $M_{\boldsymbol{\xi}_1}$, we computed the BB-coefficients following the method by Kim and Peters [10], which are shown in Table 1 for a reference. After building the BB-form of the polynomial, it is evaluated using de Casteljau method. Note that, while we need to consider only one type of reference tetrahedron to evaluate M_7 thanks to its nice symmetry, we need to consider three types of reference tetrahedra, 000, 011, and 110, due to the lower symmetry of $M_{\boldsymbol{\xi}_1}$. Other three types of tetrahedra, 001, 100, and 111 can each transformed to the types 000, 011, and 110, respectively, since $M_{\boldsymbol{\xi}_1}$ is symmetric with respect to the y- and z-axes. Note that, since $M_{\boldsymbol{\xi}_1}$ is an odd function along the x-axis, we need to negate the value if the corresponding reflection matrix is reflecting with respect to the x-axis.

Due to the symmetry of the three derivative kernels, other two kernels can be evaluated using the same evaluation code for $M_{\boldsymbol{\xi}_1}$, by remapping the tetrahedron type appropriately.

4.3 Empty Space Skipping

Compared to the direct volume rendering method, isosurface rendering does not require evaluating the reconstructed volume field along the whole ray, but only requires to find the nearest intersection of the ray with the isosurface. Therefore we can improve the performance by discarding those blocks that are guaranteed not to include the isosurface, this can be easily done in our scheme by

Table 1: The BB-coefficients of M_{ξ_1} for the tetrahedron of type (top) 000 and 001, (middle) 011 and 100 and (bottom) 110 and 111.

	3	2	1	0	2	1	0	1	0	0	2	1	0	1	0	0	1	0	0	0
	0	1	2	3	0	1	2	0	1	0	0	1	2	0	1	0	0	1	0	0
	0	0	0	0	1	1	1	2	2	3	0	0	0	1	1	2	0	0	1	0
	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	3
(0, 0, 0)		-16	-16	-8	-16	-24	-16	-24	-24	-24	-16	-20	-12	-24	-20	-24	-20	-14	-20	-14
(1, 1, 1)	8	16	16		8	8		4			12	12		6			8			
(1, 1,-1)	8	8			8	4		4			12	4		6			8			
(1,-1, 1)	8	8			8	4		4			4			2						
(1,-1,-1)	8				8			4			4			2						
(-1, 1, 1)	-8	-16	-16	-8	-8	-8	-8	-4	-4	-2	-12	-12	-8	-6	-6	-3	-8	-6	-4	-4
(-1, 1,-1)	-8	-8			-8	-4		-4	-2	-2	-12	-4		-6	-2	-3	-8	-2	-4	-4
(-1,-1, 1)	-8	-8			-8	-4		-4	-2	-2	-4			-2		-1				
(-1,-1,-1)	-8				-8			-4		-2	-4			-2		-1				
(2, 0, 0)	8	16	16	8	16	24	16	24	24	24	16	20	12	24	20	24	20	14	20	14
(-2, 0, 0)	-8																			
(0, 2, 0)			-8	-8		-4	-8	-2	-4	-2		-8	-12	-4	-8	-4	-8	-14	-8	-14
(0,-2, 0)								-2		-2										
(0, 0, 2)			-8	-8		-4	-8	-2	-4	-2		-4	-4	-2	-4	-2	-2	-2	-2	-1
(0, 0,-2)								-2		-2				-2		-2	-2		-2	-1
(0, 2, 2)				-8									-4					-2		-1
(0, 2,-2)																				-1
(2, 0, 2)			8	8		4	8	2	4	2		4	4	2	4	2	2	2	2	1
(2, 0,-2)								2		2				2		2	2		2	1
(2, 2, 0)			8	8		4	8	2	4	2		8	12	4	8	4	8	14	8	14
(2,-2, 0)								2		2										
(3, 1, 1)				8			8		4	2			8		6	3		6	4	4
(3, 1,-1)									2	2					2	3		2	4	4
(3,-1, 1)									2	2						1				
(3,-1,-1)										2						1				
(1, 3, 1)																				
(1, 3,-1)																				
(1, 1, 3)																				
(2, 2, 2)				8									4					2		1
(2, 2,-2)																				1

