Vis Comput (2009) 25: 423–430
DOI 10.1007/s00371-009-0330-8

ORIGINAL ARTICLE

Maximal independent set graph partitions for representations of body-centered cubic lattices

Kenny Erleben

Published online: 3 March 2009

Abstract A maximal independent set graph data structure for a body-centered cubic lattice is presented. Refinement and coarsening operations are defined in terms of set-operations resulting in robust and easy implementation compared to a quad-tree-based implementation. The graph only stores information corresponding to the leaves of a quad-tree thus has a smaller memory foot-print. The adjacency information in the graph relieves one from going up and down the quad-tree when searching for neighbors. This results in constant time complexities for refinement and coarsening operations.

Keywords Graph · Refinement · Coarsening · Body-centered cubic lattice

1 The body-centered cubic lattice

We present a new flexible and powerful graph data structure for a body-centered cubic (BCC) lattice. We believe the data-structure is well suited for interactive simulations, where a small memory foot-print and dynamic constant time operations are attractive. Our focus is on the data structure and not on the interactive simulation, nor on how to apply BBC lattice for solving problems. Our data structure is easy to implement, and the simplicity of only using set-operations reduces the chance of introducing bugs. Our prototype implementation was done within one hour by copying and pasting the set-operations into code.

K. Erleben (✉)
eScience Center, University of Copenhagen, Copenhagen, Denmark
e-mail: kenny@diku.dk

A BCC lattice [5] is one of the most common and simplest shapes found in crystals and minerals [3] and has many usages in computer graphics, for instance, adaptive meshing [6], physics-based animation [7, 10], and multi-resolution meshes [2]. A BCC lattice is similar to 4–8 subdivision [9], triangle quad-trees, and triangle bin-trees [4] used for adaptive terrains.

A quad-tree [8] implementation of a BCC lattice results in a nested model for a multi-resolution mesh [7]. However, a quad-tree implementation holds more generality than needed for a BCC lattice. Our contribution is a non-nested data structure for a BCC lattice.

Other 2D mesh-representation could be used, such as a half-edge data structure. The half-edge would store unneeded features of the BCC lattice and result in a much larger memory foot-print. Further, one needs book-keeping for performing refinement and coarsening operations without destroying the BCC lattice property. Our data structure does not need book-keeping and has a smaller memory foot-print. In fact, only one third of the edges and no faces need to be stored.

A BCC lattice consists of the nodes of a Cartesian grid along with cell centers. One can visualize the BCC lattice as two staggered grids, one grid made from the Cartesian grid nodes and another grid created from the cell-centers. A triangulation is created by connecting a node with the four closest neighbors from the other grid. An example is shown in Fig. 1.

This creates a regular tessellation with a simple topology. Thus we do not consider completely general triangulations in this paper. We will also restrict ourselves to the 2D domain. Observe that the triangulation is the Delaunay complex of the interlaced grid nodes and thus possesses all properties of a Delaunay triangulation.

