Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)
G.-P. Bonneau, S. Hahmann, C. D. Hansen (Editors)

Isosurfaces on Optimal Regular Samples

Hamish Carr,[1] Thomas Theußl,[2] and Torsten Möller[3]

[1] Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada
[2] Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria.
[3] Graphics, Usability, and Visualization (GrUVi) Lab, Simon Fraser University, Vancouver, BC, Canada.

Abstract

Volumetric samples on Cartesian lattices are less efficient than samples on body-centred cubic (BCC) lattices. We show how to construct isosurfaces on BCC lattices using several different algorithms. Since the mesh that arises from BCC lattices involves a large number of cells, we show two alternate methods of reducing the number of cells by clumping tetrahedra into either octahedra or hexahedra. We also propose a theoretical model for estimating triangle counts for various algorithms, and present experimental results to show that isosurfaces generated using one of our algorithms can be competitive with isosurfaces generated using Marching Cubes on similar Cartesian grids.

1. Introduction

One of the fundamental techniques for visualizing scalar volumetric data is the *isosurface*, the surface defined by a particular value in the data (the *isovalue*). Isosurfaces can be used to detect boundaries in the data set: recent work has dealt with selecting suitable isovalues[16, 17].

One of the drawbacks of isosurfaces, and of volumetric data in general, is sheer size, with data sets frequently composed of millions of data points. Theußl et al.[19] proposed using the body-centered cubic (BCC) lattice to reduce file sizes and processing time for volume rendering by nearly 30%. They did so by reducing the question of sampling to a well-known mathematical problem: packing spheres in three dimensions. Neophytou & Mueller [14] have extended this to four-dimensional data sets, where the reduction in file sizes reaches 50%. This is a strong reason for using BCC lattices instead of Cartesian lattices.

After acquisition and storage, the next task is to visualize the data: Theußl et al.[19] showed how to volume render BCC data using splatting, and achieved speedups of roughly 30%, largely due to the reduced number of samples. Neophytou & Mueller[14] extended this to four-dimensional BCC lattices, slicing the data to 3D, then splatting in 3D. Again, they achieved a 20-30% speedup. We review previous work in Section 2, and review the justification for BCC lattices in more detail in Section 3.

This paper does not seek to justify BCC lattices based on advantages in isosurface rendering. Instead, we assume that data is already available on a BCC lattice. It is then natural to ask whether it is possible to achieve similar speedups for isosurfacing to the speedups that can be achieved for volume rendering methods. One could simply resample the given BCC lattice into a more costly Cartesian lattice, but this would defeat the purpose. Instead, in this paper, we investigate different algorithms to extract isosurfaces from BCC grids directly, without a costly resampling step.

Constructing isosurfaces generally depends on subdividing the volume of interest into polyhedral cells, then dealing with each cell separately. In Section 4, we extend the sampling lattice to a polyhedral mesh that is suitable for isosurface generation. We start off by showing that the Delaunay complex of the samples is the natural set of cells for isosurfacing. In general, the Delaunay complex is built by connecting each sample to its nearest neighbours. It can also be found by taking the spatial dual of the Voronoï diagram, which defines regions according to the nearest sample: another characterization is that the Voronoï diagram is the spatial support for the nearest neighbour interpolant.

For cubical lattices, we show that the Delaunay complex of the samples is in fact the cubical cells used by Lorenson & Cline[10] for Marching Cubes. We then show that the corresponding natural cells for samples on a BCC lattice are the tetrahedra defined by the Delaunay complex of the BCC lattice.

Unfortunately, the Delaunay complex turns out to be less efficient than Marching Cubes on a resampled Cartesian lattice, due to a substantially increased number of polyhedral cells in the mesh. We deal with this by substituting octahedra and hexahedra for tetrahedra in the BCC mesh. This requires that we define the cases for Marching Octahedra in Section 5, and that we show how to simplify tetrahedra to octahedra in Section 6. In Section 7, we then repeat this for the hexahedral decomposition of the tetrahedral mesh.

In Section 8, we present a theoretical model for estimating the size of an isosurface, based on Itoh & Koyamada's estimate[8] that $O(N^{2/3})$ of the cells intersect any given isosurface, and that the number of triangles in any given cell can be approximated by the average of the number of triangles in the various Marching Cubes, Tetrahedra, Octahedra or Hexahedra cases.

Finally, we test our hypotheses in Section 9, state our conclusions in Section 10, and discuss future work in Section 11.

2. Previous Work

Early work on visualizing volumetric data used *boundary detection*. Herman & Liu[6] represented a volume by subdividing it into small cubes called *cuberilles*. Each cuberille was centred on a single sample: samples were assumed to have been acquired on a cubic lattice. In order to render significant boundaries, a threshold was chosen. Cuberilles centred on samples with values above the threshold were then rendered: values below the threshold were omitted.

This approach implicitly chooses the nearest neighbour interpolant to evaluate the function between samples. The boundary generated coincides with the isosurface under the nearest neighbour interpolant. This also ties the cuberille approach to the idea of Voronoï regions. A Voronoï region is the set of points closest to a given sample: it follows that each cuberille is the Voronoï region of the sample at its centre.

The principal drawback of the cuberille approach is that the surfaces generated were not smooth. The next development, *Marching Cubes*[10, 24], addressed this using the Delaunay mesh: the spatial dual of the Voronoï regions.

Instead of using a mesh of regions centred on each sample, Marching Cubes divides the space into cells whose vertices are the samples (Figure 1(b)). Two vertices are connected if they are adjacent in either x, y, or z. These directions correspond to the faces of the cuberilles: it is easy to see that this generates the Delaunay mesh, in which two vertices are connected if their Voronoï regions share a face. Once the mesh has been constructed, each cube is processed separately to obtain part of the isosurface[10, 24]. This part of the surface is

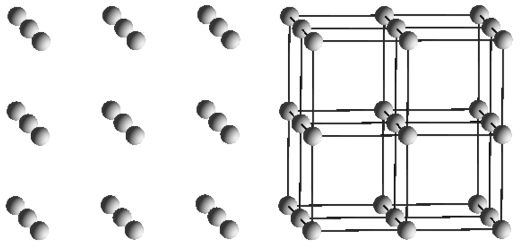

(a) Cubic Samples (b) Cubic Mesh

Figure 1: *Building a Cubic Mesh*

chosen to approximate a trilinear interpolation function, with varying levels of accuracy[10, 12, 13, 15, 22].

