Journal of Spatial Information Science 00 (2020) 1–12

Journal of Spatial Information Science

Supplementary Material for Modelling Orebody Structures: Block Merging Algorithms and Block Model Spatial Restructuring Strategies Given Mesh Surfaces of Geological Boundaries

Raymond Leung[*]

Australian Centre for Field Robotics, The University of Sydney
Sydney Robotics Hub J18, Sydney, NSW 2006, Australia

1. Context

This document elaborates on the computational aspects of "Modelling Orebody Structures: Block Merging Algorithms and Block Model Spatial Restructuring Strategies Given Mesh Surfaces of Geological Boundaries" (Leung, 2020) which describes how spatial structures can be captured in a block model given triangle mesh surfaces that describe geological boundaries. Central to that work is a flexible framework for updating the spatial structure of a block model given new surfaces and the ability to retain or overwrite existing domain classifications (block labels) in an iterative manner as newer information becomes available. Appendix A describes a method for finding blocks in the model that intersect with triangular patches on a given surface. This is used in (Leung, 2020) to identify areas where *model refinement* is needed to accurately reflect the location of boundaries and more closely approximate the curvature of said surfaces. Appendix B describes the concept of ray-tracing which is used to establish the location of blocks relative to the surface(s) in the *block tagging* system component in (Leung, 2020). Appendix D deals with the technical aspects of block merging and discusses various considerations fundamental to its design. This in-depth discussion explains the differences between two block merging conventions, the constraints, the block merging optimisation objective, and how different scanning sequences are implemented in practice. It should be noted that the overall block merging technique can be applied to areas outside of geoscience as shown in Appendix C, to reduce redundancy / fragmentation in a parent-grid aligned block model, and in instances where 3D segmentation is desired given some triangle mesh surface for an object. Appendix E provides the pseudocode for the coordinate-ascent inspired block merging algorithms which is the main contribution of (Leung, 2020). Finally, detailed commentary and results on octree subblocking are given in Appendix F and Appendix G.

Appendix A. Akenine-Möller method for block triangle overlap detection

Assessment for "block-triangle" intersection involves at most 13 tests:

- 3 along the x, y, z axes, the orthonormal bases are denoted $\mathbf{e}_0 = (0, 0, 1), \mathbf{e}_1 = (0, 1, 0), \mathbf{e}_2 = (0, 0, 1)$
- 9 for cross-products between edges of A and B, viz., cross$(\mathbf{e}_i, \mathbf{f}_j)$ for $i, j \in \{0, 1, 2\}$
- 1 for the normal of the triangle based on cross$(\mathbf{f}_i, \mathbf{f}_j)$ given vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$, edge vectors $\mathbf{f}_i = \mathbf{v}_{\text{mod}(i+1,3)} - \mathbf{v}_i$

Suppose a triangle has vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^3$, a block has centroid $\mathbf{b}_k = (b_x, b_y, b_z)$ and dimensions $\boldsymbol{\Delta}_k = (\Delta_x, \Delta_y, \Delta_z)$, the SAT test for axes x, y, z asserts "no overlap" if $\mathbf{v}'_{\min}[c] > \boldsymbol{\Delta}_k[c]/2$ or $\mathbf{v}'_{\max}[c] < -\boldsymbol{\Delta}_k[c]/2$ for any $c \in \{x, y, z\}$ where $\mathbf{v}'_{\min}$ and $\mathbf{v}'_{\max}$ represent the minimum and maximum coordinates of the translated vertices, $\mathbf{v}'_i = \mathbf{v}_i - \mathbf{b}_k$, after the block centroid is subtracted from the triangle vertices.

The SAT test for cross$(\mathbf{e}_i, \mathbf{f}_j)$ exploits the properties of axis-aligned blocks. Its efficiency derives from terms cancellation in the cross-product expansion when the geometry of interest is limited to axis-aligned prisms and triangles. This uses only simple algebra; the relevant formulas may be found in (Akenine-Möller, 2001).

The last SAT test for plane-block overlap requires $p_{\min} = \text{dot}(\hat{\mathbf{n}}, \boldsymbol{\delta}_{\min}) + d$ and $p_{\max} = \text{dot}(\hat{\mathbf{n}}, \boldsymbol{\delta}_{\max}) + d$ to be computed, where $\hat{\mathbf{n}} = \mathbf{n}/\|\mathbf{n}\|$ is the unit length plane normal, $\mathbf{n} = (a, b, c)$, d is the plane distance from origin, assuming the plane passing through the triangle is described by the equation $ax+by+cz+d = 0$. The quantities $\boldsymbol{\delta}_{\min}[c] = (1 - 2 \times \mathcal{I}(\mathbf{n}[c] > 0)) \cdot \boldsymbol{\Delta}_k[c]/2$ and $\boldsymbol{\delta}_{\max}[c] = (2 \times \mathcal{I}(\mathbf{n}[c] > 0) - 1) \cdot \boldsymbol{\Delta}_k[c]/2$ evaluate to $\pm\boldsymbol{\Delta}_k[c]/2$. The test asserts "no overlap" if $p_{\min} > 0$ or $p_{\max} < 0$.

[*]Corresponding author
Email address: raymond.leung@sydney.edu.au (Raymond Leung)

Appendix B. Side-of-surface determination via ray tracing

Ray tracing is a well known technique in the computer graphics community (Dietrich et al., 2007). In the affiliated paper (Leung, 2020), it is used to establish where a block is located with respect to one or more triangle mesh surfaces, rather than for rendering purpose. A ray emanating from a block (specifically, its centroid) is casted in some specified direction.[1] The idea is to count the number of intersections between this ray and the relevant surface. An even number of intersections (including 0) result when the block is located above (respectively, outside) an open (respectively, closed) surface, and an odd number of interesections is interpreted as below (respectively, inside) the surface. The tests are based on the Möller–Trumbore algorithm (Möller and Trumbore, 2005) which is explained below.