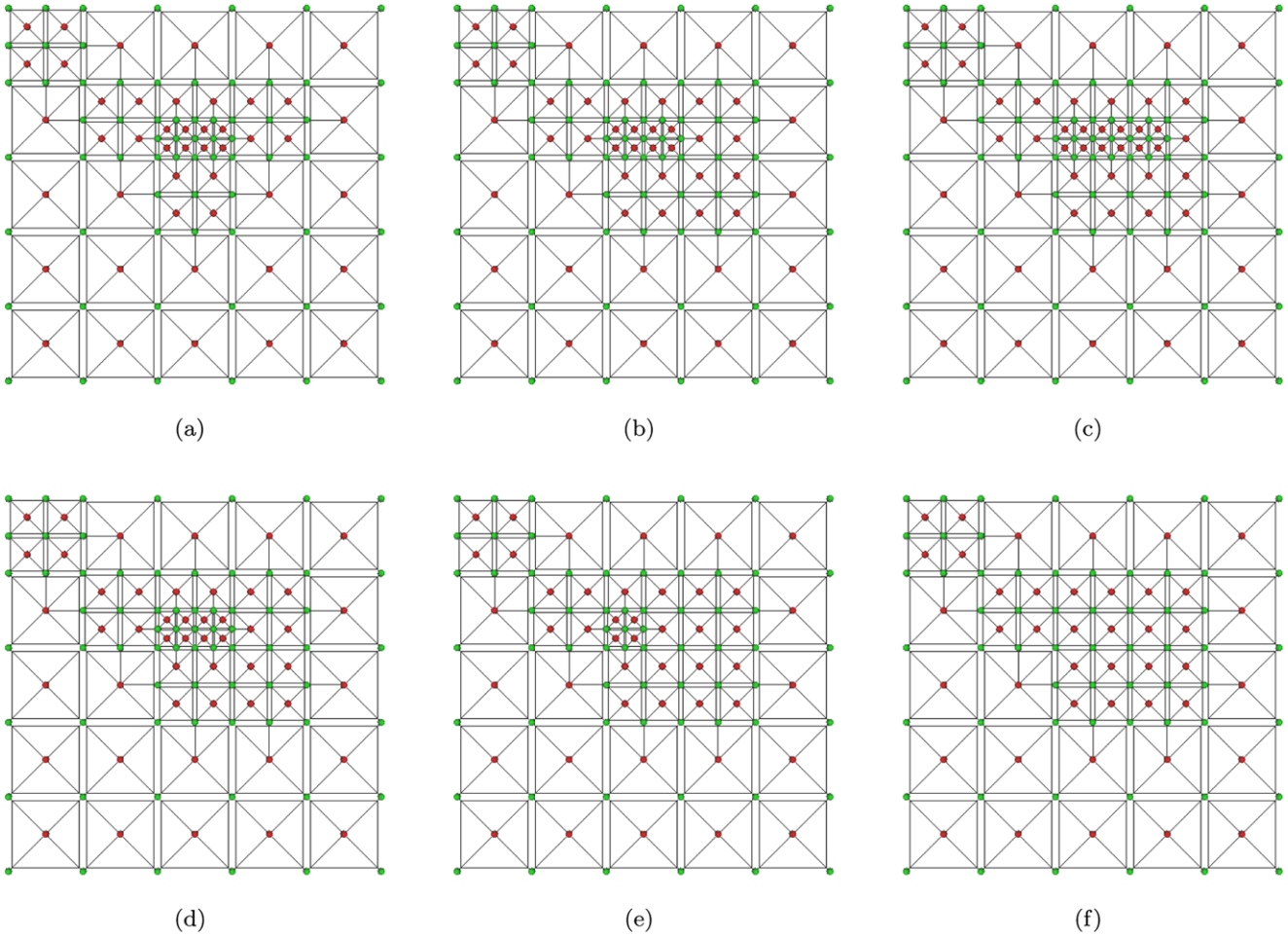

Fig. 1 Example of a body-centered cubic lattice. A fixed number of random refinement operations were done in (**a**)–(**c**). Hereafter a sequence of coarsening operations were done, (**d**)–(**f**)

One can refine the triangulation by replacing a cell with a subdivision of cells of half-size. This is illustrated in Fig. 1. Observe that T-junction nodes appear in-between the unrefined cells and the refined cell. Special attention must be paid to the junction nodes when creating the triangulation. In the example in Fig. 1 we have chosen to create triangles by creating edges from a cell-center to all nodes on the boundary of the containing Cartesian cell. This creates a star-shaped tessellation pattern of the cell.

If the cell is further refined, more junction nodes will appear on the boundary with the neighboring unrefined cells. In this case the star-tessellation strategy will in the limiting case create a lot of sliver triangles. This is an unwanted trait. Therefore we limit the refinement level of cells such that any two neighboring cells can have a difference in their refinement level of at most one. In this context a refinement level is the number of times a cell has undergone a refinement.

A BCC lattice can be implemented using an augmented octree/quad-tree data structure [6, 7, 10]. In this paper we will describe an alternative data structure based on mathematical topological set-operations.

2 The graph representation

Let $\mathcal{T}$ be a triangulation consisting of a set of vertices $\mathbf{V} = \{v_i\}$, a set of edges $\mathbf{E} = \{e_{ij}\}$, and a set of faces $\mathbf{F} = \{f_{ijk}\}$, where $e_{ij} = \{v_i v_j\}$ and $f_{ijk} = \{v_i v_j v_k\}$.

We seek a least committed data representation. The set of vertices is divided into two disjoint vertex sets called node-vertices, $\mathbf{N} \subset \mathbf{V}$, and center-vertices, $\mathbf{C} \subset \mathbf{V}$, where $\mathbf{N} \cap \mathbf{C} = \emptyset$. For a node-vertex $n \in \mathbf{N}$, we store a neighbor set, $\mathcal{N}(n)$, of indices to incident center-vertices. In a similar fashion, for each $c \in \mathbf{C}$, we store a neighbor set, $\mathcal{N}(c)$, to incident node-vertices. The neighbor sets of all the node-vertices and center-vertices represent a subset of edges in the triangulation. The edge e_{ij} is running between the vertex v_i and the vertex v_j. If v_i is a node-vertex and v_j is a center-vertex, then $v_i \in \mathcal{N}(v_j)$ and $v_j \in \mathcal{N}(v_i)$. Thus we do

not store edges where v_i and v_j both belong to the same vertex-set, **N** or **C**. Nor do we explicitly store the edge and face sets. Instead, these sets will be inferred from the node-vertex set, the center-vertex set, and the neighbor set information.

As an example, we will study how an instance of a triangulation from a BCC lattice can be stored in the above representation. Figure 1 shows an example. Here the nodes from the green lattice will make up the node-vertex set **N**, the nodes of the red lattice will correspond to the center-vertex set **C**, and the black diagonal edges would correspond to the neighbor sets. Observe that no horizontal or vertical edges are explicitly represented.

