GPU를 이용한 실시간 BCC 볼륨 등가면 레이 캐스팅

김민호 ◦ 이영준

서울시립대학교 컴퓨터과학부

{minhokim,nedis}@uos.ac.kr

Real-time BCC Volume Isosurface Ray Casting on the GPU

Minho Kim ◦ Young-joon Lee

School of Computer Science, University of Seoul

요 약

본 논문에서는 BCC (body-centered cubic) 볼륨 데이터의 등가면을 GPU(graphics processing unit)에서 실시간으로 레이 캐스팅 렌더링하는 방법을 제시한다. 우선 준-보합 전치필터를 적용한 후 7-방향 박스-스플라인 필터를 기반으로 하여 4차 스플라인 함수로 볼륨데이터를 복구한다. 그래픽스 하드웨어에서 실시간 렌더링을 하기 위해, 참조테이블 및 조건분기를 사용하지 않고 데이터 인출시의 비용을 줄이도록 쉐이더 코드를 최적화하였다. 본 방법을 기존의 BCC 레이 캐스팅과 비교해 본 결과, 비슷한 성능의 기존 방법에 비해 렌더링 속도는 20% 이상 빨라졌고 렌더링 이미지의 품질은 가장 좋았다.

Abstract

This paper presents a real-time GPU (graphics processing unit) ray casting scheme for rendering isosurfaces of BCC (body-centered cubic) volume datasets. A quartic spline field is built using the 7-direction box-spline filter accompanied with a quasi-interpolation prefilter. To obtain an interactive rendering speed on the graphics hardware, the shader code was optimized to avoid lookup table and conditional branches and to minimize data fetch overhead. Compared to previous implementations, our work outperforms the comparable one by more than 20% and the rendering quality is superior than others.

키워드: GPU, 레이 캐스팅, BCC 볼륨데이터, 박스-스플라인
Keywords: GPU, Ray Casting, BCC Datasets, Box-Splines

1 Introduction

While the Cartesian lattice, also known as the cubic lattice, has been widely used for sampling and storing volumetric data due to its separability and hardware-friendly nature, it was shown that the Cartesian lattice is very inefficient compared to other regular lattices such as the BCC and FCC (face-centered cubic) lattices. Here *inefficiency* means that the Cartesian lattice requires the largest number of samples to reconstruct the original volumetric signal without aliasing if it is isotropic and band-limited. Recently, other regular lattices have been investigated as alternative sampling lattices for volumetric datasets and several reconstruction filters have been proposed showing their superiority. From the practical point of view, rendering the reconstructed volume on non-Cartesian lattices is required for several applications. Thanks to the programmability of modern graphics hardware, several researchers have implemented interactive GPU ray casters for those datasets.

This paper presents an real-time GPU isosurface ray caster for BCC volume datasets extending the work by Lee and Kim [1]. The 7-direction box-spline reconstruction filter accompanied with a quasi-interpolation prefilter is used for volume reconstruction. By optimizing the evaluation procedure for graphics hardware, our ray caster shows interactive performance for real world volume

datasets. When compared to previous implementations, our ray caster outperforms the comparable one and the image quality is superior than others.

2 Previous Work

Extending Shannon's theorem to to the multi-dimensional case, Petersen and Middleton [2] showed that the optimal sampling lattice is the dual of the densest sphere packing lattice for a band-limited and isotropic input signal. It has been shown that the FCC lattice is the densest sphere packing lattice [3], and therefore its dual, the BCC lattice, is the optimal 3-dimensional sampling lattice.

Since then various reconstruction filters on the BCC lattice have been investigated [4, 5], but they do not reflect the lattice structure properly. Recently, some reconstruction filters based on box-splines have been proposed leveraging the structural properties of the BCC lattice. Entezari *et al.* [6] proposed the linear and quintic box-splines and showed their superiority in volume reconstruction. Later, based on the efficient evaluation scheme by Entezari *et al.* [7], Finkbeiner *et al.* [8] implemented a GPU real-time BCC volume ray caster using those box-spline filters. Extending the technique proposed by Sigg and Hadwiger [9], Csébfalvi and Hadwiger [10] proposed the tri-cubic B-spline filter with a GPU ray caster showing its superb performance due to the hardware-friendly separable structure of the B-spline. But the reconstructed field shows too much smoothing and looses many details. Lastly, Kim [11] proposed the 7-direction quartic box-spline that shows the best reconstruction quality and fastest evaluation performance on the CPU compared to the previous schemes. But no GPU ray caster based on the 7-direction box-spline has been proposed yet.

3 Background

In this section we briefly review the BCC lattice and box-splines. Refer to de Boor *et al.* [12] for more details about box-splines.

3.1 The BCC Lattice

An n-*dimensional lattice* $\mathcal{L}_n$ is the set of points generated by an $n \times n$ (invertible) square *generator matrix* as all integer linear combinations of its column vectors:

$$\mathcal{L}_n := \boldsymbol{G}\mathbb{Z}^n = \{\boldsymbol{G}\boldsymbol{j} : \boldsymbol{G} \in \mathbb{R}^{n\times n}, \operatorname{rank}\boldsymbol{G} = n, \boldsymbol{j} \in \mathbb{Z}^n\}. \quad (1)$$

Its *dual* (or *reciprocal*) lattice $\mathcal{L}_n^*$ is defined as

$$\mathcal{L}_n^* := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \cdot \boldsymbol{u} \in \mathbb{Z}, \forall \boldsymbol{u} \in \mathcal{L}_n\}, \quad (2)$$