Appendix B.1. Intersection between a ray and a plane

A ray extending from $\mathbf{p}_0$ to $\mathbf{p}_1$ intersects with a plane $\pi(\mathbf{v}_A, \mathbf{n})$ that passes through $\mathbf{v}_A \in \mathbb{R}^3$, with normal $\mathbf{n} = \mathbf{v}_A \times \mathbf{v}_B$, at $\mathbf{p}_{\text{intersect}} = \mathbf{p}_0 + \lambda(\mathbf{p}_1 - \mathbf{p}_0)$ when $\lambda \in [0, 1]$ where

$$\lambda = \frac{\hat{\mathbf{n}} \cdot (\mathbf{v}_A - \mathbf{p}_0)}{\hat{\mathbf{n}} \cdot (\mathbf{p}_1 - \mathbf{p}_0)} \tag{B.1}$$

A picture of this is shown in Fig. B.1

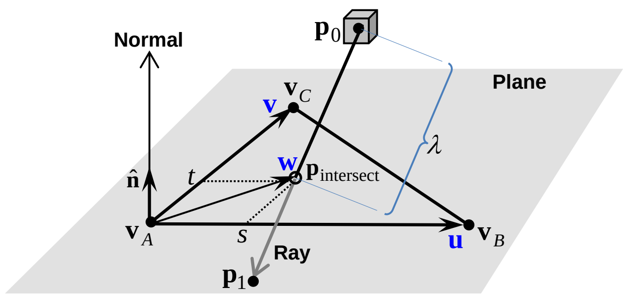


Figure B.1: Ray-triangle intersection analysis

- When $\lambda < 0$, the ray does not intersect with the triangle described by vertices $\mathbf{v}_A$, $\mathbf{v}_B$, $\mathbf{v}_C$ and plane $\pi(\mathbf{v}_A, \mathbf{n})$.
- When the denominator in (B.1) is zero, the ray is parallel to the triangle's plane. If the numerator is also zero, the ray intersects with the face of the triangle along a line. Otherwise, there is no intersection.

Appendix B.2. Intersection between a ray and a triangle

When $\lambda \in [0, 1]$, the ray intersects with the triangle at $\mathbf{p}_{\text{intersect}} = \mathbf{v}_A + s\mathbf{u} + t\mathbf{v}$ *if* the barycentric coordinates s and t (see Fig. B.1) satisfy $s \geq 0, t \geq 0$ and $s + t \leq 1$ where

$$\mathbf{u} = \mathbf{v}_B - \mathbf{v}_A, \quad \mathbf{v} = \mathbf{v}_C - \mathbf{v}_A \tag{B.2}$$

$$s = \frac{(\mathbf{u} \cdot \mathbf{v})(\mathbf{w} \cdot \mathbf{v}) - (\mathbf{v} \cdot \mathbf{v})(\mathbf{w} \cdot \mathbf{u})}{\Delta} \tag{B.3}$$

$$t = \frac{(\mathbf{u} \cdot \mathbf{v})(\mathbf{w} \cdot \mathbf{u}) - (\mathbf{u} \cdot \mathbf{u})(\mathbf{w} \cdot \mathbf{v})}{\Delta} \tag{B.4}$$

$$\Delta = (\mathbf{u} \cdot \mathbf{v})^2 - (\mathbf{u} \cdot \mathbf{u})(\mathbf{v} \cdot \mathbf{v}) \tag{B.5}$$

$$\mathbf{w} = \mathbf{p}_{\text{intersect}} - \mathbf{v}_A \tag{B.6}$$

This involves only five distinct inner products, and the quantities ($\mathbf{u} \cdot \mathbf{u}, \mathbf{v} \cdot \mathbf{v}$ and $\mathbf{u} \cdot \mathbf{v}$) may be precomputed as they are independent of $\mathbf{p}_{\text{intersect}} \in \mathbb{R}^3$, unlike $\mathbf{w}$ which is a function of the block centroid and ray direction.

Appendix B.3. Practicalities

Degenerate conditions must be handled to obtain proper results. First, when a ray intersects a surface at a common edge or vertex shared by multiple triangles, one needs to be careful that over-counting does not occur. In our implementation, a unique set of intersecting points is maintained for each ray to ensure the same intersecting point is not repeated. Second, when the denominator and numerator in (B.1) are both zero, λ is undefined, as the ray lies on the face of a triangle. This can be overcome by changing the direction slightly for the casted ray. Finally, triangles that collapse to a line segment or a single point need to be removed from the test surface since $\Delta \to \infty$ when either edge vector $\mathbf{u} = \mathbf{0}$ or $\mathbf{v} = \mathbf{0}$ in (B.2) and $\Delta \to 0$ when $\mathbf{u}$ and $\mathbf{v}$ are parallel. Users may wish to perform surface integrity checks as a preprocessing step to eliminate these conditions.

Appendix C. Demonstration on the Stanford Bunny

The block merging technique described in Algorithm 2 is applicable to more complex surfaces outside the geoscience domain. Fig. C.2 (a) shows a triangular mesh surface (McGuire, 2017) of the terracotta bunny obtained using multiple range scanners at the Stanford Computer Graphics Laboratory (Turk and Levoy, 2014). Fig. C.2 (b) shows a highly fragmented block model created by block decomposition without consolidation. In an effort to closely approximate the surface, numerous blocks at the minimum block size were produced near the surface. Fig. C.2 (c) shows a reduction in block density and increase in clarity as blocks are merged under the "dissolve sub-block boundaries" convention (see Appendix D.7). This resulted in a more efficient block representation (3D segmentation) of the object.

Appendix D. Extended discussion about using block merging to reduce fragmentation

The adjustments foreshadowed in Section 5 of the paper (Leung, 2020) improve both the robustness and accuracy of the block model spatial restructuring system which utilises at least one surface. This section considers how the block consolidation component can be extended to serve the needs of a *block merging* application where the key objective is to coalesce blocks

[1]For an open surface, this direction might be the upward (positive) direction specified in the tagging instructions. For a closed surface, the direction matters little, it is generally taken as the outward normal for the surface.