Either the cuberilles (i.e. Voronoï regions) or Marching Cubes (i.e. Delaunay meshes) can be extended to non-Cartesian lattices. Ibañez et al.[7] implemented the Voronoï approach for both BCC and FCC grids. In both cases, however, the Voronoï regions of the samples are more complex than the cube. The Voronoï region of a sample on the BCC lattice is a truncated octahedron requiring 44 triangles to render: the Voronoï region on the FCC lattice is a rhombic dodecahedron. As with cuberilles, using the Voronoï cells assumes the nearest neighbour interpolation, and fails to generate smooth surfaces. Ibañez et al., moreover, do not give any details of isosurface complexity, or compare the artifacts to other methods.

The Delaunay mesh of the BCC lattice was used by Chan and Purisma in a tessellation scheme using Marching Tetrahedra[2], and also by Treece et al.[20]. Again, no results in terms of triangle complexity were presented.

We now turn to the layout of samples in the volume of interest, to justify why we should generate isosurfaces from BCC lattices.

3. Non-Cubic Sampling

In order to discuss different ways of laying out samples in a volume, we need a framework to locate individual samples. Following Dudgeon and Mersereau[4], we describe sampling as a mapping of indices to actual sample positions, using matrices to describe the mapping. For any given layout, the matrix V, the *sampling matrix*, converts indices to sample positions as follows:

$$\begin{pmatrix} x \\ y \end{pmatrix} = V \cdot \begin{pmatrix} i \\ j \end{pmatrix} \quad (1)$$

Note that the columns of V are basis vectors for the space:

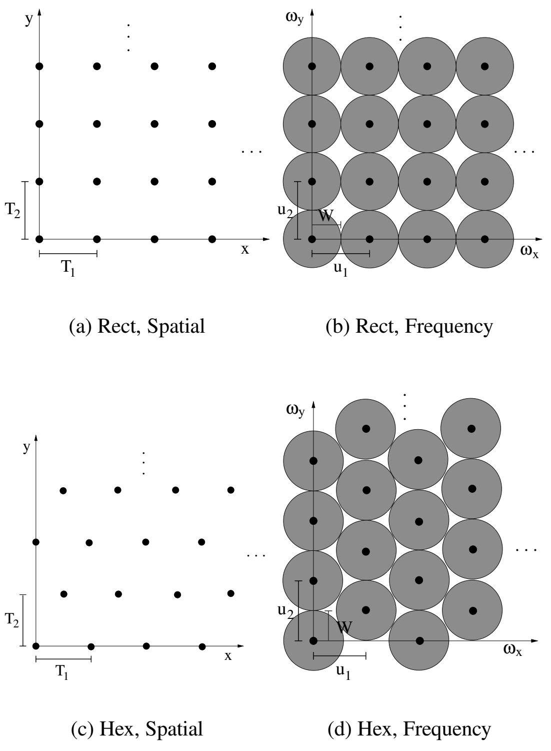


Figure 2: *Lattices in the Spatial and Frequency Domains*

each choice of basis vectors will give a different set of sampling locations. For example, the matrix

$$V_{rect2D} = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} \tag{2}$$

describes the commonly used rectangular sampling, shown in Figure 2 (a) and (b).

The rectangular sampling, however, is not the optimal sampling scheme in 2D. The optimal sampling scheme, also shown in Figure 2 is the *hexagonal close packing*, or HCP, lattice. For isotropic band-limited functions, the primary spectrum of the samples is a sphere (circle, in 2D) in the frequency domain. For lattice-based sampling schemes, the primary spectrum is replicated in the frequency on another lattice, with the property that

$$UV = I \tag{3}$$

where U is the sampling matrix in the frequency domain, and V is the sampling matrix in the spatial domain.

In order to sample data optimally, we wish to pack the replicas of the primary spectrum as tightly as possible in the frequency domain. This reduces to the well-known problem of packing spheres in 2, 3, or higher dimensions. For 2D, the optimal packing is known to be the hexagonal packing[11]: that this is better than the rectangular packing is clear from Figure 2 (b) and (d).

For the hexagonal packing in the frequency domain, the corresponding sampling in the spatial domain is also hexagonal packing, and can be described by the matrix

$$V_{hex2D} = \begin{pmatrix} T_1 & \frac{1}{2}T_1 \\ 0 & T_2 \end{pmatrix} \qquad \text{for } T_2 = \frac{\sqrt{3}}{2}T_1 \tag{4}$$

To keep the samples in a rectangular area, we note that the samples of every second row take the same x-positions, shifted by one unit. Consequently, we only shift rows with odd index by half a unit:

$$V_{hex2D} = \begin{pmatrix} T_1 & \frac{1}{2j}T_1(j \bmod 2) \\ 0 & T_2 \end{pmatrix} \quad \text{for } T_2 = \frac{\sqrt{3}}{2}T_1,\ j > 0 \tag{5}$$

Using this matrix we get indices from zero to the maximum in each dimension and still describe a rectangular area.

For 3D, there are an infinite number of optimal regular packings in the frequency domain, although there are two principal such packings: the face-centred cubic (FCC) and the hexagonal close-packing (HCP). In particular, the FCC packing in the frequency domain corresponds to a sampling lattice called the body-centred cubic (BCC) lattice in the spatial domain. Theußl et al.[19] showed that the BCC lattice can be described by the sampling matrix

$$V_{BCC} = \begin{pmatrix} T & 0 & \frac{1}{2}T \\ 0 & T & \frac{1}{2}T \\ 0 & 0 & \frac{1}{2}T \end{pmatrix} \tag{6}$$

which is modified to

$$V_{BCC} = \begin{pmatrix} T & 0 & \frac{1}{2k}T(k \bmod 2) \\ 0 & T & \frac{1}{2k}T(k \bmod 2) \\ 0 & 0 & \frac{1}{2}T \end{pmatrix} \quad \text{for } k > 0 \tag{7}$$

for an intuitive memory layout scheme (note that the corresponding formulae 20 and 21 in [19] are incorrect). The data set is stored in a 3D array where every second plane is implicitly translated half the sampling distance in the x and y directions, and the distance between planes in the z direction is half that in the other two directions.

The hexagonal close-packing is more difficult to convert into the spatial domain than the BCC, principally because, in 3D, it cannot be described simply by a sampling matrix. Accordingly, we focus our attention principally on the BCC lattice for generating isosurfaces.