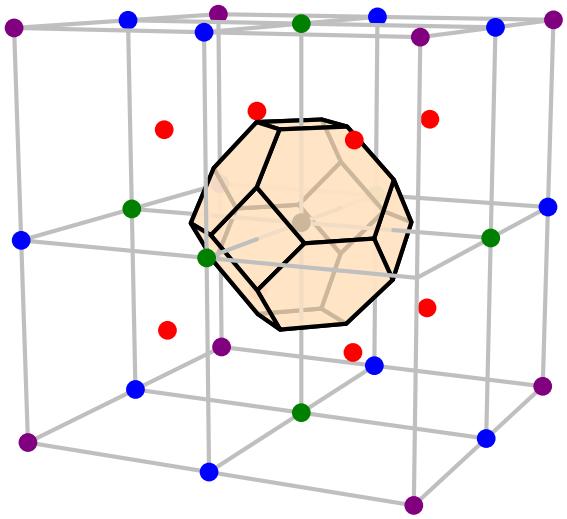


Figure 1: The BCC Lattice and its Voronoi cell. Points in $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_5$ are color-coded in red, green, blue, and violet, respectively.

and its generator matrix is $\boldsymbol{G}^{-t}$ [3]. In multi-dimensional signal processing, sampling a signal on a lattice $\mathcal{L}_n$ is equivalent to replicating its spectrum on the dual lattice $\mathcal{L}_n^*$ in the frequency domain [13].

The BCC lattice can be constructed as a superset of the 3-dimensional Cartesian lattice by inserting additional lattice points at the center of each Cartesian grid cell (Figure 1). Formally, we can define the BCC lattice with generator matrix $\boldsymbol{G}_{\text{bcc}}$ as

$$\mathbb{Z}_{\text{bcc}} := \boldsymbol{G}_{\text{bcc}}\mathbb{Z}^3, \quad \text{where } \boldsymbol{G}_{\text{bcc}} := \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}. \quad (3)$$

Note that $\det \boldsymbol{G}_{\text{bcc}} = 4$. The BCC lattice is the dual of the FCC lattice, the optimal 3D sphere packing lattice, and hence is the optimal 3D sampling lattice [2].

On the BCC lattice, we can group the lattice neighbors of a point into *shells* [3] according to their distances from that point. Let $\mathcal{S}_k$ be the k-th shell of the origin. For $\mathbb{Z}_{\text{bcc}}$, the Cartesian coordinates of the first three shells are as follows:

$$\mathcal{S}_0 = \{(0,0,0)\}, \quad (4)$$

$$\mathcal{S}_1 = \{(\pm1,\pm1,\pm1)\}, \text{ and} \quad (5)$$

$$\mathcal{S}_2 = \{(\pm2,0,0),(0,\pm2,0),(0,0,\pm2)\}, \quad (6)$$

with squared distances 0, 3 and 4, respectively. In Figure 1, $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$ and $\mathcal{S}_5$ are color-coded in red, green, blue, and violet, respectively. Considering non-parallel vectors corresponding to $\mathcal{S}_1$ and

$\mathcal{S}_2$, we get two sets of vectors,

$$\boldsymbol{\Xi}_1 := \begin{bmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \quad \text{and} \tag{7}$$

$$\boldsymbol{\Xi}_2 := \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \tag{8}$$

which are the building blocks of the three box-spline filters we consider.

3.2 Box-Splines

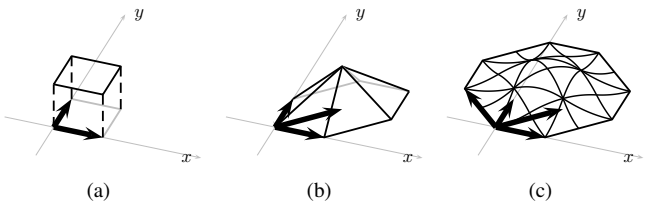
In this and the following text, a matrix also denotes a *multiset*, a set of column vectors allowing multiplicity, depending on the context.

A box-spline is a piecewise polynomial with a finite support and certain continuity and is uniquely defined by a *direction matrix*. Given an $n \times m$ (usually $n \leq m$) direction matrix $\boldsymbol{\Xi}$, a box-spline $M_{\boldsymbol{\Xi}}$ can be constructed recursively by taking consecutive directional convolutions along each column direction (Figure 2). In other words, with the base case ($n = m$)

$$M_{\boldsymbol{\Xi}}(\boldsymbol{x}) := \frac{1}{|\det \boldsymbol{\Xi}|} \chi_{\boldsymbol{\Xi}}(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^n \tag{9}$$

where $\boldsymbol{\Xi}$ is invertible and $\chi_{\boldsymbol{\Xi}}(\boldsymbol{x})$ is the characteristic function on the half-open parallelepiped $\boldsymbol{\Xi}[0,1)^m$, a box-spline defined by the direction matrix $\boldsymbol{\Xi} \cup \{\boldsymbol{\xi}\}$ can be recursively constructed as

$$M_{\boldsymbol{\Xi} \cup \{\boldsymbol{\xi}\}}(\boldsymbol{x}) := \int_0^1 M_{\boldsymbol{\Xi}}(\boldsymbol{x} - t\boldsymbol{\xi}) dt, \quad \boldsymbol{\xi} \in \mathbb{R}^n. \tag{10}$$



(a) (b) (c)

Figure 2: Construction of box-splines with directions (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) $\left[\begin{smallmatrix} 1 & 0 & 1 & -1 \\ 0 & 1 & 1 & 1 \end{smallmatrix}\right]$.