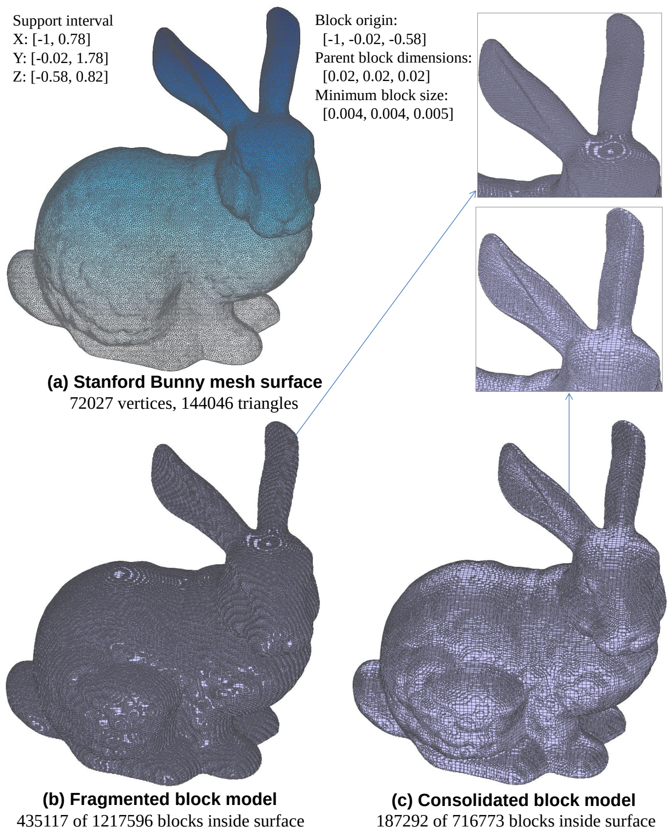

Support interval
X: [-1, 0.78]
Y: [-0.02, 1.78]
Z: [-0.58, 0.82]
Block origin:
[-1, -0.02, -0.58]
Parent block dimensions:
[0.02, 0.02, 0.02]
Minimum block size:
[0.004, 0.004, 0.005]
(a) Stanford Bunny mesh surface
72027 vertices, 144046 triangles
(b) Fragmented block model
435117 of 1217596 blocks inside surface
(c) Consolidated block model
187292 of 716773 blocks inside surface

Figure C.2: Block merging applied to Stanford Bunny to reduce block fragmentation. Zoom in to see individual blocks.

in a fragmented block model without any input surface. This extension builds upon the ideas described in Section 2.3. The characteristics and constraints of the problem will be described next. Henceforth, the established framework for block model *spatial restructuring using surfaces* from Section 5 and new *block merge* application will be abbreviated as SRUS and BM, respectively.

Appendix D.1. Problem description

In block model spatial restructuring using surfaces (SRUS), merging follows block-surface intersection detection and block decomposition, so we know precisely which input block (parent) a sub-block (cell) comes from. These input blocks may have different dimensions, particularly if the pipeline is repeated when individual surfaces are processed in cascade (see example in Section 4.1 where the output from the first iteration becomes the input in the second iteration). For the application envisaged in surface-free *block merge* (BM), the input contains only the labels, locations and dimensions of sub-blocks which are integer multiples of the minimum block size. Whilst the parent block and origin continues to provide a uniform grid structure that covers the 3D space, these parent-blocks have constant dimensions and only exist on a conceptual level for the purpose of grouping together the sub-blocks. Furthermore, ray-casting needs not be performed to determine which side of a surface a block is located, since the input provides domain labels for each block. The goal of BM is to consolidate the input blocks into larger rectangular prisms to minimise fragmentation.

Appendix D.2. Constraints

Before describing the constraints, it is instructive to first explain the spatial hierarchy and understand the assumptions. Fig. D.3 illustrates the relationship between parent block, cells and input blocks (sub-blocks) of intermediate scale. Conceptually, the whole 3D space is spanned by *parent blocks* which represent uniform, non-overlapping tiles positioned with respect to the anchor point, *block origin*. Each parent block may be identified by an index $\mathbf{p} = (p_x, p_y, p_z)$ obtained via uniform quantisation given the origin $\mathbf{o} = (o_x, o_y, o_z)$ and parent block size, (P_x, P_y, P_z). Each parent has internal structure — each is divided by the *minimum block size* into $(n_x \times n_y \times n_z)$ *cells* in the same manner. A cell is the smallest spatial unit. The cell "walls" dictate what type of merges are possible within a parent block. All input blocks and merged blocks must adhere to this structure, i.e., each consisting of one or more whole cells.

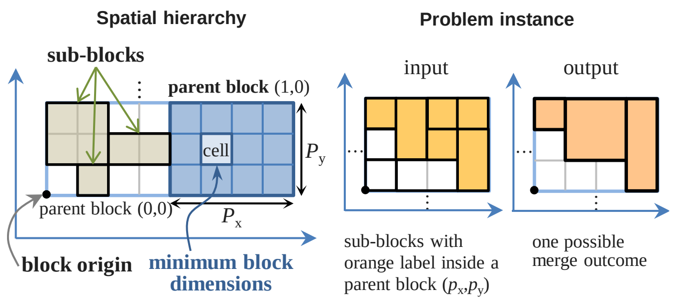


Figure D.3: Block merging spatial hierarchy

The assumptions are: 1) all input sub-blocks have dimensions which are integer-multiples of the minimum block; 2) all blocks must be rectangular prisms; 3) no sub-block straddles the boundary of any parent block; 4) edges of input and merged blocks must align perfectly with the internal grid lines of the parent block to which they belong; 5) only sub-blocks from the same class and parent may be merged.

Appendix D.3. Broad strategy

Beside some changes to the cell-expansion feasibility test, the block consolidation strategy based on coordinate-ascent merging is almost directly applicable to this problem. At a high level, the strategy comprises the following steps.

1. Establish an input sub-block to parent block mapping.
2. Divide and conquer (compartmental processing)
    - Each problem instance is restricted to a set of input blocks associated with (indexed by) a parent block. This is highly amendable to parallel processing.
3. Within each parent block, process each category (collection of input blocks with the same class label) in turn.
    - The position / extent of sub-blocks undergoing consolidation are maintained by a 3D cell occupancy map and stateful objects.

4. A modified coordinate-ascent merging algorithm is used to merge blocks from the same parent and class.
    - Feasibility of cell expansion is governed by specific rules which depend on the merging convention. However, the general goal remains the same, it still cycles through the x, y and z-coordinate one-by-one to consider if incremental expansion is possible.

Appendix D.4. Feasibility of cell expansion

For a parent block with cell dimensions $(K_\mathrm{x}, K_\mathrm{y}, K_\mathrm{z})$, a 3D cell occupancy map θ with identical dimensions is used to manage merging states. To initialise this object, the cells occupied by each input block with the same label are set to 1 (active). A default value of 0 is set for the remaining (inactive) cells to signify a different domain classification. To advance this discussion, it is helpful to define a pooling function,

$$\zeta_v(\mathbf{n},\mathbf{k}) = \sum_{d_\mathrm{z}=0}^{k_\mathrm{z}-1}\sum_{d_\mathrm{y}=0}^{k_\mathrm{y}-1}\sum_{d_\mathrm{x}=0}^{k_\mathrm{x}-1} \mathcal{I}(\theta(n_\mathrm{x}+d_\mathrm{x}, n_\mathrm{y}+d_\mathrm{y}, n_\mathrm{z}+d_\mathrm{z}) = v) \tag{D.1}$$

which counts the number of cells with label value v over a support interval that extends from $\mathbf{n} = (n_\mathrm{x}, n_\mathrm{y}, n_\mathrm{z}) \in \mathbf{Z}^3$ (the minimum cell coordinates) to $\mathbf{n}+\mathbf{k}-\mathbf{1} = (n_\mathrm{x}+k_\mathrm{x}-1, n_\mathrm{y}+k_\mathrm{y}-1, n_\mathrm{z}+k_\mathrm{z}-1)$ (the maximum cell coordinates) where $\mathbf{k}$ represents the provisional size of a block undergoing expansion. At any point during the coordinate-ascent algorithm, an incremental expansion $\boldsymbol{\delta} \in \mathbb{Z}^3$ — typically $\boldsymbol{\delta} \in \{(1,0,0), (0,1,0), (0,0,1)\}$ — is feasible if $\zeta_1(\mathbf{n}, \mathbf{k}+\boldsymbol{\delta}) = (k_\mathrm{x}+\delta_\mathrm{x})\cdot(k_\mathrm{y}+\delta_\mathrm{y})\cdot(k_\mathrm{z}+\delta_\mathrm{z})$ for a block with current cell dimensions $\mathbf{k}$.