4. Mesh Construction

Once we have defined a sampling lattice, we need to convert the lattice to a mesh before performing isosurface construction. As described in Section 2, we use the Delaunay complex of the sampling lattice. For body-centred cubic lattices, the Delaunay complex is shown in Figure 3(b), and is composed entirely of tetrahedra. This BCC mesh involves two

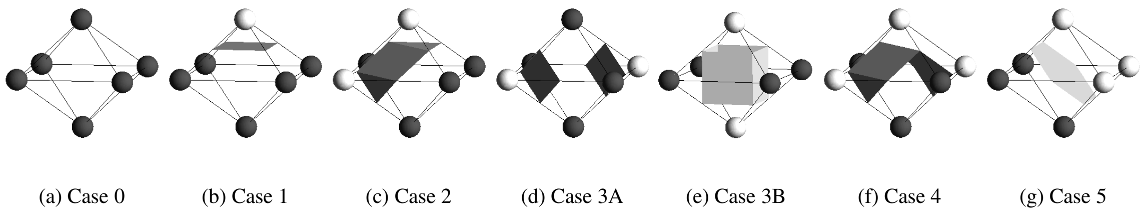


Figure 4: *Marching Octahedra Cases*

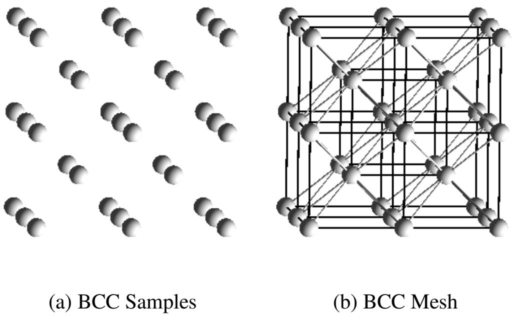


Figure 3: *Generating Meshes from Regular Samples*

superimposed cubic meshes, shown in dark grey, with diagonal connections between the two meshes, shown in light grey. For convenience, we refer to the two superimposed cubic meshes as the *primary* and *secondary* lattices. Note that we have to deal with boundary cases at the edge of the primary lattice, where the secondary lattice would stretch past the boundary. These cases can be dealt with either by using pyramidal cells, as shown in Figure 3(b), or by adding an extra layer of secondary lattice vertices off the boundary, with an assigned isovalue of 0. We have chosen the latter approach.

Since all of the cells in this mesh are tetrahedra, we need only define the cases for tetrahedral cells in order to extract isosurfaces. For tetrahedral cells, we apply *Marching Tetrahedra*[2]. However, we will also need octahedral cells for one of the alternate algorithms: we describe the octahedral cases in the next section. In Section 6, we will then describe how to adjust the octahedral cases to match the topology of the tetrahedral cells. And in Section 7, we will define a set of hexahedral cases that also match the tetrahedral topology.

5. Marching Octahedra

Following Marching Tetrahedra and Marching Cubes, we classify each vertex of an octahedron as above or below the isovalue. After eliminating symmetries and complements,

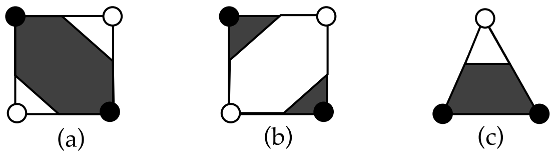


Figure 5: *Ambiguous Faces*

we are left with six basic cases, shown in Figure 4. Note that case 3 can be tessellated in two different ways, shown as 3A and 3B. Unlike Marching Cubes, this ambiguity does not result in cracks in the isosurface, such as those that Dürst[5] noted. These cracks only occurred when two diagonal corners of a face of the cube were above the isosurface, and the other two below. As shown in Figure 5 (a) and (b), there are two possible cases: in each case, the light areas are "above" the isosurface, and the dark areas "below". Cracks occur when the two cubes sharing a face choose opposite cases. One solution to this was the asymptotic decider introduced by Nielson & Hamann[15].

For the triangular face of an octagon, however, there is only one possible case, shown in Figure 5(c), and no ambiguity is possible on the face. The ambiguity between 3A & 3B is instead similar to the cases that Natarajan introduced[13], in which topological changes in the surface can occur in the body of the cube, depending on the interpolant function used. These body saddles do not, however, cause cracks in the surface. As a result, we can choose either 3A or 3B without fear of cracks in the surface.

We propose four possible ways of deciding between case 3A and 3B. The simplest solution is to use case 3A all the time, as it generates fewer triangles than case 3B. The second solution is to base our choice on the interpolant function for the octahedron. As with Nielson & Hamann's *asymptotic decider*[15], we could test for saddle points in the function to distinguish between the two cases. Unfortunately, unlike the cube and the tetrahedron, no interpolant function is immediately apparent for the octahedron.

The third solution is to assume the function value at the centre of the octahedron should be the average of the values

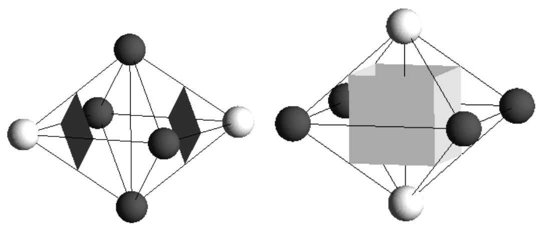

(a) Case 3A (b) Case 3B

Figure 6: *Modified Marching Octahedra Cases*

at the vertices. If the isovalue is less than this average value, we would choose 3B, else 3A. All other cases are unaffected by this assumption, at least to the extent that no topological change to the surfaces shown in Figure 4 is required. Using this average value is topologically equivalent to subdividing the octahedron into 8 simplices by adding an interpolated vertex at the centre.

The fourth solution is to choose the cases so that we replicate the topology generated by isosurfacing with four tetrahedra instead of the octahedron. This forms the basis for a simplification of the BCC lattice, which we discuss in the next section.

6. Modified Marching Octahedra

In the body-centred cubic lattice, we observe that four tetrahedra share each edge in the primary and secondary lattices. If we select either the primary or secondary lattice, we can group tetrahedra to form octahedra, based on this shared edge, as shown in Colour Figure 9(a). Note that, unlike Figure 4, we have shown. For convenience, we refer to the shared edge as the *spine* of the octahedron: the spine of the octahedron is shown in red in Colour Figure 9(a); the edges of the octahedron are shown in green.

For all cases except case 3, if we extract isosurfaces separately in the four tetrahedra along the spine, we obtain the same surface topologically. To see this, consider case 1 in Figure 4. Suppose that the light grey vertex is "above" the surface, and along the spine of the octahedron. It is not difficult to see that the spine will pass through the centre of the quadrilateral surface, dividing it into four triangles. If we extract the surface separately in the four tetrahedra along the spine, each tetrahedron will contribute one of the four triangles. In comparison, we can render the surface in the octahedron with two triangles. The analysis for other cases is similar, except for case 3.