Because none of the red-edges is part of the triangulation, the center-vertex set is a maximal independent set [1]. A maximal independent set means that no node-vertex can be taken from **N** and added to **C** without creating an edge that is incident to more than one center-vertex. The node-vertex set is not a maximal independent set of the original triangulation due the fact that all green edges are part of the triangulation. However, by not representing any horizontal or vertical edges we have a graph where the center-vertices and the node-vertices are both maximal independent sets.

The green edges we have discarded can be generated by computing the edges of the convex hull of the neighbor sets of the node-vertices,

$$\bigcup_{\forall c \in \mathbf{C}} \mathbf{conv}\big(\mathcal{N}(c)\big). \tag{1}$$

The union of the edges from all the hulls corresponds to the vertical and horizontal green edges that are not part of the current representation. Thus we have shown that we have a maximum independent set partition of a graph which is equivalent to the triangulation of a BCC lattice. In the following we will outline the refinement and coarsening operations on this special case graph data structure.

2.1 The refinement operation

We specify a cell-vertex, c, corresponding to the cell that should be refined. Initially a logical test should be performed ensuring that the refinement level of any neighboring cells is larger than or equal to the current refinement level. If one of the neighboring cells has a refinement level less than the cell we want to refine, then the result would be a refinement difference greater than one, which is illegal.

We want to obtain all the vertices, **B**, on the boundary of the cell that is subject to refinement. These vertices are stored in the neighbor set of the cell-vertex, $\mathcal{N}(c)$. One may observe that the cardinality of the neighbor set of a junction-vertex is always 3. Thus we separate the neighbor vertices of the cell-vertex into pure node-vertices, **P**, and junction node-vertices, **J**:

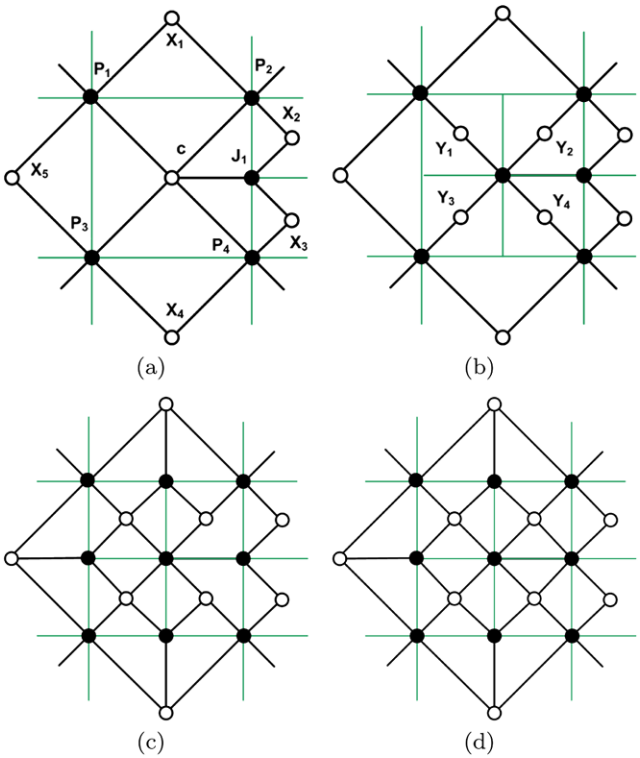

Fig. 2 The four steps of the refinement operation, (**a**) find the neighborhood sets, (**b**) creating sub cell centers and changing c into a node-vertex, (**c**) creating and connecting new junctions, and (**d**) adding junctions to neighboring cells. Observe that the neighbor sets of the neighboring cell-vertices have been extended with a single junction-vertex. The red grid has been hidden for clarity in some of the steps

$$\mathbf{B} = \mathcal{N}(c), \tag{2a}$$

$$\mathbf{J} = \big\{b_i | b_i \in \mathbf{B} \wedge \big\|\mathcal{N}(b_i)\big\| = 3\big\}, \tag{2b}$$

$$\mathbf{P} = B \setminus J. \tag{2c}$$

From the pure node-vertices, **P**, we can find all the neighboring cell-vertices, **X**. We take the union of all the neighbor sets of the pure node-vertices of the boundary,

$$\mathbf{X} = \left\{ \bigcup_{\forall b_i, b_j \in \mathbf{B}, i \neq j} \mathcal{N}(b_i) \cap \mathcal{N}(b_j) \right\} \setminus \{c\}. \tag{3}$$

At this point we have complete knowledge about the topology of the cell that should be refined. Figure 2(a) illustrates the vertex sets.

One can iterate over the vertices in **X** and verify that the refinement level is not less than the refinement level of c. If the test passes, we can continue the refinement operation.