Using this definition, the coordinate-ascent merging procedure from Section 2.3 as used in the SRUS (spatial restructuring using surfaces) framework is formally described in **Algorithm 1** on page 7.

Appendix D.5. Modifications

There are two key differences in the BlockMerge (BM) case. First, the boolean occupancy map $\theta \in \{0,1\}^{K_\mathrm{x}\times K_\mathrm{y}\times K_\mathrm{z}}$ now holds sub-block indices and becomes multi-valued, viz., $\theta \in \mathbb{Z}^{K_\mathrm{x}\times K_\mathrm{y}\times K_\mathrm{z}}$. Second, when a block expansion step is feasible in one of the coordinate directions, the increment takes on the dimension of the block (or blocks) along the axis of expansion; this being typically larger than 1. Merging states are managed using an ordered[2] list of structure similar to $\mathcal{M}$ in Algorithm 1, where each structure initially contains the minimum vertex of an input block $\mathbf{v}^{(b)}_{\mathrm{min}}$, its cell dimensions $\mathbf{s} = (s_\mathrm{x}, s_\mathrm{y}, s_\mathrm{z}) \in \mathbb{Z}^3$ which can grow, the block label $\lambda^{(b)}$ and a boolean flag, *subsumed*, which is set to false. The idea is to revise $\mathbf{s}$, the block dimensions expressed in terms of cells, as a block grows; blocks which have been swallowed are invalidated by setting *subsumed* to true and will be ignored in subsequent iterations. This effectively results in a shrinking set, the coalesced blocks are the surviving entries when the algorithm terminates. The algorithm continues as long as the cell count changes for any block between iterations. Details are given in **Algorithm 2** on page 8.

Appendix D.6. Cell expansion feasibility test

For block merging, Algorithm 2 has essentially the same blueprint as Algorithm 1. The main difference is the acceptance criteria for each expansion step, see *FeasibleCellExpansion* in lines 17, 24 and 31 in Algorithm 2. This is explained with the aid of Fig. D.4. When block merging is attempted, the expansion step proposes an elongation of the current block along one of the axes of expansion. The volumetric difference, before and after the proposed expansion, is referred as the *delta region*. Fig. D.4 further illustrates 5 situations where a merge with adjacent block(s) are infeasible. A proposed expansion step is feasible when two conditions are satisfied: 1) the dimension along the axis of expansion is the same for all adjoining blocks in the delta region; 2) the lateral dimensions of these adjoining blocks are compatible with the current block; in other words, their cross-sections must join perfectly. The computation inside *FeasibleCellExpansion* is described in **Subroutine 2**.

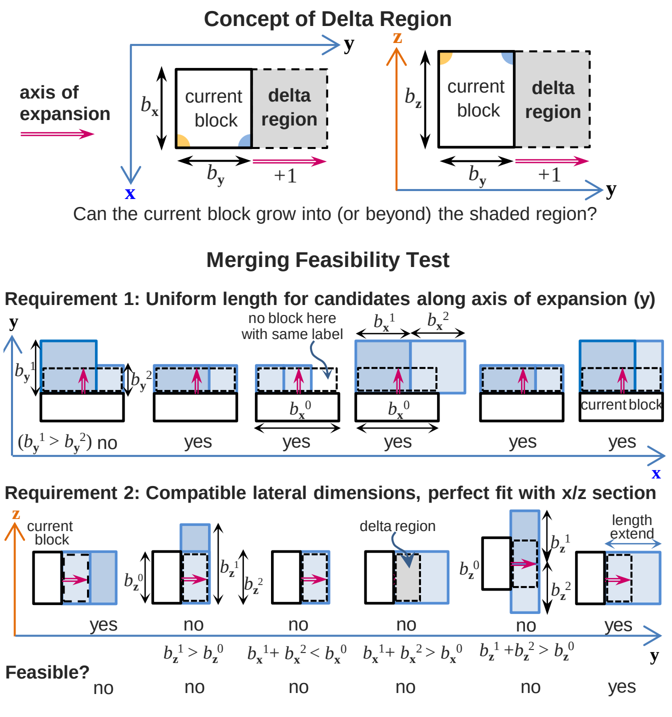

Concept of Delta Region
axis of expansion
current block
delta region
b_x
b_y
+1
b_z
x
y
z
Can the current block grow into (or beyond) the shaded region?
Merging Feasibility Test
Requirement 1: Uniform length for candidates along axis of expansion (y)
no block here with same label
b_y^1
b_y^2
b_x^0
b_x^1
b_x^2
current block
$(b_\mathrm{y}^1 > b_\mathrm{y}^2)$ no
yes
yes
yes
yes
yes
Requirement 2: Compatible lateral dimensions, perfect fit with x/z section
current block
delta region
length extend
b_z^0
b_z^1
b_z^2
yes
no
no
no
no
yes
$b_\mathrm{z}^1 > b_\mathrm{z}^0$
$b_\mathrm{x}^1 + b_\mathrm{x}^2 < b_\mathrm{x}^0$
$b_\mathrm{x}^1 + b_\mathrm{x}^2 > b_\mathrm{x}^0$
$b_\mathrm{z}^1 + b_\mathrm{z}^2 > b_\mathrm{z}^0$
Feasible?
no
no
no
no
no
yes

Figure D.4: Delta region and sub-block expansion feasibility tests

Appendix D.7. Merging conventions

In Algorithm 2, we have a block merging procedure that preserves the boundary of the input blocks, in the sense that it does not introduce new partitions (sub-divisions) that are not already present in a parent block. This is because when a sub-block is subsumed, it is swallowed whole by another block. This merging convention is referred as **persistent block memory** for future reference. A key property is that each input block is mapped uniquely to a single block in the merged model.