	3	2	1	0	2	1	0	1	0	0	2	1	0	1	0	0	1	0	0	0
	0	1	2	3	0	1	2	0	1	0	0	1	2	0	1	0	0	1	0	0
	0	0	0	0	1	1	1	2	2	3	0	0	0	1	1	2	0	0	1	0
	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	3
(0, 0, 0)		-16	-16	-8		-12	-12		-8		-16	-20	-12	-12	-14	-8	-20	-14	-14	-14
(1, 1, 1)	8	16	16		12	20	16	14	20	14	12	12		14	12	14	8		8	
(1, 1,-1)	8	8			12	8		14	8	14	12	4		14	4	14	8		8	
(1,-1, 1)	-8	-16	-16	-8	-12	-20	-16	-14	-20	-14	-12	-12	-8	-14	-12	-14	-8	-6	-8	-4
(1,-1,-1)	-8	-8			-12	-8		-14	-8	-14	-12	-4		-14	-4	-14	-8	-2	-8	-4
(-1, 1, 1)	8	8			4	4		2	2	1	4			2		1				
(-1, 1,-1)	8				4			2		1	4			2		1				
(-1,-1, 1)	-8	-8			-4	-4		-2	-2	-1	-4			-2		-1				
(-1,-1,-1)	-8				-4			-2		-1	-4			-2		-1				
(2, 0, 0)			-8	-8		-4	-12		-8			-8	-12	-4	-14	-8	-8	-14	-14	-14
(-2, 0, 0)																				
(0, 2, 0)	8	16	16	8	8	12	12	6	8	4	16	20	12	12	14	8	20	14	14	14
(0,-2, 0)	-8				-8			-6		-4										
(0, 0, 2)			-8	-8			-4					-4	-4		-2		-2	-2	-1	-1
(0, 0,-2)																	-2		-1	-1
(0, 2, 2)			8	8			4					4	4		2		2	2	1	1
(0, 2,-2)																	2		1	1
(2, 0, 2)				-8			-4						-4		-2			-2	-1	-1
(2, 0,-2)																			-1	-1
(2, 2, 0)			8	8		4	12	2	8	4		8	12	4	14	8	8	14	14	14
(2,-2, 0)								-2		-4										
(3, 1, 1)									2	1						1				
(3, 1,-1)										1						1				
(3,-1, 1)									-2	-1						-1				
(3,-1,-1)										-1						-1				
(1, 3, 1)				8									8					6		4
(1, 3,-1)																		2		4
(1, 1, 3)																				
(2, 2, 2)				8			4						4		2			2	1	1
(2, 2,-2)																			1	1