The cell c is subdivided into four new cells. This action will convert c into a node-vertex and change the neighbor set of c to hold indices of four new cell-vertices,

$\mathbf{Y} = \{y_1, \ldots, y_4\}$, corresponding to the four new cells in the subdivision.

$$\mathbf{Y} = \{y_1, \ldots, y_4\}, \tag{4a}$$

$$\mathbf{C} = \mathbf{C} \cup \mathbf{Y}, \tag{4b}$$

$$\mathcal{N}(c) = \mathbf{Y}, \tag{4c}$$

$$\mathcal{N}(j_i) = \mathcal{N}(j_i) \setminus \{c\} \quad \forall j_i \in \mathbf{J}, \tag{4d}$$

$$\mathcal{N}(p_i) = \mathcal{N}(p_i) \cup \{y_i\} \setminus \{c\} \quad \forall p_i \in \mathbf{P}, \tag{4e}$$

$$\mathcal{N}(y_i) = \{p_i, c\} \quad \forall y_i \in \mathbf{Y}, \tag{4f}$$

$$\mathbf{C} = \mathbf{C} \setminus \{c\}, \tag{4g}$$

$$\mathbf{N} = \mathbf{N} \cup \{c\}. \tag{4h}$$

The set of pure node-vertices always have $\|\mathbf{P}\| = 4$ and similarly $\|\mathbf{Y}\| = 4$. We have exploited this to create a one-to-one mapping between $y_i \in Y$ and $p_i \in P$. This is illustrated in Fig. 2(b).

Next we create the missing junction-vertices. We already know how many junction-vertices that are currently part of the cell, so we can create the correct number of new junction-vertices needed,

$$\mathbf{J}^+ = \left\{j_1^+, \ldots, j_{4-\|\mathbf{J}\|}^+\right\}, \tag{5a}$$

$$\mathbf{N} = \mathbf{N} \cup \mathbf{J}^+. \tag{5b}$$

Now for all $p_i, p_j \in P$ with $i \neq j$ and $\|\mathcal{N}(p_i) \cap \mathcal{N}(p_j)\| = 1$, we carry out the steps below for $k = 1, \ldots, \|\mathbf{J}^+\|$:

$$x_k = \mathcal{N}(p_i) \cap \mathcal{N}(p_j), \tag{6a}$$

$$\mathcal{N}(x_k) = \mathcal{N}(x_k) \cup \left\{j_k^+\right\}, \tag{6b}$$

$$\mathcal{N}(j_k^+) = \{x_k, y_i, y_j\}, \tag{6c}$$

$$\mathcal{N}(y_i) = \mathcal{N}(y_i) \cup \left\{j_k^+\right\}, \tag{6d}$$

$$\mathcal{N}(y_j) = \mathcal{N}(y_j) \cup \left\{j_k^+\right\}, \tag{6e}$$

where $j_k^+ \in \mathbf{J}^+$. The newly created junction-vertices have been connected into the graph. Figure 2(c) illustrates the operation on the example from Fig. 2(b).

The final step in the refinement operation is to handle the old junction-vertices from the set $\mathbf{J}$. These node-vertices need to be connected to the newly created cell-vertices. If $\|\mathcal{N}(j_k) \cap \mathcal{N}(p_i)\| = 1$ for $j_k \in \mathbf{J}$ and $p_i \in \mathbf{P}$, then

$$\mathcal{N}(j_k) = \mathcal{N}(j_k) \cup \{y_i\}, \tag{7a}$$

$$\mathcal{N}(y_i) = \mathcal{N}(y_i) \cup \{j_k\}. \tag{7b}$$

This completes the refinement operation. The final outcome of our example is shown in Fig. 2(d).

2.2 The coarsening operation

Coarsening is the inverse operation of refinement. The coarsening operation is initiated by specifying a node-vertex, n, that must be turned into a cell-vertex. The node-vertex cannot be allowed to be a junction-vertex. Thus we must check whether $\|\mathcal{N}(n)\| = 4$. If this is the case, we can continue the coarsening operation.

We find the four cell-vertices, $\mathbf{Y}$, corresponding to the sub-cells that should collapse into one larger cell. Hereafter, we use the neighbor sets of the found cell-vertices to determine all node-vertices, $\mathbf{B}$, currently lying on the boundary of the resulting larger cell,

$$\mathbf{Y} = \mathcal{N}(n), \tag{8a}$$

$$\mathbf{B} = \bigcup_{\forall y_i \in \mathbf{Y}} \mathcal{N}(y_i) \setminus \{n\}. \tag{8b}$$