[2]Objects of type $\mathcal{M}$ are sorted in ascending order by the number of cells within each block, then by the minimum vertex coordinates to break ties. This priority gives smaller blocks the earliest opportunity to grow.

In contrast, Algorithm 1 implicitly erases sub-block boundaries before block consolidation begins. This merging convention is referred as **dissolve sub-block boundaries**, it generally achieves higher compaction because it makes no distinction between input blocks from the same class and parent. It is able to grow blocks more freely and produce fewer merged blocks since the size compatibility constraints between individual blocks no longer apply when they are treated as one. This can be useful for healing a fractured block model. It can consolidate sub-blocks introduced by a false boundary from a previous surface update. Under the "dissolve sub-block boundary" convention, coordinate-ascent can start from a clean slate. Sub-blocks in a fragmented area may grow back to the largest possible extent even if individual sub-block dimensions or internal boundary alignments are otherwise incompatible. It does not suffer the negative consequences of block structure decomposition from previous iterations. Some of these differences are shown in Fig. D.5.

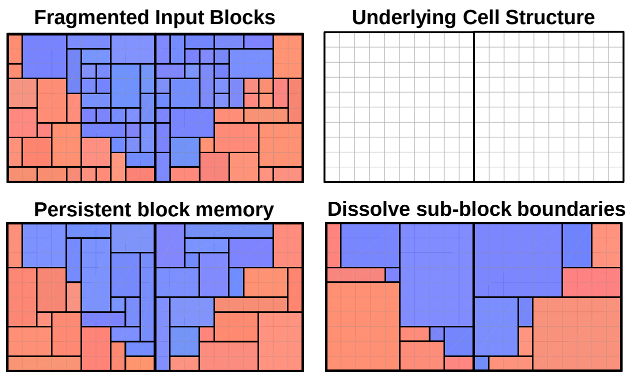


Figure D.5: Example of differences under the 'persistent block memory' and 'dissolve sub-block boundaries' block merging conventions

Appendix D.8. Fairness and regulating parameters

Both algorithms include optional parameters. The token life span, T, limits the number of uninterrupted sequential merging steps a block can take during coordinate-ascent, to moderate aggressive merging behaviour. This token value is decremented by 1 after each x-y-z cycle. When it reaches zero, the current block must cease expansion and give other blocks the opportunity to grow. When every block in the queue has had its turn, this block may resume expansion. The token value is reset to T each time a block takes possession. By default, T is set to infinity so no progress is ever halted. An upper bound on merged block dimensions is given by $(M_x, M_y, M_z) \in \mathbb{R}^3$. By default, this is set to the parent block size to remove any restriction.

Appendix D.9. Scan sequences to improve block aspect ratio

The final design consideration relates to the order in which input blocks are processed during coordinate-ascent. The main observation from Fig. D.6 is that depending on the shape and direction of the class boundary, a sequential algorithm may generate a stair-case artefact, producing long narrow blocks which certain applications may find objectionable. The incremental block expansion may be obstructed by the boundary if it approaches from a certain direction as it cycles through each coordinate axis; this can lead to excessive growth in an unimpeded direction. In general, no single deterministic scanning sequence (e.g., increasing x, increasing y and increasing z as in the "standard" case) can be optimal in all situations. One way to overcome this is by introducing multiple scan patterns. For instance, instead of scanning (processing blocks) top-down, left-to-right, one can scan from bottom-up, from right-to-left. This is equivalent to flipping the x and y axes.

Accordingly, there are 8 distinct possibilities given we have 3 axes, these scan sequences may be abbreviated as $\pi_0 = (+x, +y, +z)$, $\pi_1 = (-x, +y, +z)$, $\pi_2 = (+x, -y, +z)$ and so forth, where a negative sign indicates reversal of the relevant axis. The algorithm will try all 8 scan patterns and select the result which minimises an objective function. In this work, the preferred solution $\operatorname{argmin}_\pi f_\pi(\{\boldsymbol{\Delta}^{(b,\pi)}\}_{b\in\mathcal{S}_{\mathbf{p},\lambda}})$ minimises the volume-weighted *block aspect ratio*, the objective function may be expressed as

$$f_\pi(\{\boldsymbol{\Delta}^{(b,\pi)}\}_{b\in\mathcal{S}_{\mathbf{p},\lambda}}) = \frac{\sum_{b\in\mathcal{S}_{\mathbf{p},\lambda}} v^{(b,\pi)} \cdot \dfrac{\max\{\Delta_x^{(b,\pi)}, \Delta_y^{(b,\pi)}, \Delta_z^{(b,\pi)}\}}{\min\{\Delta_x^{(b,\pi)}, \Delta_y^{(b,\pi)}, \Delta_z^{(b,\pi)}\}}}{\sum_{b\in\mathcal{S}_{\mathbf{p},\lambda}} v^{(b,\pi)}} \quad \text{(D.2)}$$

where merged block b belongs to class λ in parent block $\mathbf{p}$, $(\Delta_x^{(b,\pi)}, \Delta_y^{(b,\pi)}, \Delta_z^{(b,\pi)}) \in \mathbb{R}^3$ and $v^{(b,\pi)}$ represent the dimensions and volume of the merged block, obtained from scan sequence π.

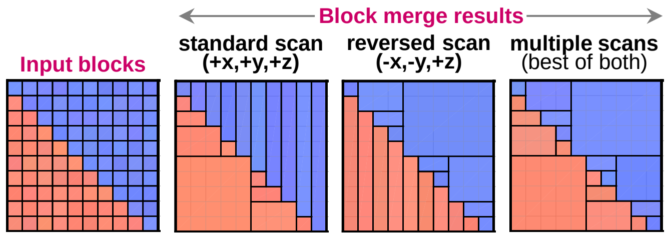


Figure D.6: Block merging results from different scan sequences

Appendix D.10. Scan sequence implementation

In practice, the eight individual scan patterns are not programmed explicitly. Instead, sub-blocks are rearranged within a parent block before coordinate-ascent, in such a way that a specific scan sequence is attained when the permuted data is subject to the standard scan. This is done to avoid code duplication and preserve the existing logic.[3]

The approach is explained in Fig. D.7. The key observation is that only the STANDARD scan is necessary (we do not need to implement 8 different scans directly) provided the cells occupied

[3]An explicit implementation for each scan pattern would involve 2^3 nested FOR loops, this includes the standard/existing scan pattern — for ($z = z_{\min}$; $z < z_{\max}$; z++) for ($y = y_{\min}$; $y < y_{\max}$; y++) for ($x = x_{\min}$; $x < x_{\max}$; x++) — and seven other combinations including, for instance, the (-x,-y,+z) scan pattern — for ($z = z_{\min}$; $z < z_{\max}$; z++) for ($y = y_{\max} - 1$; $y \geq y_{\min}$; y- -) for ($x = x_{\max} - 1$; $x \geq x_{\min}$; x- -). This is not easy to maintain.