In case 3, if the two vertices "above" the isosurface are along the spine of the octahedron, Marching Tetrahedra would render the surfaces shown in Figure 6(b). Similarly, if the two vertices "above" the isosurface are *not* along the spine of the octahedron, Marching Tetrahedra would render the surfaces shown in Figure 6(a). This leads to a simple test for whether we use case 3A or 3B: if the two "above" vertices are along the spine, use case 3B. Otherwise, use case 3A. Again, we guarantee that the surfaces generated are topologically equivalent to the surface extracted using tetrahedra.

In most of our cases, the number of triangles generated by Marching Octahedra is fewer than that generated by Marching Tetrahedra. Thus, by substituting octahedra for tetrahedra, we can reduce the number of triangles rendered for an isosurface on a body-centred cubic lattice.

Although this modified version of Marching Octahedra does reduce the number of triangles generated, we will see in Section 8 and Section 9 that the reduction is not sufficient to compete with Marching Cubes on a cubic mesh. Although the octahedron has 6 vertices instead of the 8 vertices of the cube, each vertex is degree 4 rather than degree 3. Thus, a surface that separates one vertex from the rest takes a minimum of 2 triangles, compared with 1 for the cube. As a result, the average number of triangles per cell is significantly greater than for Marching Cubes.

To further reduce the number of triangles, we note that Modified Marching Octahedra only uses the primary and secondary lattice edges as spines. If we instead choose diagonal edges as spines, we can reduce the BCC mesh to a set of hexahedra, as discussed in the next section.

7. Modified Marching Hexahedra

In the BCC mesh, there are four different diagonal directions, shown as dark grey edges in Figure 3(b). Any three of these directions form a basis for the lattice. The hexahedron defined by the basis edges consists of six tetrahedra sharing the remaining direction as a spine, as shown in Colour Figure 9(b). As with octahedral reduction of the BCC mesh, we show the spine in red, and the edges of the hexahedron in green. Six edges from the primary and secondary lattice also appear as diagonals of the faces of the hexahedra: these are shown in blue.

In the previous section, we modified the Marching Octahedra cases to obtain a guarantee of topological consistency with the tetrahedrally-extracted isosurface. We now construct a set of cases for the hexahedra that give a similar guarantee. We show these cases in Colour Figure 9(b): the cases are drawn in cubes in order to maintain consistency with the Marching Cubes cases, and to save space on the page. Unsurprisingly, we have the same basic cases (0 - 13) as Marching Cubes.

In Marching Cubes, as noted in Section 5, the cases are carefully chosen so that face diagonals are consistently

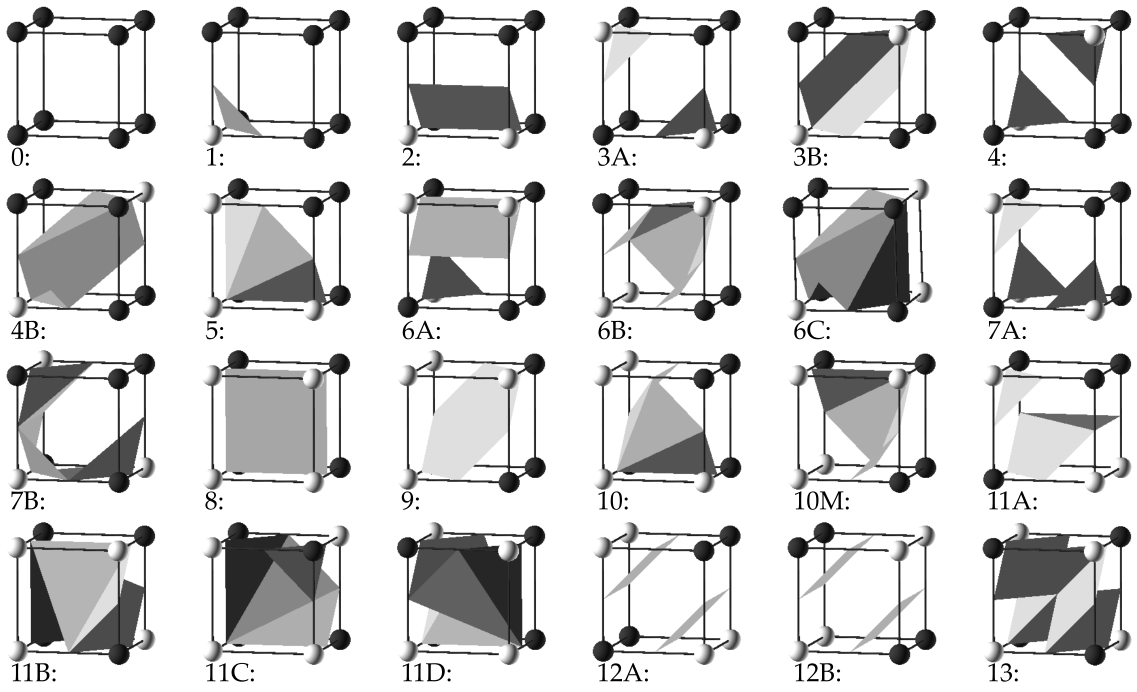


Figure 7: *Cases for Modified Marching Hexahedra*

treated. The rule applied is that two vertices above the isosurface, and diagonal to each other, are never connected, but two vertices below the isosurface, and diagonal, are always connected.

For our hexahedral simplification, this rule does not capture the correct topology. In Figure 3(b), we see that each face of the hexahedron is divided by a blue tetrahedron edge. If both vertices on this edge are above the isosurface, then the entire edge must also be above the isosurface. Similarly, if both are below, the entire edge must also be below the isosurface. And the same rule applies to the spine of the hexahedron. Thus, instead of the rule above, we have a new rule: if two vertices above (below) are connected by an edge, they may not be separated by an isosurface. If they are not connected by an edge, they must always be separated. Once we have defined these constraints, it is easy to construct the full set of cases shown in Figure 7. In contrast to the original Marching Cubes, where rotations around the x-, y-, or z-axes define the symmetric case reductions, our symmetric case reductions are based on rotations around the spine, or flipping the spine end-over-end. Moreover, the complementary cases obey the same rule, so we need only show cases 0 - 13, with their subcases.