All the $\mathbf{Y}$-vertices will be deleted, thus their connectivity to the other vertices in the graph must be removed. We start by unconnecting them from all the node-vertices on the boundary of the larger cell:

$$\mathcal{N}(b_i) = \mathcal{N}(b_i) \setminus \mathbf{Y} \quad \forall b_i \in \mathbf{B}, \tag{9a}$$

$$\mathcal{N}(y_i) = \emptyset \quad \forall y_i \in \mathbf{Y}, \tag{9b}$$

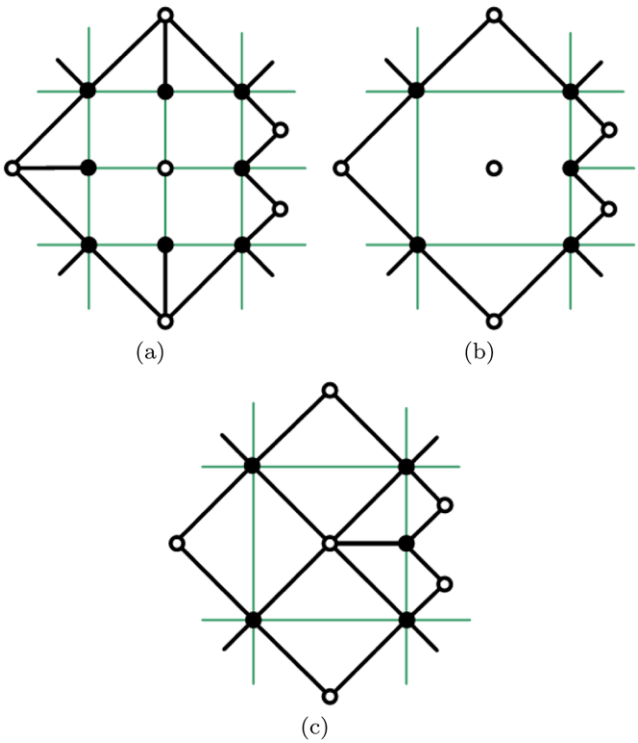

Fig. 3 The steps of the coarsening operation. In (**a**) all the cell-vertices of the sub-cells are unconnected and deleted, (**b**) all unneeded junction-vertices are unconnected and deleted, and (**c**) the final state of the coarsening operation