by the input blocks are permuted to reflect a reversal of the relevant axes. For instance, a bottom–up, right–left scan sequence on the original block data may be implemented by mapping the white cells from the south-east corner to north-west corner (see Fig. D.7 (top)), then applying the "standard" top-down, left-right scan. The two are equivalent. Fig. D.7 (bottom) outlines the steps involved.

a) (**Forward permutation**) For each input block labelled *white* in parent block **p**, populate the occupancy map by sampling cells according to the direction of each axis specified in the scan instruction.[4]

b) (**Perform merging in rotated frame**) Apply coordinate-ascent merging algorithm to permuted data using the standard scan pattern.

c) (**Inverse permutation**) Register the location of merged blocks in the original frame using table-lookup.

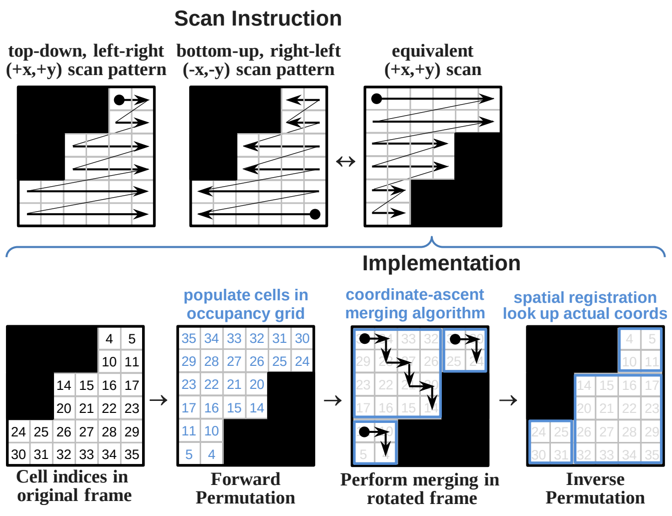

Scan Instruction
top-down, left-right (+x,+y) scan pattern
bottom-up, right-left (-x,-y) scan pattern
equivalent (+x,+y) scan
Implementation
populate cells in occupancy grid
coordinate-ascent merging algorithm
spatial registration look up actual coords
Cell indices in original frame
Forward Permutation
Perform merging in rotated frame
Inverse Permutation

Figure D.7: Block merging scan sequence implementation

Synthesizing all the ideas, **Algorithm 3** (page 10) describes the final block merging strategy which supports different merging conventions, multiple scan patterns and block aspect ratio optimisation. To elaborate on the the multi-threading aspect of the code, interleaved parent blocks are processed by individual threads within a region of interest. This choice, see interleaved parent indices in line 3 of Algorithm 3, is motivated by load balancing consideration. The intention is to spread the computation load evenly amongst the threads by decoupling spatial correlation, to avoid situations where too few (or too many) of the blocks processed by a thread actually intersect a surface.

[4]In reality, the occupancy map is a 3D array, but for simplicity, we only draw it in 2D.

Appendix E. Pseudocode

This pseudocode comprises the following:

Algorithm 1:. **Coordinate-ascent merging algorithm v1** (as used for spatial restructuring in Section 2.3 of (Leung, 2020))

Subroutine 1:. **Compute sub-block properties**

Algorithm 2:. **Coordinate-ascent merging algorithm v2** (as used for model de-fragmentation in Appendix D)

Subroutine 2:. **Feasibility tests and state updates during block expansion**

Algorithm 3:. **Block merging with multiple scans and optimised block aspect ratios**

Algorithm 1 Coordinate-ascent merging algorithm (as used in the spatial restructuring SRUS framework in Section 2.3)

```
Pre-requisite: Occupancy map, $\theta$, is populated s.t. all active
    cells that belong to sub-blocks of class $\lambda$ are set to 1.
Assumption: Cells in occupancy map are enumerated in
    raster-scan order, thus index $i(n_x, n_y, n_z) = (n_z K_y + n_y)K_x + n_x$. Parent block index is denoted $p$.
Input: $\theta \in \{0,1\}^{K_x \times K_y \times K_z}$
Parameters: Parent block cell dimensions: $K_x, K_y, K_z \in \mathbb{Z}$
            Min. block dimensions: $\mathbf{\Delta}^{\text{block}}_{\text{min}} \in \mathbb{R}^3$
            Max. merge cell dimensions: $M_x, M_y, M_z \in \mathbb{Z}$
            Token life span: $T \in \mathbb{Z}^+$
Variables: Active cells: $\mathbf{a} = []$ (initially an empty list)
           Merged blocks: $\mathcal{M} = \emptyset$ (initially an empty set)
           Stride length: $\mathbf{s} = (s_x, s_y, s_z) \in \mathbb{Z}^3$
           Provisional block dims: $\mathbf{d} = (d_x, d_y, d_z) \in \mathbb{Z}^3$
           Min. coordinates of current block: $\mathbf{v}^{(b)}_{\text{min}} \in \mathbb{R}^3$
           Obstacles count: barriers
           Iterations remaining: $i \in \mathbb{Z}$
1:  Find all active cells: $\mathbf{a} \leftarrow$ IndexOfOccupants($\theta$)
2:  Set count = 0, $n_{\text{occupant}} = |\mathbf{a}|$
3:  while number of active cells $|\mathbf{a}| \geq 1$ do
4:      Set $i = T$ and $s_x = s_y = s_z = 1$
5:      if $|\mathbf{a}| = 1$ then
6:          $\mathcal{M}$.append( SubBlockProperties($\mathbf{v}^{(b)}_{\text{min}}$, $\mathbf{s}$, $\mathbf{\Delta}^{(\text{block})}_{\text{min}}$, $\lambda$) )$^\dagger$
              NOTE † see description in Subroutine 1
7:          break
8:      end if
9:      Set $\mathbf{n} = (n_x, n_y, n_z)$ = Subscript( cell $\mathbf{a}[0]$ )
        where $n_x, n_y, n_z \geq 0$
10:     while true do
11:         barriers = 0
12:         $(d_x, d_y, d_z) \leftarrow (\min\{s_x + 1, K_x - n_x\}, s_y, s_z)$
13:         if $(d_x \leq M_x$ and $d_y \leq M_y$ and $d_z \leq M_z)$
            and $(d_x > s_x)$ and $\zeta_1(\mathbf{n}, \mathbf{d}) = d_x \cdot d_y \cdot d_z$ then
14:             $s_x = d_x$
15:         else
16:             barrier += 1
17:         end if
18:         $(d_x, d_y) = (\min\{s_x, K_x - n_x\}, \min\{s_y + 1, K_y - n_y\})$
19:         if $(d_x \leq M_x$ and $d_y \leq M_y$ and $d_z \leq M_z)$
            and $(d_y > s_y)$ and $\zeta_1(\mathbf{n}, \mathbf{d}) = d_x \cdot d_y \cdot d_z$ then
20:             $s_y = d_y$
21:         else
22:             barrier += 1
23:         end if
24:         $(d_y, d_z) = (\min\{s_y, K_y - n_y\}, \min\{s_z + 1, K_z - n_z\})$
25:         if $(d_x \leq M_x$ and $d_y \leq M_y$ and $d_z \leq M_z)$
            and $(d_z > s_z)$ and $\zeta_1(\mathbf{n}, \mathbf{d}) = d_x \cdot d_y \cdot d_z$ then
26:             $s_z = d_z$
27:         else
28:             barrier += 1
29:         end if
30:         $i$ −= 1
31:         if (count + $s_x s_y s_z = n_{\text{occupant}}$)
            or (barriers = 3) or ($i = 0$) then
32:             break (no further expansion is possible)
33:         end if
34:     end while
35:     Compute sub-block anchor point: $\mathbf{x}_{\text{min}} = \mathbf{v}^{(b)}_{\text{min}} + \mathbf{n} \circ \mathbf{\Delta}^{(\text{block})}_{\text{min}}$
36:     $\mathcal{M}$.append( SubBlockProperties($\mathbf{x}_{\text{min}}$, $\mathbf{s}$, $\mathbf{\Delta}^{(\text{block})}_{\text{min}}$, $\lambda$) )
37:     Update occupancy map: set $\theta[c_x, c_y, c_z]$ to 0 (inactive)
        for all cells bounded by $\mathbf{x}_{\text{min}}$ and $\mathbf{x}_{\text{max}} = \mathbf{x}_{\text{min}} + \mathbf{s}$.
38:     count += $s_x s_y s_z$
39:     Find remaining active cells: $\mathbf{a} \leftarrow$ IndexOfOccupants($\theta$)
40: end while
Output: consolidated sub-blocks $\mathcal{M}$
```