Given a discrete dataset sampled on the Cartesian lattice, a continuous spline $s(\boldsymbol{x})$ can be built by a convolution of the dataset $V : \mathbb{Z}^n \mapsto \mathbb{R}$ and a box-spline filter $M_{\boldsymbol{\Xi}}$ as follows:

$$s(\boldsymbol{x}) := V * M_{\boldsymbol{\Xi}} := \sum_{\boldsymbol{j} \in \mathbb{Z}^n} V(\boldsymbol{j}) M_{\boldsymbol{\Xi}}(\boldsymbol{x} - \boldsymbol{j}). \tag{11}$$

But such a construction does not result in a good approximation and in most cases we can obtain better reconstruction by applying a discrete quasi-interpolation prefilter q on the dataset:

$$(V \star q) * M_{\boldsymbol{\Xi}}, \tag{12}$$

where $\star$ denotes the *discrete convolution*

$$(V \star q)(\boldsymbol{k}) := \sum_{\boldsymbol{j} \in \mathbb{Z}^n} V(\boldsymbol{j}) q(\boldsymbol{k} - \boldsymbol{j}). \tag{13}$$

Then the reconstructed spline annihilates all the lower terms of the Taylor expansion of the input signal thus reducing approximation error. The maximal approximation order is $\rho(\boldsymbol{\Xi})$ where $\rho(\boldsymbol{\Xi})$ is defined as the minimum number of directions of $\boldsymbol{\Xi}$ such that, when they are removed from $\boldsymbol{\Xi}$, the remaining columns in $\boldsymbol{\Xi}$ do not span $\mathbb{R}^n$ [12]. For the procedure to compute discrete quasi-interpolation prefilters, refer to de Boor *et al.*[12]. Note that while de Boor *et al.*[12] discuss box-splines based on shifts on the Cartesian lattice, box-splines on non-Cartesian lattices can be easily obtained by change-of-variables [14].

4 The Box-Spline Filters on the BCC Lattice

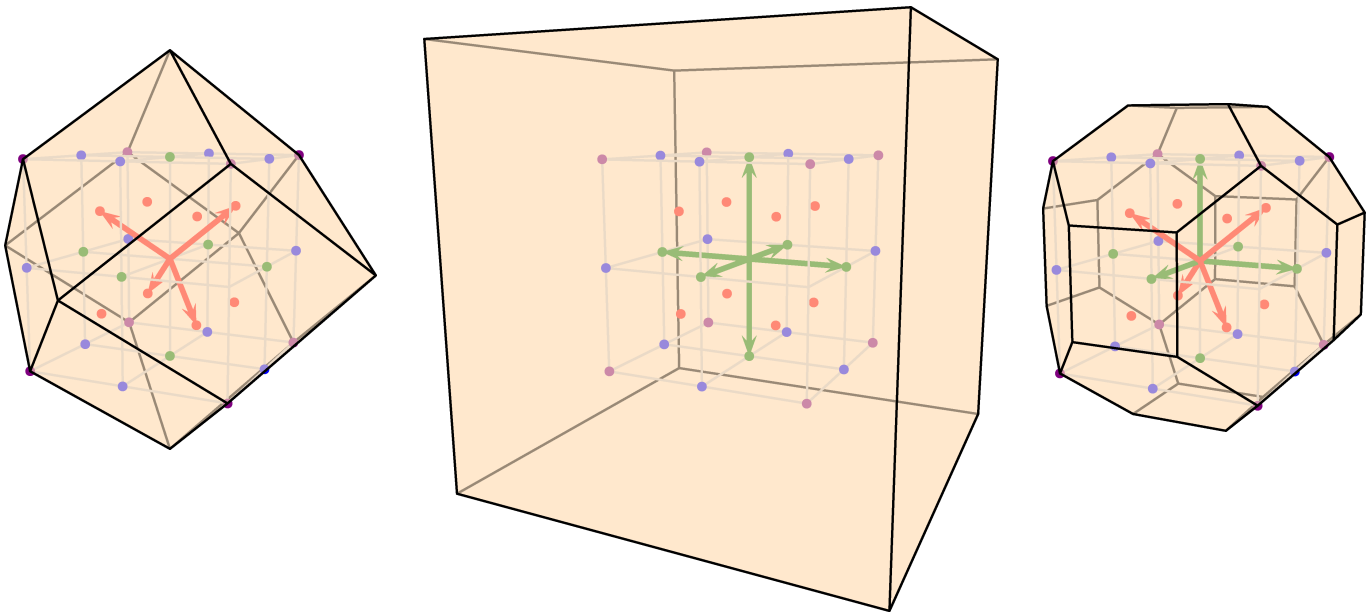
In this section we discuss three box-spline filters defined on the BCC lattice, namely, the symmetric quintic box-spline filter (bcc8), the tri-cubic B-spline filter (bcc12), and the symmetric quartic box-spline filter (bcc7), all of which have the same approximation order of four. Refer to Kim [11] for in-depth analysis and comparison of the filters.

4.1 Symmetric Quintic Box-Spline on the BCC Lattice

Based on the idea that the BCC lattice is the 3-dimensional extension of the hexagonal lattice [3], Entezari *et al.* [6] proposed the the symmetric quintic box-spline, bcc8, on the BCC lattice as an extension of the symmetric quartic box-spline on the hexagonal lattice. The box-spline is defined by the direction matrix

$$\begin{bmatrix} \boldsymbol{\Xi}_1 & \boldsymbol{\Xi}_1 \end{bmatrix}, \tag{14}$$

i.e., the four (non-parallel) directions, each repeated two times, corresponding to the eight lattice points in $\mathcal{S}_1$ (5), as shown in Figure 3(a). Its support is the shape of a rhombic dodecahedron, with its volume 128, and therefore the stencil size (the maximum number of samples required to evaluate an arbitrary point on the spline) is $128/|\det \boldsymbol{G}_{\text{bcc}}| = 32$ [12]. The (total) polynomial degree is $8 - 3 = 5$, and hence is quintic. Since any three directions in $\boldsymbol{\Xi}_1$



(a) (b) (c)

Figure 3: The directions and supports of three box-spline filters on the BCC lattice; (a) the symmetric quintic box-spline (bcc8) [6], (b) the tri-cubic B-spline (bcc12) [10], and (c) the symmetric quartic box-spline (bcc7) [11].

span $\mathbb{R}^3$, $\rho(\boldsymbol{\Xi}_1) = 2$; therefore, $\rho(\boldsymbol{\Xi}_8) = 4$ is the approximation order of the quintic spline.