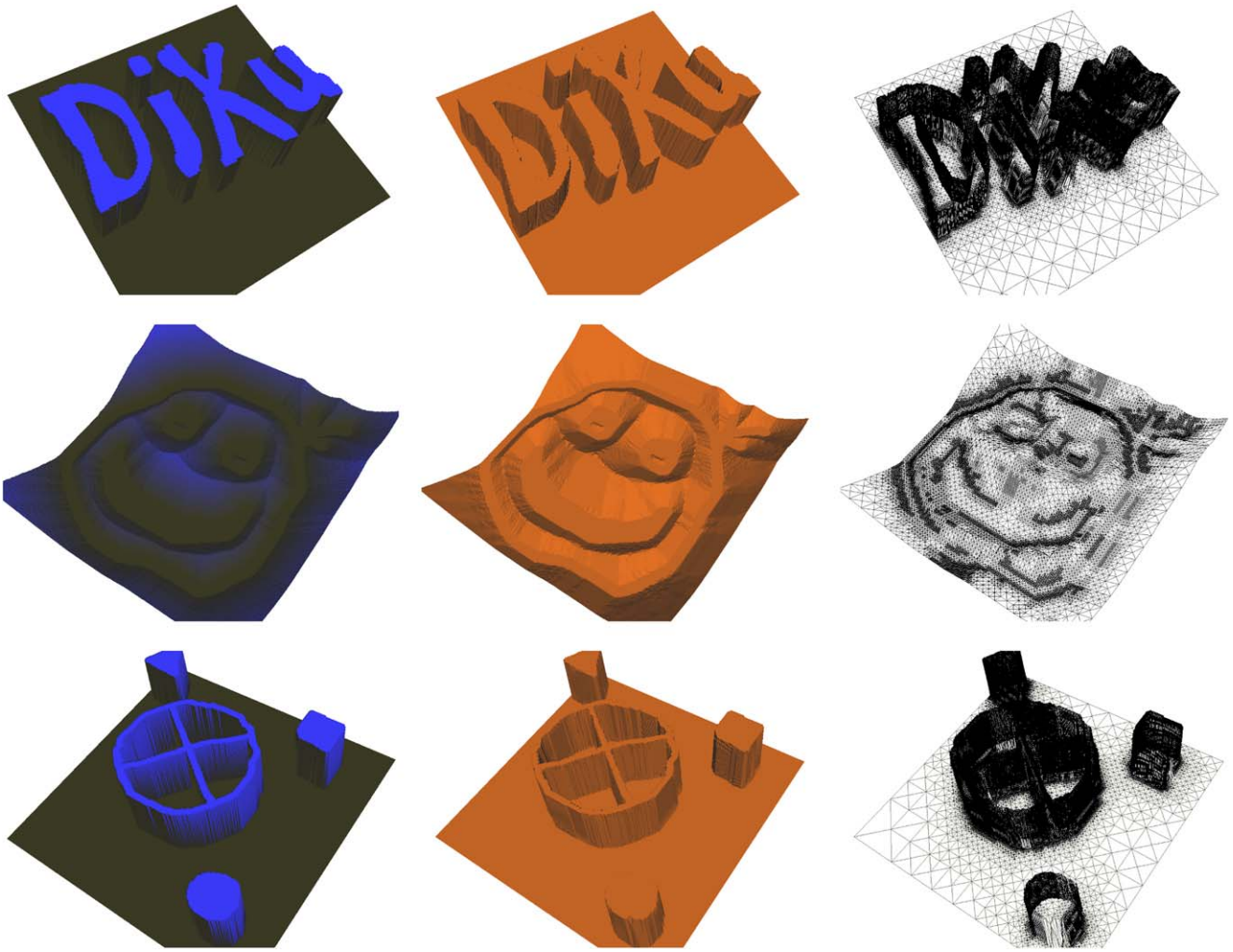

Fig. 4 A body-centered cubic lattice (*middle* and *right columns*) is adaptively refined to fit an underlying height-field (*left column*)

$$\mathcal{N}(n) = \emptyset, \tag{9c}$$

$$\mathbf{N} = \mathbf{N} \setminus \{n\}, \tag{9d}$$

$$\mathbf{C} = \mathbf{C} \cup \{n\}, \tag{9e}$$

$$\mathbf{C} = \mathbf{C} \setminus \mathbf{Y}. \tag{9f}$$

This effectively changes the node-vertex n into a cell-vertex. Now all the $\mathbf{Y}$-vertices can be deleted. In Fig. 3(a) we have shown the first step of the coarsening operation on the example from Fig. 2(d).

Next we determine if any junction-vertices should be deleted. These junction-vertices can be identified through the cardinality of their neighbor sets:

$$\mathbf{J}^- = \left\{ b_i | b_i \in \mathbf{B} \wedge \left\| \mathcal{N}(b_i) \right\| = 1 \right\}, \tag{10a}$$

$$\mathbf{B} = \mathbf{B} \setminus \mathbf{J}^-, \tag{10b}$$

$$\mathbf{N} = \mathbf{N} \setminus \mathbf{J}^-. \tag{10c}$$

Hereafter all the junction-vertices that are about to be deleted are unconnected. For each $j_i^- \in \mathbf{J}^-$,

$$x_i = \mathcal{N}(j_i^-), \tag{11a}$$

$$\mathcal{N}(x_i) = \mathcal{N}(x_i) \setminus \{j_i^-\}, \tag{11b}$$

$$\mathcal{N}(j_i^-) = \emptyset, \tag{11c}$$

$$\mathbf{N} = \mathbf{N} \setminus \mathbf{J}^-. \tag{11d}$$

Now all $\mathbf{J}^-$-vertices are deleted. The step is illustrated in Fig. 3(b).

The final step consists in connecting the cell-vertex n with the remaining boundary vertices of the larger cell:

$$\mathcal{N}(n) = \mathbf{B}, \tag{12a}$$

$$\mathcal{N}(b_i) = \mathcal{N}(b_i) \cup \{n\} \quad \forall b_i \in \mathbf{B}. \tag{12b}$$

This completes the coarsening operation. Figure 3(c) shows the end-result.