Subroutine 1 Compute sub-block properties

```
1: SubBlockProperties($\mathbf{v}^{(b)}_{\text{min}}$, $\mathbf{s}$, $\mathbf{\Delta}^{(\text{block})}_{\text{min}}$, $\lambda$)
2: Compute:
       sub-block dimensions: $\mathbf{\Delta}^{(b)}_{\text{sub-block}} = \mathbf{s} \circ \mathbf{\Delta}^{(\text{block})}_{\text{min}}$,
       sub-block max coordinates: $\mathbf{v}^{(b)}_{\text{max}} = \mathbf{v}^{(b)}_{\text{min}} + \mathbf{\Delta}^{(b)}_{\text{sub-block}}$
       sub-block centroid: $\mathbf{c}^{(b)}_{\text{sub-block}} = \frac{1}{2}(\mathbf{v}_{\text{min}} + \mathbf{v}^{(b)}_{\text{max}}) \in \mathbb{R}^3$
       sub-block label: $\lambda^{(b)} \leftarrow \lambda$
3: return $\langle \mathbf{c}^{(b)}_{\text{sub-block}}, \mathbf{\Delta}^{(b)}_{\text{sub-block}}, \lambda^{(b)} \rangle$
   note: $\circ$ denotes the Hadamard (element-wise) product.
```

Algorithm 2 Coordinate-ascent merging algorithm (as used in block model de-fragmentation in Appendix D.5)

Pre-requisites: The list of merged blocks $\mathcal{M}$ is initialised with one tuple $\langle \mathbf{v}_{\min}^{(b)}, \mathbf{s}^{(b)}, \lambda^{(b)}, n_{\text{cells}}^{\text{prev}(b)}, n_{\text{cells}}^{\text{curr}(b)}, \textit{subsumed}^{(b)} = 0\rangle$ for each sub-block b in class λ within the parent block, where $\mathbf{v}_{\min}^{(b)} \in \mathbb{R}^3$, $\mathbf{s}^{(b)} \in \mathbb{Z}^3$, $n_{\text{cells}}^{\text{prev}(b)}$ and $n_{\text{cells}}^{\text{curr}(b)}$ denote the sub-block minimum vertex, sub-block cell-dimensions, number of cells in the previous and current iteration, respectively. The occupancy map θ is populated such that each active cell is assigned the relevant sub-block index, viz., b; all remaining cells are set to -1 (inactive).
Input: $\mathcal{M}$ (with all $n_{\text{cells}}^{\text{prev}(b)}$ set to 0) and $\theta \in \mathbb{Z}^{K_x \times K_y \times K_z}$
Parameters: same as Algorithm 1
Variables: Active sub-blocks: $\mathbf{a}$ = [] (initially an empty list)
Otherwise, similar to Algorithm 1

```
1:  do
2:     Sort list of block properties, $\mathcal{M}$, by cell count,
       then minimum vertex, in ascending order.
3:     Find all active sub-blocks:
       $\mathbf{a} \leftarrow$ FindAllActiveSubBlocks($\mathcal{M}$) where subsumed=0
4:     if $|\mathbf{a}| = 1$ then
5:        break
6:     end if
7:     for each $b$ in ordered sub-blocks $\mathbf{a}$ do
8:        Set $n_{\text{cells}}^{\text{prev}(b)} = n_{\text{cells}}^{\text{curr}(b)}$
9:        if subsumed$^{(b)}$ then
10:          continue
11:       end if
12:       Set $i = T$ and $(s_x, s_y, s_z) = \left(s_x^{(b)}, s_y^{(b)}, s_z^{(b)}\right)$
13:       Set $\mathbf{n} = (n_x, n_y, n_z) =$ Subscript(lowest cell in block $b$)
14:       while true do
15:          barriers = 0
16:          $(d_x, d_y, d_z) \leftarrow (\min\{s_x + 1, K_x - n_x\}, s_y, s_z)$
17:          if $(d_x > s_x)$ and FeasibleCellExpansion($\theta, \mathcal{M}, b$ |
18:          $(n_x+s_x, n_y, n_z), (n_x+d_x, n_y+s_y, n_z+s_z)$, "x") then
19:             $s_x = s_x^{(b)\star}$
20:          else
21:             barrier += 1
22:          end if
23:          $(d_x, d_y) = (\min\{s_x, K_x - n_x\}, \min\{s_y + 1, K_y - n_y\})$
24:          if $(d_y > s_y)$ and FeasibleCellExpansion($\theta, \mathcal{M}, b$ |
25:          $(n_x, n_y+s_y, n_z), (n_x+s_x, n_y+d_y, n_z+s_z)$, "y") then
26:             $s_y = s_y^{(b)\star}$
27:          else
28:             barrier += 1
29:          end if
30:          $(d_y, d_z) = (\min\{s_y, K_y - n_y\}, \min\{s_z + 1, K_z - n_z\})$
31:          if $(d_z > s_z)$ and FeasibleCellExpansion($\theta, \mathcal{M}, b$ |
32:          $(n_x, n_y, n_z+s_z), (n_x+s_x, n_y+s_y, n_z+d_z)$, "z") then
33:             $s_z = s_z^{(b)\star}$
34:          else
35:             barrier += 1
36:          end if
37:          $i$ −= 1
38:          if $(s_x = K_x - n_x$ and $s_y = K_y - n_y$ and $s_z = K_z - n_z)$
39:          or (barriers = 3) or $(i = 0)$ then
40:             break (no further expansion is possible)
41:          end if
42:       end while
43:    end for
44: while $n_{\text{cells}}^{\text{curr}(b)} \neq n_{\text{cells}}^{\text{prev}(b)}$ for any block in $\mathcal{M}$
45: Remove all subsumed sub-blocks from $\mathcal{M}$
```