In case 13, we found it necessary to add a vertex along the spine of the hexahedron, to avoid separating vertices connected by an edge. In all other cases, we avoided adding a vertex along the spine, to keep the triangle count down.

We have now defined the necessary cases for extracting isosurfaces on a BCC lattice, and wish to predict how they will perform. For this, we define a theoretical model of isosurface size in the next section.

8. Expected Triangle Counts

Visualization using isosurfaces has two principal steps: isosurface extraction, and (finally) isosurface rendering. The basic technique, Marching Cubes, costs $O(N)$ to extract each isosurface, where N is the number of polyhedral cells in the mesh. A related parameter, n, counts the number of samples. For regular grids such as those discussed here, $n = \Theta(N)$. For extracting a single isosurface, the extraction cost is at least $\Omega(N)$[10]. Much effort has since gone into reducing this cost for extracting multiple isosurfaces, using techniques such as octrees[23], span space[9], interval trees[3], extrema graphs[8], and contour trees[21, 1].

Broadly speaking, these techniques trade off preprocessing time (of as little as $N\log(N)$) to reduce the cost of extracting individual isosurfaces to as little as $\log N + k$, where k is an output-sensitive parameter. Since k is generally expected to be $O(N^{2/3})$, it is clear that k is the dominant term in defining an individual isosurface. The cost of rendering the isosurface is generally borne by specialized hardware, which takes a surface defined by triangles and renders it to the screen. Since there is a fixed bound on the number of triangles generated in a single cell, it is clear that the rendering time after extraction is $\Omega(k)$: $\Theta(k)$ if Z-buffering is used, but as much as $\Theta(k \log k)$ if a sorting-based algorithm is used. Moreover, modern graphics hardware generally caches the triangles in video RAM: since a triangle occupies as much as 100 bytes in OpenGL, the memory bandwidth required between the CPU and the video card is also $\Omega(k)$.

Thus, we claim that k, the number of triangles constructing the isosurface, is the parameter that needs to be addressed when comparing isosurfacing schemes. Even if isosurface simplification is to be performed, k will still describe the input to the simplification routine. We do not address the effect of point-based isosurfaces, beyond noting that the isosurface is defined by the cells that it passes through: point-based isosurfaces will generally wish to render at least one point per cell: thus, k will be a lower bound for such techniques.

In order to compare schemes on an abstract level, we first look at an abstract model based on a random distribution of cases, then consider some experimental results in Section 9. We have two reasons for considering it at this abstract level: first, to predict which approaches are merited, and second, because BCC data is, at present, difficult to obtain for non-analytical functions.

Here we would like to point out that the comparison of techniques on Cartesian grids (Marching Cubes) and on BCC grids (Marching Tetrahedra or Marching Octahedra or Marching Hexahedra) assumes an equivalent representation of the data set on each lattice. In practice this is not always possible and we have to take sampling artifacts into account for resampling a Cartesian lattice from a given BCC lattice (or vice versa).

Our model assumes that the number of triangles in an isosurface can be predicted by the formula:

$$N_{triangles} = N_{cells} \times N_{triangles/cell} \qquad (8)$$

We follow Itoh & Koyamada[8] in estimating that $N_{cells} = O(N^{2/3})$. This leaves us with the problem of estimating $N_{tri/cell}$. As an initial approximation, we assume that all cases of Marching Cubes, Marching Tetrahedra or Marching Octahedra, with two exceptions, are equally likely. In practice, we expect that topologically complex cases will be less common than topologically simple cases. Since topologically complex cases require more triangles than simple cases, this means that our initial estimate is likely to be a conservative over-estimate. In practice, our results bear this out, as is shown in Section 9. These results also show that the average number of triangles per cell is consistent across different types of data.

The exceptions to our assumption are case 0 and its complement in each scheme. Case 0 occurs when all of the vertices of the cell are above the isosurface: its complement applies when all of the vertices are below the isosurface. Since most isosurfaces pass through a relatively small fraction of cells, case 0 is expected to predominate. However, since no triangles are generated in either case, this makes it fairly easy to compensate, by leaving these cases out of the computation.

For Marching Tetrahedra, with four vertices, there are 16 distinct possibilities, including 1 instance of case 0, 1 instance of its complement, 4 cases each of case 1 and its complement, and 6 instances of case 2. Since case 1 uses 1 triangle, and case 2 uses 2, the average number of triangles generated per cell is:

$$\frac{4}{14} \times 1 + \frac{6}{14} \times 2 + \frac{4}{14} \times 1 = \frac{20}{14} = 1.43 \qquad (9)$$

For the basic Marching Octahedra, where case 3A is always chosen, the calculation is similar. Where the complement of a case exists, we simply double the calculation for the basic case, rather than listing it separately:

$$\frac{12}{62} \times 2 + \frac{24}{62} \times 4 + \frac{6}{62} \times 4 + \frac{12}{62} \times 6 + \frac{8}{62} \times 4 = \frac{248}{62} = 4.00 \qquad (10)$$

For the Modified Marching Octahedra, there are two instances of case 3B, in each of which 4 additional triangles are rendered. Thus, the expected number of triangles increases to 4.07. Since Marching Cubes has 256 cases, falling into 13 types, we omit the detailed calculation. The result is 2.85 triangles per cell. For the Modified Marching Hexahedra of Section 7, the corresponding figure is 3.51 triangles per cell, principally because respecting the diagonals in the cell adds complexity, and therefore triangles, to the cases.

To compare the expected results for different sampling methods, we assume that N samples are required for a cubic mesh. Looking at Figure 1(b), we note that each cubic cell has 8 vertices, so we charge $1/8$ cell to each vertex (i.e. sample). Since each vertex is involved in 8 cells, the average number of cells per sample is 1, disregarding cases at the boundary of the mesh. Thus, the cubic mesh is composed of roughly N cells. Of these, we follow Itoh & Koyamada[8] in assuming that roughly $N^{2/3}$ cells intersect the isosurface.

Thus, the expected number of triangles for an isosurface using Marching Cubes is:

$$2.85N^{2/3}. \qquad (11)$$

In comparison, $0.707N$ samples are required for either a body-centred cubic lattice or a hexagonal close packing. For the body-centred cubic lattice, $1/4$ of each tetrahedron is charged to each of its vertices (samples). Each sample is involved in 24 tetrahedra, so we expect 6 tetrahedra on average per sample, for a total of $6 \times 0.707N = 4.24N$ cells, of which $(4.24N)^{2/3} = 2.62N^{2/3}$ are expected to intersect the surface.