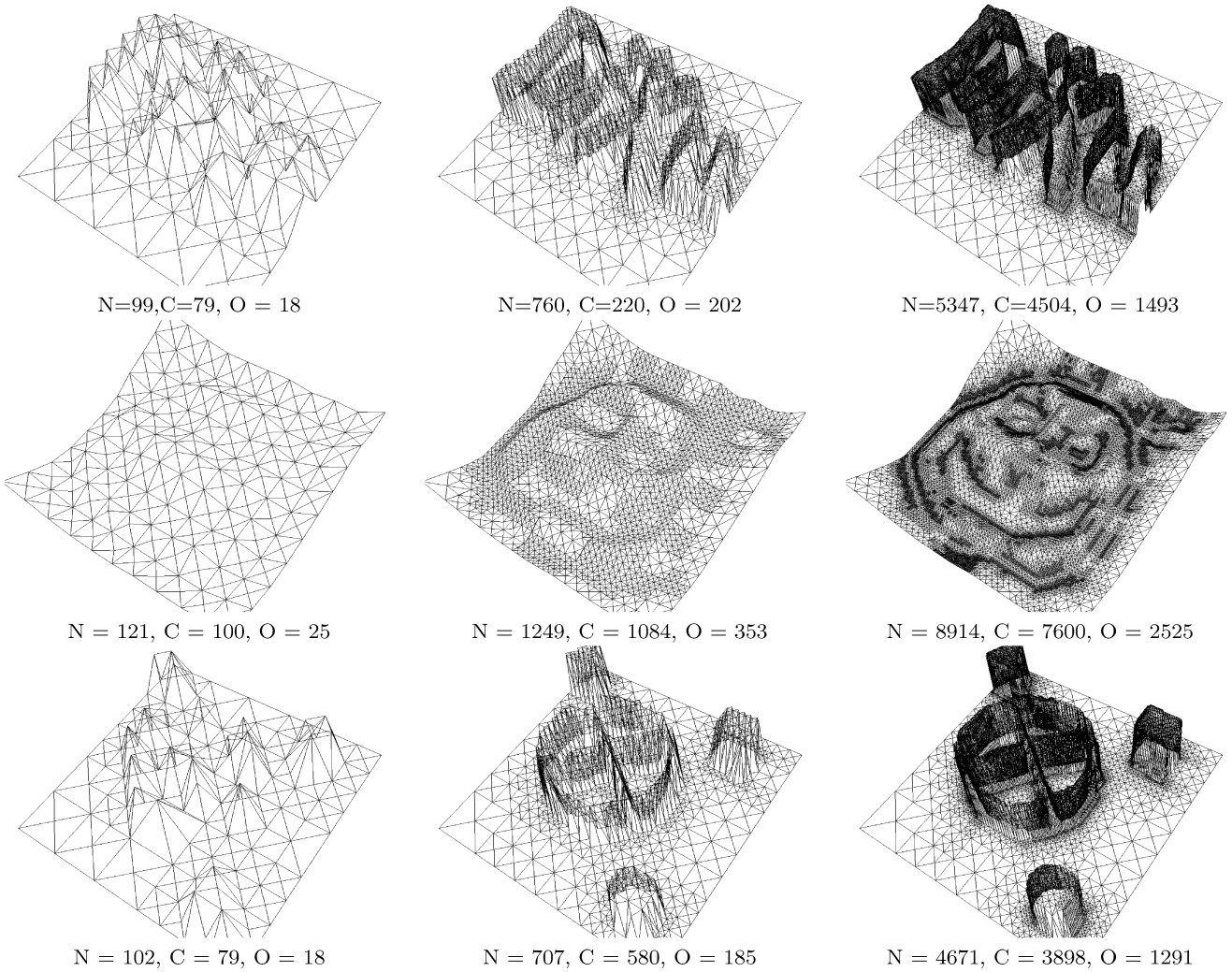

Fig. 5 A body-centered cubic lattice is adaptively refined to fit an underlying height-field. Refinement levels are increased from left-to-right. Number of grid nodes (N), cell-centers (C), and total number of refinement operations (O) are shown

3 Results

In Fig. 1 we show a subset of operations testing the robustness of our implementation. Refinement operations are first used to create a random tessellation. Hereafter coarsening operations are used to invert the effect of all the refinement operations. As the example illustrates, the refinement and coarsening operations are truly inverses of each other.

We have also applied the data-structure for an adaptive meshing application. We use the BCC lattice to adaptively tessellate a high-resolution height-field. The height-field is created from a 256-by-256 pixel image. An initial 6-by-6 BCC lattice is overlaid the height-field. Hereafter we test each cell of the BCC lattice whether it should be refined or not. The test is independent of the data-structure, and we choose a naive test since our focus is on the data structure rather than on surface modeling or refinement. The test is based on how well a fixed number of random height samples within a cell can be interpolated from the enclosing grid nodes. Several sweeps are done over all the vertices until a visual pleasing result is obtained. Figure 4 shows the initial height fields of 3 examples together with adaptive tessellations after having performed 5–6 sweeps over the cell-vertices.

In Fig. 5 we have illustrated intermediate tessellation results. As demonstrated, the graph data structure can be used as a spatial data structure.

The tessellations were computed using a Dell Precision M90 laptop, Intel(R) Core Duo CPU, 2.33 GHz, 2 GB of RAM. Running times ranged from 1–5 seconds, no special optimization was done, and only a single core was utilized. Although not the most efficient choice, we used STL algorithms for set-operations in our implementation.

To demonstrate that the 2D BCC lattice is not restricted to height-fields, we performed a cloth simulation where coars-

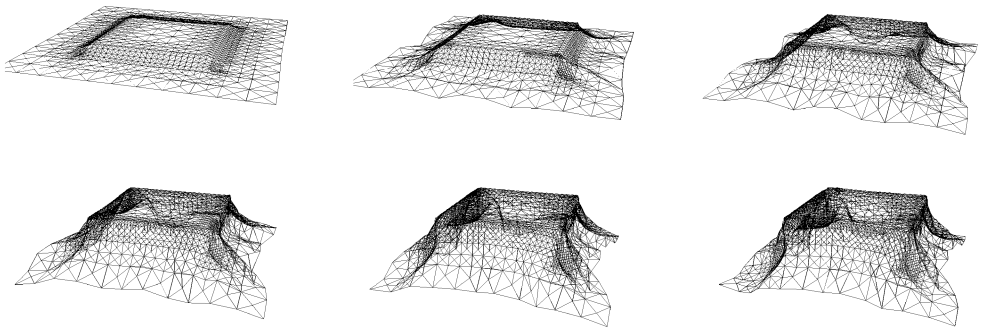

Fig. 6 A piece of cloth is being represented by a body-centered cubic lattice. The refinement and coarsening are based on mean curvature at contact points. Notice the finer resolution at the sharp edges of the impacting box-object

ening and refinement of the cloth mesh was done based on a mean curvature measure at the contact points. Resulting frames of the animation are shown in Fig. 6.