Entezari *et al.* [7] proposed an efficient evaluation algorithm of splines based on the quintic box-spline by leveraging the repeated directions of the box-spline. Finkbeiner *et al.* [8] optimized it for graphics hardware and implemented a GPU ray caster.

4.2 Tri-Cubic B-Spline on the BCC Lattice

The tri-cubic B-spline on the BCC lattice [10], bcc12, is defined by the twelve directions

$$\begin{bmatrix} \boldsymbol{\Xi}_2 & \boldsymbol{\Xi}_2 & -\boldsymbol{\Xi}_2 & -\boldsymbol{\Xi}_2 \end{bmatrix}, \tag{15}$$

i.e., the three directions corresponding to the lattice points in $\mathcal{S}_2$ (6), each with multiplicity four (Figure 3(b)). Its total polynomial degree is $12 - 3 = 9$, and the approximation order is four since $\rho(\boldsymbol{\Xi}_2) = 1$; hence, $\rho(\boldsymbol{\Xi}_{12}) = 4$. In addition, the volume of the support is 512; therefore, its stencil size is $512/|\det \boldsymbol{G}_{\text{bcc}}| = 128$. Note that tensor-product B-splines with uniform knots are special cases of box-splines.

4.3 Symmetric Quartic Box-Spline on the BCC Lattice

The symmetric quartic box-spline [11], bcc7, is defined by the seven directions

$$\begin{bmatrix} \boldsymbol{\Xi}_1 & \boldsymbol{\Xi}_2 \end{bmatrix} \tag{16}$$

corresponding the non-parallel directions in $\mathcal{S}_1 \cup \mathcal{S}_2$ (Figure 3(c)). Its total polynomial degree is $7 - 3 = 4$ and the approximation order is four. The support is the shape of the truncated rhombic dodecahedron and the volume of the support is 120; therefore, its stencil size is 30, two less than that of bcc8. Refer to Kim [11] for more details about bcc7.

To evaluate quartic splines generated by bcc7, we need to investigate the polynomial structure as done by Kim [11]. There are nine knot planes induced by the generator matrix of bcc7 (Figure 4). The three axis-aligned knot planes first decompose the whole space into unit cubes. Then each cube is further decomposed into six tetrahedra, and therefore there are total 24 shift-invariant tetrahedral polynomial pieces. How each cube is decomposed depends on the location of the cube, more specifically, the *parity* (modulo 2 of the lower corner coordinates) of the cube as in Figure 5. In general, we need to consider separate stencil and evaluation procedure for each polynomial type, but thanks to the nice symmetry of bcc7 all the tetrahedra are congruent (up to reflection) and therefore we only need to consider evaluation for one *reference* polynomial piece.

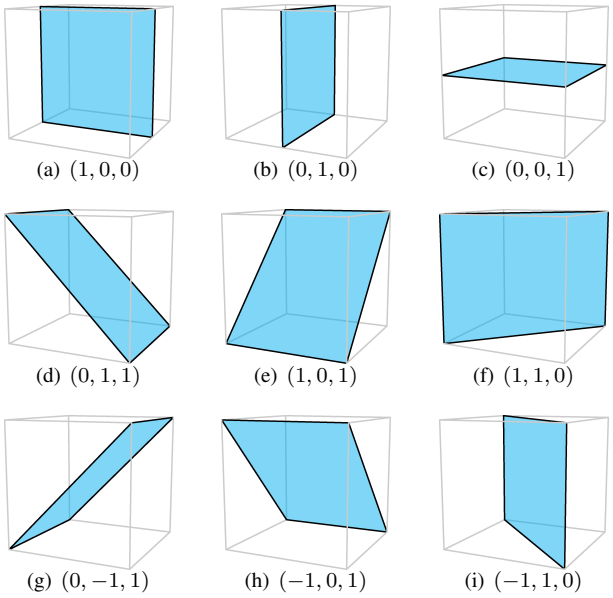


Figure 4: Nine knot planes, with their respective normals, induced by the generator matrix of bcc7.

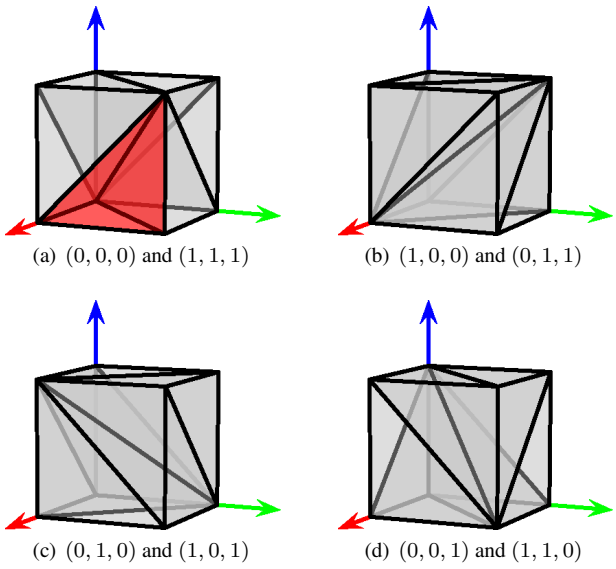


Figure 5: Polynomial structure of quartic splines. There are four types of cube decomposition based on the parity of each cube. Each cube is further decomposed into six tetrahedra. The red tetrahedron of the cube with its parity $(0,0,0)$ in (a) is the 'reference tetrahedron.'

We picked the tetrahedron with its vertices

$$\{(0,0,0),(1,1,1),(1,0,0),(1,1,0)\} \tag{17}$$

as the reference tetrahedron (red tetrahedron in Figure 5(a)). Figure 6 shows the stencil for the reference tetrahedron. After fetching those 30 data, we can construct the polynomial in either BB(Bernstein-Bézier)-form or the power form and evaluate it. The supplementary material of Kim [11] provides necessary formula for both.