Multiplying by the average number of triangles per cell, we get an expected number of triangles of:

$$1.43 \times (6 \times 0.707N)^{2/3} = 3.75N^{2/3} \qquad (12)$$

Compared to the figure for Marching Cubes, we predict that Marching Tetrahedra on the body-centred cubic lattice will generate approximately $3.75/2.85 = 1.32$ times as many triangles as Marching Cubes.

For Modified Marching Octahedra, each sample on the primary lattice is involved in 12 octahedra: each sample on the secondary lattice is involved in 6 octahedra. We charge $1/6$ to each sample, giving an average of 1.5 octahedra per sample. Note that we use the estimate of 4.07 triangles per cell, as we are using case 3B instead of case 3A for two of the 64 cases. The expected number of triangles is then:

$$4.07 \times (1.5 \times 0.707N)^{2/3} = 4.23N^{2/3} \qquad (13)$$

and we expect that Modified Marching Octahedra will generate $4.23/2.85 = 1.48$ times as many triangles as Marching Cubes. This reduces to 1.46 if we always use case 3A.

If we use the Marching Hexahedra to simplify the BCC mesh, we note that there is exactly one spine edge for each sample, and therefore one hexahedron per sample. But we have 0.707 times as many hexahedra as we would cubes. Thus, the number of expected triangles is:

$$3.51 \times (0.707N)^{2/3} = 2.79N^{2/3} \qquad (14)$$

and we expect 0.98 times as many as for Marching Cubes.

Although these results should be taken cautiously, what they indicate is that isosurfaces generated from optimal regular samples should be composed of roughly similar numbers of triangles as those generated from the same data set using a cubic mesh and Marching Cubes. In the next section, we present some experimental results to test this hypothesis.

9. Experimental Triangle Counts

In this section we present the results of experimentally measuring k, the number of triangles in an isosurface, for different data sets sampled on both BCC as well as Cartesian grids. We must first note, however, that data is not presently acquired using BCC lattices. Thus, except for synthetic functions, we have been forced to resample from Cartesian-grid data to BCC data. We do not claim that this is ideal. Until someone constructs a BCC-based scanner, however, we have little choice. Notwithstanding this, we claim that it is worthwhile testing the abstract model as rigorously as possible.

In order to test our hypothesis, we took a selection of volumetric data sets from volvis.org, each of which was sampled on both BCC and Cartesian lattices: these sets are summarized in Table 1. Of these, one data set was synthetic (M-Lobb), and was sampled independently on both lattices. The remaining data sets were only available on the cubic lattice, so we resampled to the body-centred cubic lattice using a width three Kaiser windowed sinc with alpha=8.93[18]. We note that it is entirely possible that this affected our results.

Data	Cartesian Dimension	BCC Dimension
M-Lobb	40x40x40	28x28x56
Fuel	64x64x64	45x45x90
Hipiph	64x64x64	45x45x90
Neghip	64x64x64	45x45x90
Lobster	120x120x34	84x84x48
Skull	256x256x256	181x181x362
Tooth	256x256x161	181x181x227
Vessels	256x256x256	181x181x362

Table 1: *Datasets*

All data was stored as 8 bit data. We note that the isosurface at a height of 0.5 must pass through the same cells as any other isosurface in the interval $[0,1)$, using exactly the same cases. The interval is half-open because the Marching tables assume that any isovalue is either above or below the value at a vertex: isosurfaces at the vertex' value are generated as if they were a small amount higher.

Thus, in order to test our hypothesis, we need only generate 255 distinct isosurfaces: for convenience, we took the isosurfaces at $0.5, 1.5, 2.5, \ldots, 254.5$ for each data set, and each possible isosurfacing scheme: sample isosurfaces of the lobster data set are shown in Colour Figure 8. Note that, in order to highlight differences, we used normals based on the triangles generated, rather than the gradient of the field at the surface. We expect that the visual artifacts using gradient-based normals would be significantly reduced.

Sadly, the isosurfaces extracted on the BCC lattice in Colour Figure 8 (b) - (d) are significantly less smooth than the isosurface extracted using Marching Cubes in Colour Figure 8(a). We attribute this to two factors. First, the lobster, and most of the other data sets were resampled from cubic data. We expect, therefore, that the Marching Cubes will be a more faithful replication of the function. However, we observed similar artifacts even for the one analytical data set. We think that the reason for this is rather subtle. We initially assumed that our function was band-limited and anisotropic. Under these assumptions, the region of influence of each sample ought to be inherently spherical in the spatial domain. This is perhaps why the results of Theußl et al. for splatting were so good: the splatting kernels were spherical distributions. Here, the tetrahedra we use are not good approximations of a sphere: they are longer on two edges than the other 4. Even if they were regular, these tetrahedra are still worse approximations of a sphere than a cube is. It is difficult to prove this assertion, but we suspect that some such effect plays a part.

For each isosurface, we computed the number of cells intersected by the isosurface, the number of triangles generated for the isosurface, and the average number of triangles per cell. We summarize this information in Table 2 and Table 3. Where no isosurface existed (e.g. no isosurface exists at a value of 200.5 if the highest valued sample in the data set is 100), we excluded that isosurface from the computation.

From Table 2, we observed that the average number of triangles per cell was remarkably consistent in each data set, and even between data sets. We were expecting a value of 2.85 for Marching Cubes. While the ratio was lower than expected (around 2.00), they were quite consistent. For Marching Tetrahedra (MT), we expected 1.43, and got approximately 1.25. For Modified Marching Octahedra (mmo), we expected 3.75, and got roughly 3.1. For Modified Marching Hexahedra (MMH), we expected 3.51, and got apprximately 2.20. Only the hipiph data set was unusual. On inspection, we discovered that this data set is quite unevenly distributed: high isovalues have small cell counts, which seem to consist entirely of simple cases, with a ratio close to 1.0 for nearly $1/3$ of the possible isovalues. Since we averaged over all isovalues, these cases significantly distorted the average. In all cases, the average number of triangles per cell was lower than expected, but remarkably uniform (standard deviations were around 0.05 for each data set). We conclude that it is reasonable to predict triangle counts based on the type of cell, although the distribution is less uniform than we had predicted.