4 Discussion

In this paper we have introduced a graph-based data structure for a body-centered cubic lattice. Further we have developed both refinement and coarsening operations for the new data structure and shown examples illustrated the reversibility of the two operations. An adaptive meshing and a cloth simulation application demonstrate the usability of the graph-based data structure for real problems.

In this paper we have only considered the two-dimensional case. We believe that our ideas could be extended to three dimensions. This extension would require changes in the algorithm in regards to the handling of junctions. In higher dimensions junctions may appear with different cardinalities, ranging from 3 to 7. Further the green edges in three dimensions will be given by the edges of the faces of the convex hull of the neighbor sets of the node node-vertices.

In comparison with a quad-tree-based implementation, one would require $\mathcal{O}(n \log n)$-space complexity for storing the BCC lattice, where n is the number of cells in the BCC lattice. A vertex can have at most 8 vertices in its neighbor set, and we therefore consider the storage per vertex to be constant. Thus our graph-based data structure requires $\mathcal{O}(n)$-space complexity. We consider all set-operations as being constant time operations for larger meshes due to the small-size upper bound on the vertex neighbor sets. Refinement and coarsening only consist of a sequence of such set-operations. Refinement and coarsening operations can therefore be considered to be of $\mathcal{O}(1)$ time-complexity. In a quad-tree data structure one may have to ascend all the way to the root before being able to descending to a neighboring cell. Since neighboring cells must be located to deal correctly with junction nodes, this implies at least a $\mathcal{O}(\log n)$ time-complexity for refinement and coarsening operations in a quad-tree-based implementation.

If an application needs explicit access to the face sets, then our graph-based data structure needs to infer the face sets from the node, center, and neighbor sets. This requires the computation of a convex hull, which may be too costly during run-time of an interactive application.

The data locality of the graph-based data structure may also prove to be useful if one seeks a GPU or CELL CPU implementation. Due to lack of hardware, we have not pursued this.

References

1. Cormen, T.H., Leiserson, C.E., Riverst, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
2. De Floriani, L., Magillo, P.: Multiresolution mesh representation: Models and data structures. In: Tutorials on Multiresolution in Geometric Modelling, pp. 363–418. Springer, New York (2002)
3. Delgado-Friedrichs, O., O'Keeffe, M.: Crystal nets as graphs: Terminology and definitions. J. Solid State Chem. **178**(8) (2005)
4. Duchaineau, M., Wolinsky, M., Sigeti, D.E., Miller, M.C., Aldrich, C., Mineev-Weinstein, M.B.: Roaming terrain: real-time optimally adapting meshes. In: VIS '97: Proceedings of the 8th Conference on Visualization '97, Los Alamitos, CA, USA, 1997, pp. 81–88. IEEE Computer Society Press, Los Alamitos (1997)
5. Entezari, A., Van De Ville, D., Möller, T.: Practical box splines for reconstruction on the body centered cubic lattice. IEEE Trans. Vis. Comput. Graph. **14**(2), 313–328 (2008)
6. Labelle, F., Shewchuk, J.R.: Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. ACM Trans. Graph. **26**(3), 57 (2007)
7. Molino, N., Bridson, R., Teran, J., Fedkiw, R.: A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In: Proceedings of the 12th International Meshing Roundtable, pp. 103–114, Santa Fe, New Mexico, USA, September 2003
8. Samet, H.: Spatital data structures. In: SIGGRAPH '07: ACM SIGGRAPH 2007 Courses, New York, NY, USA, 2007. ACM, New York (2007)
9. Velho, L., Zorin, D.: 4–8 subdivision. Comput. Aided Geom. Des. **18**(5), 397–427 (2001)
10. Wojtan, C., Turk, G.: Fast viscoelastic behavior with thin features. In: SIGGRAPH '08: ACM SIGGRAPH 2008 Papers, New York, NY, USA, 2008, pp. 1–8. ACM, New York (2008)

Kenny Erleben After his studies Erleben was employed as full-time researcher in the Company 3DFacto A/S for a period of 10 months. In 2001 Erleben started on his PhD studies. During 2004 Erleben stayed 3 months at the Department of Mathematics, University of Iowa. Hereafter he received his PhD degree in the beginning of 2005, and finally late 2005 Erleben was employed as an Assistant Professor at the Department of Computer Science, University of Copenhagen. In 2008 Erleben received NVIDIA professor partnership and was appointed Innovation Ambassador at the Faculty of Science in 2007. Research Interests: Creating nonlinear nonsmooth mathematical models and finding efficient numerical methods for solving them, in particular, in the fields of Multibody Dynamics, Inverse Kinematics, and Collision Detection.