	3	2	1	0	2	1	0	1	0	0	2	1	0	1	0	0	1	0	0	0
	0	1	2	3	0	1	2	0	1	0	0	1	2	0	1	0	0	1	0	0
	0	0	0	0	1	1	1	2	2	3	0	0	0	1	1	2	0	0	1	0
	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	3
(0, 0, 0)		-16	-16	-8		-12	-12		-8			-8	-8		-6			-4		
(1, 1, 1)	8	16	16		12	20	16	14	20	14	16	24	16	20	24	20	24	24	24	24
(1, 1,-1)	-8	-16	-16	-8	-12	-20	-16	-14	-20	-14	-16	-24	-16	-20	-24	-20	-24	-24	-24	-24
(1,-1, 1)	8	8			12	8		14	8	14	8	4		8	4	8	4	2	4	2
(1,-1,-1)	-8	-8			-12	-8		-14	-8	-14	-8	-4		-8	-4	-8	-4	-2	-4	-2
(-1, 1, 1)	8	8			4	4		2	2	1	8	4		4	2	2	4	2	2	2
(-1, 1,-1)	-8	-8			-4	-4		-2	-2	-1	-8	-4		-4	-2	-2	-4	-2	-2	-2
(-1,-1, 1)	8				4			2		1										
(-1,-1,-1)	-8				-4			-2		-1										
(2, 0, 0)			-8	-8		-4	-12		-8			-4	-8		-6			-4		
(-2, 0, 0)																				
(0, 2, 0)			-8	-8			-4					-4	-8		-2			-4		
(0,-2, 0)																				
(0, 0, 2)	8	16	16	8	8	12	12	6	8	4	8	8	8	6	6	4	4	4	3	2
(0, 0,-2)	-8				-8			-6		-4	-8			-6		-4	-4		-3	-2
(0, 2, 2)			8	8			4					4	8		2		2	4	1	2
(0, 2,-2)																	-2		-1	-2
(2, 0, 2)			8	8		4	12	2	8	4		4	8	2	6	4	2	4	3	2
(2, 0,-2)								-2		-4				-2		-4	-2		-3	-2
(2, 2, 0)				-8			-4						-8		-2			-4		
(2,-2, 0)																				
(3, 1, 1)									2	1					2	2		2	2	2
(3, 1,-1)									-2	-1					-2	-2		-2	-2	-2
(3,-1, 1)										1										
(3,-1,-1)										-1										
(1, 3, 1)																		2		2
(1, 3,-1)																		-2		-2
(1, 1, 3)				8																
(2, 2, 2)				8			4				0		8		2			4	1	2
(2, 2,-2)																			-1	-2

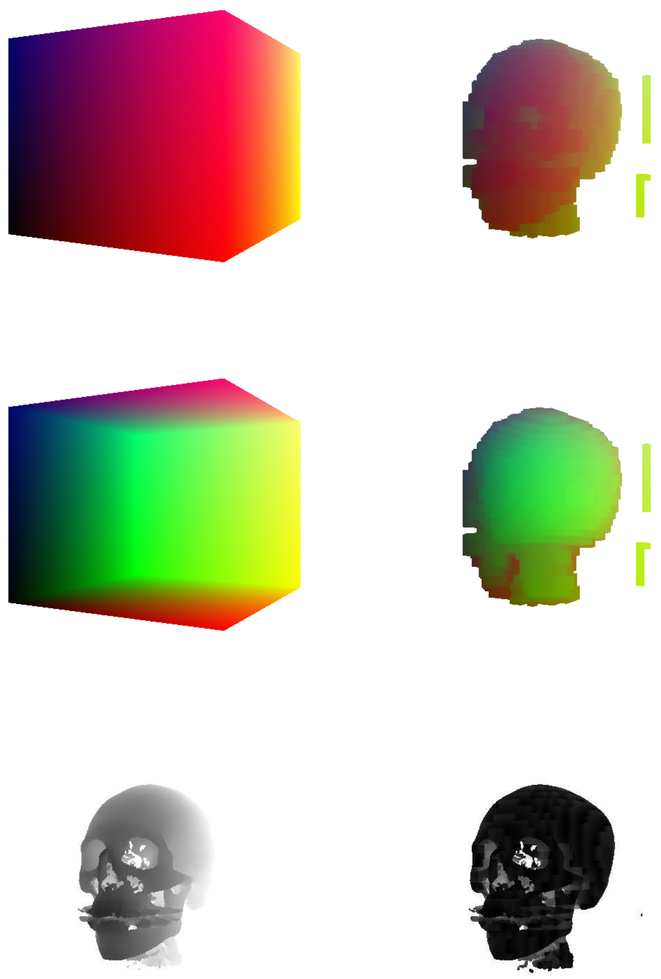
Figure 6: Visualization of ray-casting (left) without and (right) with the empty-space skipping technique. The (top) entry and (middle) exit points for rays are color-coded and (bottom) the number of evaluation is color-coded in grayscale, where bright color denotes large number of computations.

leveraging the BB-form of the polynomial pieces. Specifically, in the preprocessing step, we first build a hierarchical octree based on the min/max values of the BB-forms of all polynomial pieces. At each rendering step, those blocks that do not contain the isolevel are discarded in the vertex shader hence the lengths of rays are shortened significantly. Figure 6 visualizes the performance improvement due to our empty space skipping technique. While the preprocessing step is done only once at the beginning, we implemented it using OpenCL to further improve the overall performance. Note that while the octree is constructed hierarchically, only one fixed level is chosen for rendering and no adaptive technique is used, which is left as one of our future work. (See Table 4.)