5 Real-Time BCC Volume Isosurface Ray Casting on the GPU

Since we can evaluate any input in the reference tetrahedron, we need to figure out how to transform input in other polynomial type to the reference tetrahedron. Let

$$p = p(\boldsymbol{x}) := \lfloor \boldsymbol{x} \rfloor \text{ modulo } 2 \tag{18}$$

be the parity of the cube that contains the input $\boldsymbol{x} \in \mathbb{R}^3$. As can be seen in Figure 5, eight types of cubes can be transformed to the *reference cube* (the cube that contains the reference tetrahedron, Figure 5(a)) via an appropriate reflection R_p following the translation by p

$$\dot{\boldsymbol{x}} = R_p\left(\boldsymbol{x} - \lfloor \boldsymbol{x} \rfloor - p(\boldsymbol{x})\right) \tag{19}$$

where

$$R_{000} := \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix}, R_{001} := \begin{bmatrix} 1&0&0\\0&1&0\\0&0&-1 \end{bmatrix}, R_{010} := \begin{bmatrix} 1&0&0\\0&-1&0\\0&0&1 \end{bmatrix},$$

$$R_{100} := \begin{bmatrix} -1&0&0\\0&1&0\\0&0&1 \end{bmatrix}, R_{111} := \begin{bmatrix} -1&0&0\\0&-1&0\\0&0&-1 \end{bmatrix}, R_{110} := \begin{bmatrix} -1&0&0\\0&-1&0\\0&0&1 \end{bmatrix},$$

$$R_{101} := \begin{bmatrix} -1&0&0\\0&1&0\\0&0&-1 \end{bmatrix}, \text{ and } R_{011} := \begin{bmatrix} 1&0&0\\0&-1&0\\0&0&-1 \end{bmatrix}. \tag{20}$$

Now that $\dot{\boldsymbol{x}}$ is in the reference cube, we need to transform it such that it is inside the reference tetrahedron. We first need to figure out in which of six tetrahedra in Figure 5(a) $\dot{\boldsymbol{x}}$ is, which can be done by membership tests against three knot planes (g), (h), (i) in Figure 4. Then the parity of tetrahedron is defined as

$$t = t(\dot{\boldsymbol{x}}) := (\chi(\dot{\boldsymbol{x}} \cdot \boldsymbol{n}_i), \chi(\dot{\boldsymbol{x}} \cdot \boldsymbol{n}_h), \chi(\dot{\boldsymbol{x}} \cdot \boldsymbol{n}_g)) \tag{21}$$

where $\boldsymbol{n}_i$, $\boldsymbol{n}_h$, and $\boldsymbol{n}_g$ are the normals of the corresponding knot planes in Figure 4 and

$$\chi(x) := \begin{cases} 1 & (x >= 0) \\ 0 & (x < 0) \end{cases}, \quad x \in \mathbb{R}. \tag{22}$$

Then $\dot{\boldsymbol{x}}$ can be transformed into the reference tetrahedron via appropriate permutation matrix P_t as follows:

$$\ddot{\boldsymbol{x}} := P_t \dot{\boldsymbol{x}} \tag{23}$$

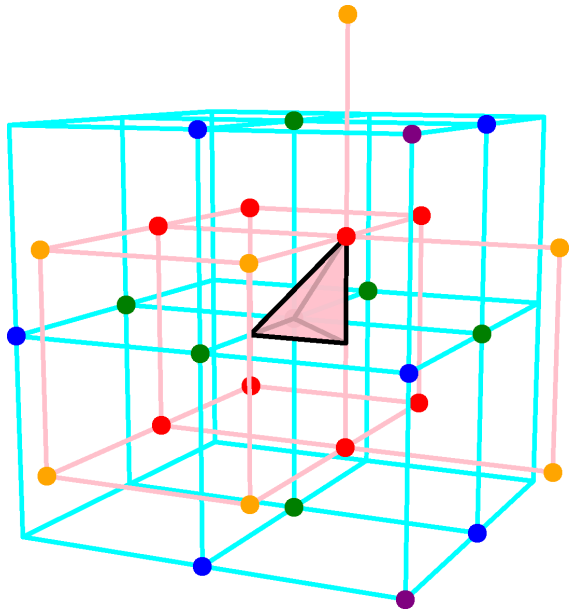


Figure 6: The stencil (30 colored points) of the bcc7 for the reference tetrahedron (pink). The points colored in red, green, blue, yellow, and violet belong to $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, $\mathcal{S}_4$, and $\mathcal{S}_5$, respectively.

where

$$P_{000} := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, P_{001} := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, P_{011} := \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

$$P_{100} := \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, P_{110} := \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \text{ and } P_{111} := \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}. \quad (24)$$

Finally, we fetch 30 volume data

$$\left\{ c_j = V\left(\lfloor x \rfloor + p + (P_t R_p)^{-1} \mathcal{J}_7(j) \right) \right\}_{j=1}^{30}, \quad (25)$$

where $\mathcal{J}_7$ is the stencil of the reference tetrahedron, and evaluate the spline at $\ddot{\boldsymbol{x}}$ analytically using either the BB-form or the power form [11]. Note that

$$(P_t R_p)^{-1} = R_p P_t^T. \quad (26)$$

Algorithm 1 shows the overall evaluation procedure.