Lattice	Cubic	BCC	BCC	BCC
Method	Marching Cubes (MC)	Marching Tets (MT)	Modified Octs (MMO)	Marching Hexes (MMH)
M-Lobb	2.07	1.32	3.48	2.68
Fuel	1.98	1.25	3.08	1.58
Hipiph	1.75	1.15	2.60	1.58
Neghip	1.99	1.29	3.20	2.25
Lobster	2.02	1.27	3.10	2.17
Skull	1.99	1.28	3.20	2.24
Tooth	1.98	1.26	3.13	2.07
Vessels	1.92	1.27	3.09	2.14

Table 2: *Triangles per Cell*

Turning our attention to the actual triangle counts in Table 3, we recall that we predicted ratios of 1.32 (BCC/MT), 1.48 (MMO), and 0.98 (MMH), as compared to Marching Cubes (MC). We show the actual ratios in Table 4. As predicted, Marching Hexahedra is the best BCC-based scheme: here, however, the prediction fails, as Marching Cubes was marginally superior to Marching Hexahedra, with Marching Octahedra and Marching Tetrahedra doing significantly worse.

We also noted that these competitive ratios varied more

Lattice	Cubic	BCC	BCC	BCC
Method	Cubes (MC)	Tets (MT)	Octs (MMO)	Hexes (MMH)
M-Lobb	11,398	18,821	15,208	8,574
Fuel	3,341	11,267	8,847	4,806
Hipiph	3,446	11,532	8,984	4,838
Neghip	20,755	85,428	66,037	36,531
Lobster	69,835	174,716	134,851	72,513
Skull	1,246,032	3,410,422	2,656,530	1,491,528
Tooth	335,500	904,491	708,326	395,789
Vessels	186,264	586,556	454,815	255,157

Table 3: *Triangle Counts*

Lattice	Cubic	BCC	BCC	BCC
Method	Cubes (MC)	Tets (MT)	Octs (MMO)	Hexes (MMH)
Predicted	1	1.32	1.46	0.98
M-Lobb	1	1.66 ± 0.15	1.34 ± 0.10	0.75 ± 0.07
Fuel	1	3.15 ± 0.57	2.48 ± 0.46	1.34 ± 0.27
Hipiph	1	3.42 ± 1.43	2.68 ± 1.13	1.32 ± 0.47
Neghip	1	4.00 ± 0.49	3.10 ± 0.38	1.74 ± 0.22
Lobster	1	2.33 ± 0.41	1.82 ± 0.32	0.98 ± 0.18
Skull	1	3.23 ± 0.53	2.53 ± 0.42	1.43 ± 0.26
Tooth	1	3.59 ± 0.98	2.83 ± 0.79	1.56 ± 0.45
Vessels	1	2.96 ± 0.63	2.30 ± 0.49	1.28 ± 0.29

Table 4: *Competitive Ratios: Means / Standard Deviations*

between data sets than did the average number of triangles per cell. The ratio also varied significantly between different isosurfaces in the same data set.

From these observations, we conclude that, although it is reasonable to predict the average number of triangles per cell, the model used for predicting average number of cells intersected is less reasonable. We can see two possible reasons for this. The estimate of $O(N^2/3)$ due to Itoh & Koyamada[8] could be incorrect, or the constants of proportionality could be dependent on mesh type and data type. We also conclude, however, that Marching Hexahedra are sufficiently competitive with Marching Cubes to warrant further investigation.

10. Conclusions

We have introduced and analyzed new algorithms that enable us to generate isosurfaces directly from a given BCC lattice without resampling the BCC lattice into a Cartesian lattice.

Secondly, we have introduced a theoretical model for estimating isosurface siz: we used this model to guide our decisions when establishing the Modified Marching Octahedra and Modified Marching Hexahedra algorithms. From our experimental results in Table 2, we see that, across multiple isosurfaces and different data types, we can estimate the average number of triangles per cell. However, our results from Table 4 imply that, even if the $O(N^{2/3})$ estimate of Itoh & Koyamada[8] is correct, different constants apply for different mesh and data types.

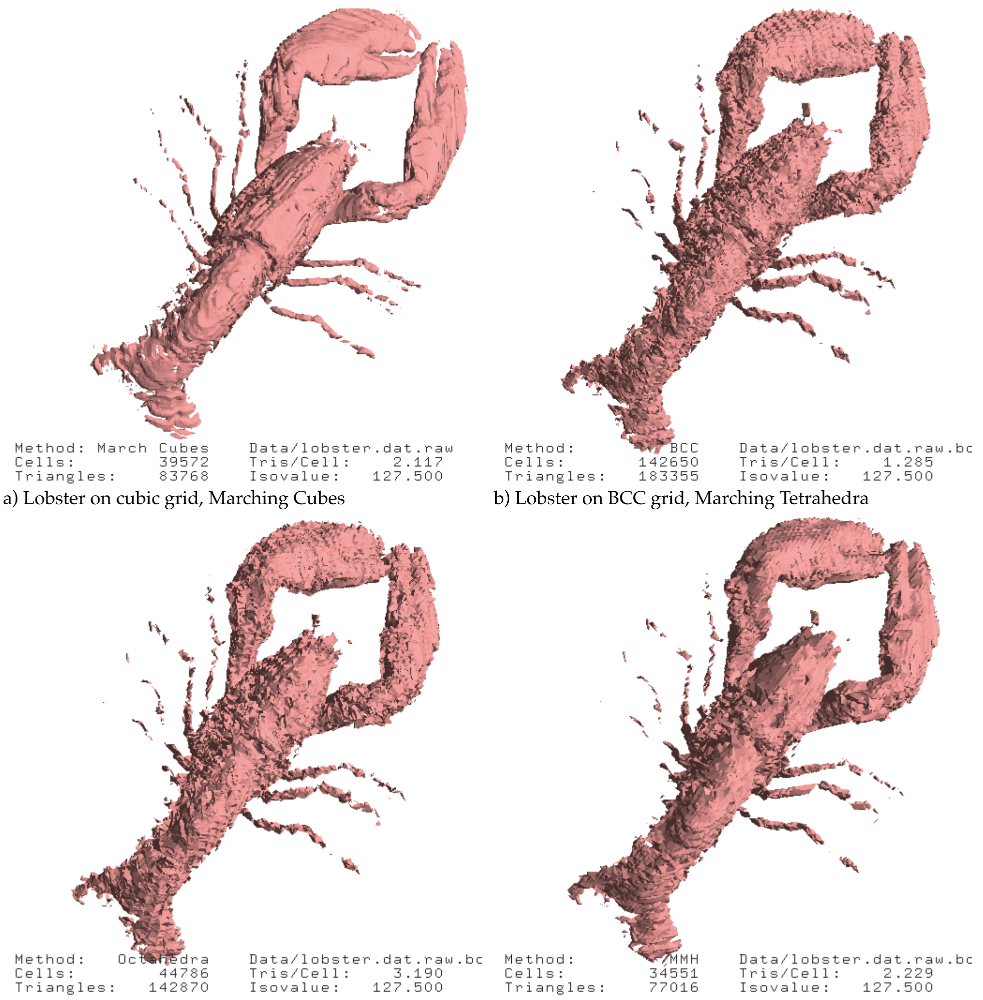
Finally, we note that the isosurfaces generated from BCC grids seem to be of poor quality compared with those generated from Cartesian grids. However, we have shown that isosurfaces on BCC grids can be computed in such a way that they are within a reasonable factor of the cost of Marching Cubes isosurface construction, and, in one case, we achieved a reduction in isosurface size similar to the results of Theußl et al. for volume rendering and file size.