Table 2: Test datasets (courtesy of The Volume Library [11]).

dataset	size	cell size (10^{-3})
`MRI-Head` (#1)	128×128×128×2	3.91×3.91×3.13
`VisMale` (#2)	64×128×128×2	6.16×3.89×3.94
`CT-Head` (#3)	128×128× 57×2	3.71×3.91×7.37
`Carp` (#4)	128×128×256×2	1.53×0.76×1.95

Table 3: Performance comparison (fps) of two normal computation schemes.

dataset	finite difference	analytic	speed-up
#1	22.3	39.4	1.77
#2	27.5	47.6	1.73
#3	21.7	35.0	1.61
#4	15.7	24.4	1.55

5 Results and Discussion

Table 2 shows the detailed information about four BCC volume datasets we used for our tests. Our test environment is as follows:

- Intel® Xeon® CPU X5550 @2.67 GHz
- 12GB memory
- Windows 7 Professional (64 bit)
- NVIDIA GeForce GTX 670 (2GB DDR5) with driver version 314.22.

Figure 7 shows the rendered images of three ray-casters and our result shows very similar results with bcc8, while bcc12 results in oversmoothed surface. Refer to Kim [1] for an in-depth analysis of three reconstruction schemes.

Table 3 shows the performance improvement due to our analytic gradient computation. We gain at least 55% improvement for all test datasets. Figure 8 compares the rendering results using two different gradient computation methods. Note that the value computed by the finite difference method is sensitive to the δ value used due to the round-off error (when δ is too small) and the poor approximation (when δ is too large).

For the empty-space skipping technique, we need to be little bit careful for choosing the resolution of min/max blocks. Too low number of blocks, thus larger blocks, may result in inefficient empty-space skipping and too high number of blocks may result in a high overhead for primitives. Table 4 shows the performance for various resolutions of blocks. The optimal resolution for our test datasets is 32^3 and our method results in 360–700% performance improvements.

Table 5 shows the overall performance comparison of various BCC ray-casters. Our method (b) improves the performance by

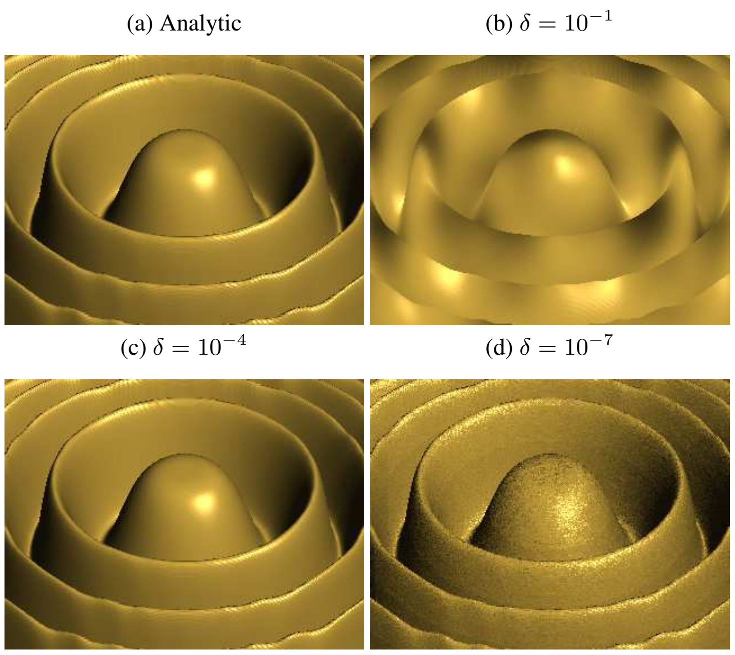

Figure 8: Comparison of gradient computation methods. (a) the analytic and (b) – (d) the finite difference methods with various δ values.