Algorithm 1 Pseudocode for evaluation of a quartic spline.

function EVALUATEQUARTIC(V, $\boldsymbol{x}$)
  $p \leftarrow \lfloor \boldsymbol{x} \rfloor$ modulo 2
  $\dot{\boldsymbol{x}} \leftarrow R_p(\boldsymbol{x} - \lfloor \boldsymbol{x} \rfloor - p)$
  $t \leftarrow (\chi(\dot{\boldsymbol{x}} \cdot \boldsymbol{n}_i), \chi(\dot{\boldsymbol{x}} \cdot \boldsymbol{n}_h), \chi(\dot{\boldsymbol{x}} \cdot \boldsymbol{n}_g))$
  $\ddot{\boldsymbol{x}} \leftarrow P_t \dot{\boldsymbol{x}}$
  for $j = 1$ to 30 **do**
    $c_j \leftarrow V(\lfloor \boldsymbol{x} \rfloor + p + (P_t R_p)^{-1} \mathcal{J}_7(j))$
  end for
  Evaluate the spline at $\ddot{\boldsymbol{x}}$.
end function

5.1 Evaluation on the GPU

While the evaluation procedure can be efficiently implemented on the CPU using lookup tables and conditional branches, such overheads degrade performance severely on the GPU. Figure 7 shows our implementation in GLSL (OpenGL Shading Language) that avoids both lookup tables and conditional branches. Further, we optimized the fetching process by rearranging the order of data fetches. Namely, to avoid matrix-vector multiplication for each data fetch, we reorder them such that adjacent data are axis-aligned and can be done with a vector addition.

6 Results and Discussion

While analytic BCC volume dataset can be easily generated, there is no real world BCC dataset yet due to the lack of such an equipment for them. So we obtained BCC datasets as subsets of Cartesian volume datasets provided by "vollib library" [15].

For all three GPU implementations, a simple fixed-step ray marching method, where the isosurface is detected by sign change, is used. For normal computation for lighting, the central difference method is used. In other words, the only difference among three implementations is the evaluation procedure and therefore the overall rendering speed is mainly differentiated by the evaluation performance. The evaluation procedures for bcc12 and bcc8 are based on Finkbeiner *et al.* [8] and Csébfalvi and Hadwiger [10], respectively.

Table 1 shows the rendering speed for three ray casters. The performance of our ray caster (bcc7) is $21 - 22\%$ faster than bcc8. While bcc12 excels in rendering speed others by more than twice, it looses too many details of the original dataset due to the oversmooting aliasing [16] (Figure 8 and Figure 9). When we compare the image quality of bcc8 and bcc7, while both show very close results for real datasets (Figure 9), bcc7 shows marginal improvement than bcc8 as can be seen in (Figure 8). Refer to Kim [11] for detailed analysis and comparison of three reconstruction schemes.

7 Conclusion and Future Work

We implemented a real-time GPU isosurface volume ray caster for BCC datasets. While it is not the fastest ray caster, it outperforms the comparable method and provides best rendering quality. As future work we plan to improve the performance by adopting the techniques used by Kim [17]; analytic normal computation and efficient empty space skipping.

```
#define FETCH(var)  var = texelFetch(volume_tex, o, 0).r;
#define IT0 itv1.x
#define IT1 itv1.y
#define IT3 itv1.z
#define IT4 itv2.x
#define IT6 itv2.y
#define IT7 itv2.z
    float   c[30];
    ivec3   o;
    vec3    xtet;
    ivec3   xlow = ivec3(p_in);
    ivec3   parity = ivec3(xlow.x&0x01, xlow.y&0x01, xlow.z&0x01);
    o = ivec3(xlow) + parity;
    ivec3   Rc = ivec3(1,1,1)-2*parity;
    vec3    xc = Rc*(p_in-o);
    int itet = (int(xc.y >= xc.x)<<2) + (int(xc.z >= xc.x)<<1) + int(xc.z >= xc.y);
    ivec3   itv1 = ivec3(itet==0, itet==1, itet==3);
    ivec3   itv2 = ivec3(itet==4, itet==6, itet==7);
    xtet = IT0*xc.xyz + IT1*xc.xzy + IT3*xc.zxy + IT4*xc.yxz + IT6*xc.yzx + IT7*xc.zyx;
    ivec3   R[3] = ivec3[](
                        2*ivec3(Rc.x*(IT0+IT1), Rc.y*(IT4+IT6), Rc.z*(IT3+IT7)),
                        2*ivec3(Rc.x*(IT3+IT4), Rc.y*(IT0+IT7), Rc.z*(IT1+IT6)),
                        2*ivec3(Rc.x*(IT6+IT7), Rc.y*(IT1+IT3), Rc.z*(IT0+IT4)));
                    FETCH(c01); o -= R[0];      FETCH(c11); o += (R[0]<<1); FETCH(c10);
    o -= R[1];      FETCH(c21); o -= R[0];      FETCH(c13); o += (R[1]<<1); FETCH(c12);
    o += R[0];      FETCH(c20); o -= R[2];      FETCH(c30); o -= R[1];      FETCH(c19);
    o -= R[0];      FETCH(c15); o += R[1];      FETCH(c17); o += (R[2]<<1); FETCH(c16);
    o += R[0];      FETCH(c29); o -= R[1];      FETCH(c18); o -= R[0];      FETCH(c14);
    o += ((-R[0]+R[1]-3*R[2])>>1);  FETCH(c07);
    o -= R[1];      FETCH(c09); o += R[0];      FETCH(c05); o += (R[1]<<1); FETCH(c27);
    o -= R[1];      FETCH(c03); o += (R[2]<<1); FETCH(c28); o -= R[2];      FETCH(c02);
    o += R[1];      FETCH(c26); o -= (R[1]<<1); FETCH(c04); o -= R[0];      FETCH(c08);
    o += R[1];      FETCH(c06); o += (R[0]<<1); FETCH(c22); o -= R[2];      FETCH(c23);
    o -= R[1];      FETCH(c25); o += R[2];      FETCH(c24);
```

Figure 7: GLSL code snippet for Algorithm 1 (excluding evaluation step).

Acknowledgment

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 and Y. joon Lee, "GPU raycasting of BCC volume datasets," *Proceedings of KCGS Conference*, pp. 75–76, 2012.

[2] 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.

[3] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*, 3rd ed. New York, NY, USA: Springer-Verlag New York, Inc., 1998.