11. Future Work

There are a number of issues left unresolved. For completeness, triangle counts should be computed for the hexagonal close-packing. We have seen that the triangle counts do not quite match the estimates in Section 8. These estimates could be refined by computing average numbers of triangles per cell for differing types of data, and by conducting experiments to test Itoh & Koyamada's hypothesis that $O(N^{2/3})$ cells intersect any given isosurface.

References

1. H. Carr, J. Snoeyink, and U. Axen. Computing Contour Trees in All Dimensions. *Computational Geometry: Theory and Applications*, 24(2):75–94, 2003.
2. S. L. Chan and E. O. Purisima. A new tetrahedral tesselation scheme for isosurface generation. *Computers & Graphics*, 22(1):83–90, Feb. 1998. ISSN 0097-8493.
3. P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding Up Isosurface Extraction Using Interval Trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–169, 1997.
4. D. E. Dudgeon and R. M. Mersereau. *Multidimensional Digital Signal Processing*. Prentice-Hall, Inc., Englewood-Cliffs, NJ, 1st edition, 1984.
5. M. Dürst. Letters: Additional Reference to "Marching Cubes". *Computer Graphics*, 22(4):65–74, 1988.
6. G. T. Herman and H. K. Lun. Three-Dimensional Display of Human Organs from Computed Tomograms. *Computer Graphics and Image Processing*, 9:1–21, 1979.
7. L. Ibáñez, C.Hamitouche, and C.Roux. Determination of discrete sampling grids with optimal topological and spectral properties. In *Procceedings of the 6th International Workshop in Discrete Geometry for Computer Imagery DGCI'96*, pages 181–192, 1996.
8. T. Itoh and K. Koyamada. Automatic Isosurface Propagation Using an Extrema Graph and Sorted Boundary Cell Lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, 1995.
9. Y. Livnat, H.-W. Shen, and C. R. Johnson. A Near Optimal Isosurface Extraction Algorithm Using the Span Space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.
10. W. E. Lorenson and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163–169, 1987.
11. R. Mersereau. The processing of hexagonally sampled two-dimensional signals. *Proceedings of the IEEE*, 67(3):930–946, June 1979.
12. C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of Marching Cubes. *Visual Computer*, 10:353–355, 1994.
13. B. Natarajan. On generating topologically consistent isosurfaces from uniform samples. *Visual Computer*, 11:52–62, 1994.
14. N. Neophytou and K. Mueller. Space-Time Points: 4D Splatting on Efficient Grids. In *Proceedings of Volume Visualization 2002*, 2002.
15. G. M. Nielson and B. Hamann. The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes. In *Proceedings of Visualization 1991*, pages 83–91. IEEE, 1991.
16. V. Pekar, R. Wiemker, and D. Hempel. Fast detection of meaningful isosurfaces for volume data visualization. In T. Ertl, K. Joy, and A. Varshney, editors, *Proceedings of IEEE Visualization 2001*, pages 223 – 230, 2001.
17. S. Tenginakai, J. Lee, and R. Machiraju. Salient iso-surface detection with model-independent statistical signatures. In T. Ertl, K. Joy, and A. Varshney, editors, *Proceedings of IEEE Visualization 2001*, pages 231 – 238, 2001.
18. T. Theußl, H. Hauser, and M. E. Gröller. Mastering windows: Improving reconstruction. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 101–108, 2000.
19. T. Theußl and T. Möller and M. E. Gröller. Optimal regular volume sampling. In T. Ertl, K. Joy, and A. Varshney, editors, *Proceedings of IEEE Visualization 2001*, pages 91–98, 2001.
20. G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computers and Graphics*, 23(4):583–598, Aug. 1999.
21. M. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D. R. Schikore. Contour Trees and Small Seed Sets for Isosurface Traversal. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 212–220, 1997.
22. J. Wilhelms and A. van Gelder. Topological Considerations in Isosurface Generation. *Computer Graphics*, 24(5):79–86, 1990.
23. J. Wilhelms and A. van Gelder. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.
24. G. Wyvill, C. McPheeters, and B. Wyvill. Data Structure for Soft Objects. *Visual Computer*, 2:227–234, 1986.



```
Method: March Cubes     Data/lobster.dat.raw
Cells:         39572    Tris/Cell:     2.117
Triangles:     83768    Isovalue:    127.500
```

a) Lobster on cubic grid, Marching Cubes

```
Method:          BCC    Data/lobster.dat.raw.bc
Cells:        142650    Tris/Cell:     1.285
Triangles:    183355    Isovalue:    127.500
```

b) Lobster on BCC grid, Marching Tetrahedra

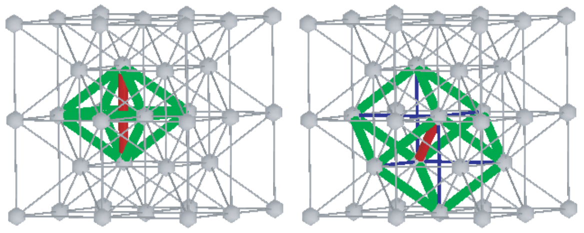
```
Method:    Octahedra    Data/lobster.dat.raw.bc
Cells:         44786    Tris/Cell:     3.190
Triangles:    142870    Isovalue:    127.500
```

c) Lobster on BCC grid, Modified Marching Octahedra

```
Method:          MMH    Data/lobster.dat.raw.bc
Cells:         34551    Tris/Cell:     2.229
Triangles:     77016    Isovalue:    127.500
```

d) Lobster on BCC grid, Modified Marching Hexahedra

Figure 8: *Four Lobsters*



(a) Octahedral BCC (b) Hexahedral BCC

Figure 9: *Grouping Tetrahedra*