Table 4: Performance comparison (fps) for difference resolutions of blocks.

dataset	not used	8^3	16^3	32^3	64^3	speed-up
#1	8.6	16.8	30.6	39.4	38.0	≤ 4.6
#2	7.6	26.6	41.2	47.6	41.0	≤ 6.3
#3	6.4	17.3	31.3	35.0	31.6	≤ 5.5
#4	3.2	11.0	17.9	24.4	25.5	≤ 8.0

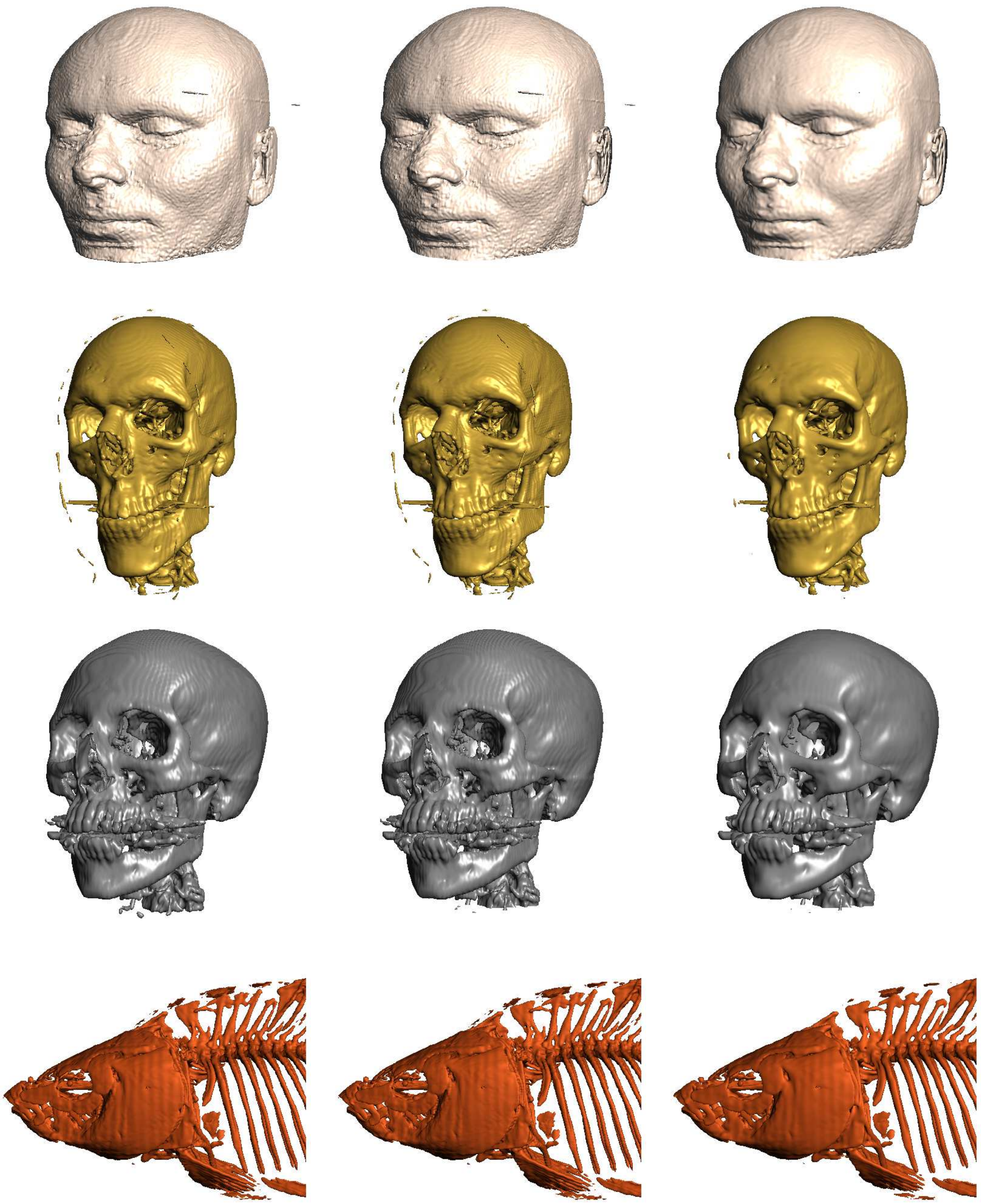

Figure 7: Rendered images of three methods; (left) our method, (middle) bcc8, and (right) bcc12 for datasets #1, #2, #3, and #4 from top to bottom.