[4] S. Matej and R. Lewitt, "Efficient 3D grids for image reconstruction using spherically-symmetric volume elements," *Nuclear Science, IEEE Transactions on*, vol. 42, no. 4, pp. 1361–1370, Aug. 1995.

[5] T. Theuß l, T. Möller, and M. E. Gröller, "Optimal regular volume sampling," *Proceedings of the conference on Visualization '01*, pp. 91–98, 2001.

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

[7] A. Entezari, D. Van De Ville, and T. Möller, "Practical box splines for reconstruction on the body centered cubic lattice,"

Table 1: Rendering speed (fps: frames per second) of BCC datasets.

Experimental system: Intel® Core™ i7 860 @2.80 GHz, Windows 7 Professional (64 bit), 8GB memory, NVIDIA GeForce GTX 460.

Dataset	Size	Image size	bcc12	bcc8	bcc7	bcc7/bcc8	bcc7/bcc12
`CT-Head`	$128 \times 128 \times 56 \times 2$	512×512	26.5358	10.4807	12.7021	1.2120	0.4787
`Carp` (part)	$64 \times 64 \times 128 \times 2$	512×512	16.0361	6.7642	8.2587	1.2210	0.5150
`MRI-Head`	$128 \times 128 \times 128 \times 2$	512×512	35.2842	13.6075	16.4923	1.2120	0.4674
`VisMale`	$64 \times 128 \times 128 \times 2$	512×512	36.5782	13.5530	16.9610	1.2515	0.4637
ML	$39 \times 39 \times 39 \times 2$	480×320	33.7487	13.7503	16.8196	1.2232	0.4984
ML	$31 \times 31 \times 31 \times 2$	480×320	34.2323	13.8090	16.7790	1.2151	0.4902
ML	$26 \times 26 \times 26 \times 2$	480×320	34.2470	13.7327	16.7058	1.2165	0.4878
ML	$22 \times 22 \times 22 \times 2$	480×320	34.8418	13.9543	16.9900	1.2175	0.4876
ML	$19 \times 19 \times 19 \times 2$	480×320	35.5754	14.2897	17.2601	1.2079	0.4852

IEEE Transactions on Visualization and Computer Graphics, vol. 14, no. 2, pp. 313–328, Mar. 2008.

[8] 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.

[9] C. Sigg and M. Hadwiger, "Fast third-order texture filtering," *GPU Gems 2*, pp. 313–329, 2005.

[10] 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.

[11] M. Kim, "Quartic box-spline reconstruction on the BCC lattice," *IEEE Transactions on Visualization and Computer Graphics*, Feb. 2013, (in print).

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

[13] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984.

[14] M. Kim and J. Peters, "Symmetric box-splines on root lattices," *Journal of Computational and Applied Mathematics*, vol. 235, no. 14, pp. 3972–3989, May 2011.

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

[16] S. R. Marschner and R. J. Lobb, "An evaluation of reconstruction filters for volume rendering," *Proceedings of the IEEE Conference on Visualization*, pp. 100–107, Oct. 1994.

[17] M. Kim, "GPU isosurface raycasting of FCC datasets," *Graphical Models*, 2012, (to appear).

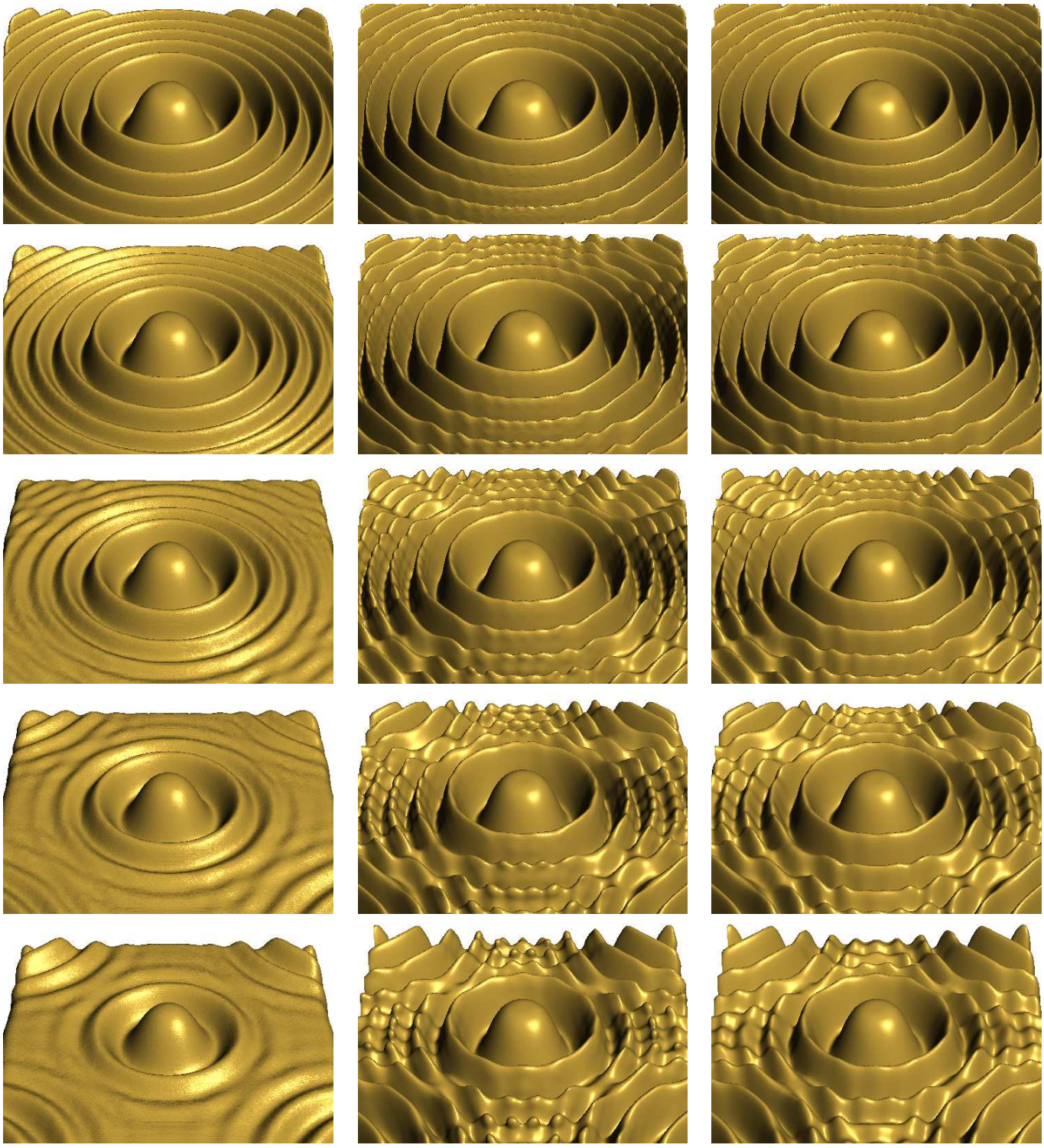


Figure 8: Rendered images of ML ML (Marschner-Lobb) datasets [16] of sizes (from top to bottom) $39 \times 39 \times 39 \times 2$, $31 \times 31 \times 31 \times 2$, $16 \times 16 \times 16 \times 2$, $22 \times 22 \times 22 \times 2$, and $19 \times 19 \times 19 \times 2$, respectively, reconstructed by (left) bcc12, (center) bcc8, and (right) bcc7, respectively. Notice that the reconstruction by bcc8 shows more 'ripples' around rims than bcc7.

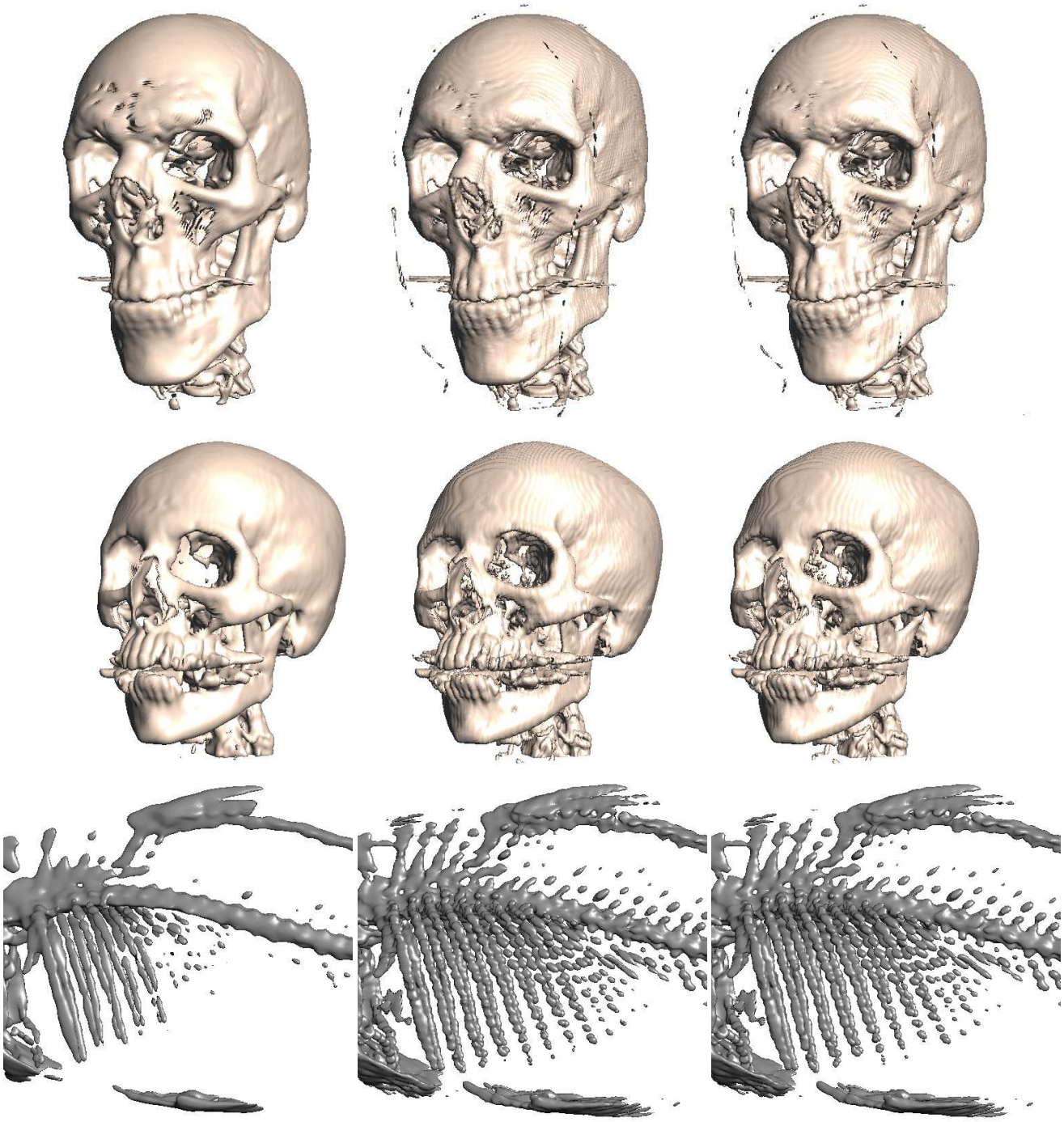


Figure 9: Rendered images of BCC datasets; (top) `VisMale` (middle) `CT-Head` and (bottom) `Carp` reconstructed by (left) bcc12 (center) bcc8 and (right) bcc7.