Table 5: Performance comparison for (a) previous method [2], (b) our method, (c) bcc8 [6], and (d) bcc12 [7].

dataset	(a)	(b)	(c)	(d)	(b)/(a)	(b)/(c)	(b)/(d)
#1	8.3	39.4	11.3	34.1	4.75	3.49	1.16
#2	7.6	47.6	10.3	34.0	6.26	4.62	1.40
#3	6.5	35.0	8.1	22.8	5.38	4.32	1.53
#4	3.2	24.4	4.6	13.0	7.63	5.30	1.88

375–663% from the previous one (a) where the finite difference method is used for gradient computation and no empty-space skipping technique is used. Our method also outperforms other two ray-casters. But note that, while it requires some additional work, the two optimization techniques we adopted here can be also applied to other ray-casters too.

6 Conclusion and Future Work

By incorporating two optimization techniques, we can improve the overall performance significantly while keeping the superior visual quality of our previous method. This enables our GPU isosurface ray-casting practical for modern graphics hardware. While our optimization techniques are limited to isosurface rendering, we plan to investigate an efficient direct volume ray-casting kernel of BCC volume datasets. Also we plan to handle large scale datasets and investigate an adaptive empty-space skipping technique to further improve the rendering performance.

Acknowledgments

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0024007).

References

[1] M. Kim, “Quartic box-spline reconstruction on the BCC lattice,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 2, pp. 319–330, Feb. 2013.

[2] M. Kim and Y. Lee, “Real-time BCC volume isosurface ray casting on the GPU,” *Journal of the Korea Computer Graphics Society*, vol. 18, no. 4, pp. 25–34, Dec. 2012.

[3] M. Kim, “GPU isosurface raycasting of FCC datasets,” *Graphical Models*, vol. 75, no. 2, pp. 90–101, Mar. 2013.

[4] D. P. Petersen and D. Middleton, “Sampling and reconstruction of wave-number-limited functions in N-dimensional euclidean spaces,” *Information and Control*, vol. 5, no. 4, pp. 279–323, 1962.

[5] A. Entezari, R. Dyer, and T. Möller, “Linear and cubic box splines for the body centered cubic lattice,” in *Proceedings of the IEEE Conference on Visualization*. IEEE Computer Society, 2004, pp. 11–18.

[6] B. Finkbeiner, A. Entezari, D. Van De Ville, and T. Möller, “Efficient volume rendering on the body centered cubic lattice using box splines,” *Computers & Graphics*, vol. 34, no. 4, pp. 409–423, Aug. 2010.

[7] B. Csébfalvi and M. Hadwiger, “Prefiltered B-spline reconstruction for hardware-accelerated rendering of optimally sampled volumetric data,” *Workshop Vision, Modeling, and Visualization*, pp. 325–332, 2006.

[8] M. Kim, A. Entezari, and J. Peters, “Box spline reconstruction on the face-centered cubic lattice,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1523–1530, Nov.-Dec. 2008.

[9] C. de Boor, K. Höllig, and S. Riemenschneider, *Box splines*. Springer-Verlag New York, Inc., 1993.

[10] M. Kim and J. Peters, “Fast and stable evaluation of box-splines via the BB-form,” *Numerical Algorithms*, vol. 50, no. 4, pp. 381–399, Apr. 2009.

[11] S. Roettger, “Volume library (online),” Jan. 2012. [Online]. Available: http://www9.informatik.uni-erlangen.de/External/vollib

[12] J. Kruger and R. Westermann, “Acceleration techniques for gpu-based volume rendering,” in *Visualization, 2003. VIS 2003. IEEE*, Oct. 2003, pp. 287–292.