Output: consolidated sub-blocks $\mathcal{M}$
NOTE *: The properties of $\mathcal{M}^{(b)}$ are updated implicitly by *FeasibleCellExpansion* when the expansion is feasible. Details are given in Subroutine 2.

Subroutine 2 Feasibility tests and state updates during block expansion

```
Parameters: Parent block cell dimensions: $(K_x, K_y, K_z) \in \mathbb{Z}^3$
            Current block cell dimensions: $(s_x, s_y, s_z) \in \mathbb{Z}^3$
            Max. merge cell dimensions: $(M_x, M_y, M_z) \in \mathbb{Z}^3$
Mutable objects: Occupancy map: $\theta \in \mathbb{Z}^{K_x \times K_y \times K_z}$
                 List of block properties: $\mathcal{M}$
Notations: ◦ Delta region: $\mathcal{R}$
           ◦ Length along axis of expansion for sub-blocks
             found in the delta region: $l_{b' \in \mathcal{R}}(direction)$
           ◦ Number of cells from sub-blocks found in the
             delta region: $n_{\mathcal{R}}^{(\text{cells})}$
           ◦ Unique set of sub-blocks in delta region: $\mathcal{S}$
1: FeasibleCellExpansion($\theta$, $\mathcal{M}$, $b$ |
       $(n_x^0, n_y^0, n_z^0)$, $(n_x^1, n_y^1, n_z^1)$, direction)
2: if $n_x^0 \geq K_x$ or $n_y^0 \geq K_y$ or $n_z^0 \geq K_z$ then
3:     return false
4: end if
5: for each cell $(c_x, c_y, c_z)$ in $\mathcal{R}$ do
6:     Let sub-block index $b' = \theta(c_x, c_y, c_z)$
7:     if $b' \neq -1$ then
8:         $\mathcal{S}$.insert( $b'$ )
       else $\mathcal{R}$ contains at least one foreign cell
9:         return false
10:    end if
11: end for
12: if $l_{b' \in \mathcal{R}}(direction)$ is identical for all blocks in $\mathcal{R}$ then
13:     Set $n_{\text{extend}} = l_{b' \in \mathcal{R}}(direction) \in \mathbb{Z}^+$
14: else
15:     return false (failed uniform length requirement)
16: end if
17: Let $\tilde{\mathcal{S}} = \{b' \in \mathcal{S} \mid subsumed^{(b')} = \text{false}\} \subseteq \mathcal{S}$
18: if direction is “x” then
19:     if $(s_x + n_{\text{extend}}, s_y, s_z)$ exceeds $(M_x, M_y, M_z)$ then
20:         return false
21:     else
22:         Compute $n_{\mathcal{R}}^{(\text{cells})}$ from blocks $b' \in \tilde{\mathcal{S}}$
23:         Set compatible = $(n_{\mathcal{R}}^{(\text{cells})} = n_{\text{extend}} \cdot s_y \cdot s_z)$? true : false
24:     end if
25: else if direction is “y” then
26:     if $(s_x, s_y + n_{\text{extend}}, s_z)$ exceeds $(M_x, M_y, M_z)$ then
27:         return false
28:     else
29:         Compute $n_{\mathcal{R}}^{(\text{cells})}$ from blocks $b' \in \tilde{\mathcal{S}}$
30:         Set compatible = $(n_{\mathcal{R}}^{(\text{cells})} = s_x \cdot n_{\text{extend}} \cdot s_z)$? true : false
31:     end if
32: else
33:     if $(s_x, s_y, s_z + n_{\text{extend}})$ exceeds $(M_x, M_y, M_z)$ then
34:         return false
35:     else
36:         Compute $n_{\mathcal{R}}^{(\text{cells})}$ from blocks $b' \in \tilde{\mathcal{S}}$
37:         Set compatible = $(n_{\mathcal{R}}^{(\text{cells})} = s_x \cdot s_y \cdot n_{\text{extend}})$? true : false
38:     end if
39: end if
40: if compatible then
41:     Update block properties list $\mathcal{M}$
42:     Set $subsumed^{(\beta)}$ = true $\forall \beta \in \tilde{\mathcal{S}}$
43:     Set $n_{\text{cells}}^{\text{prev}(b)} = n_{\text{cells}}^{\text{curr}(b)}$
44:     Set $n_{\text{cells}}^{\text{curr}(b)} \mathrel{+}= \sum_{\beta \in \tilde{\mathcal{S}}} n_{\text{cells}}^{\text{curr}(\beta)}$
45:     Update occupancy map $\theta$
46:     Set $\theta(c_x, c_y, c_z) = b$ for all cells in blocks $\beta \in \tilde{\mathcal{S}}$.
47: end if
48: return compatible
```

Algorithm 3 Block merging with multiple scans and optimised block aspect ratios

```
1:  parallel for thread $t$ from 0 to $n_{\text{thread}}-1$ do
2:    Set $coalesced\_blocks^{(t)} = \emptyset$
3:    for parent block $p$ in $\{t + i \cdot n_{\text{thread}}\}_{i\in\mathbb{Z}}$ and $p < n^{(\text{block})}_{\text{parent}}$ do
4:      Find input blocks $\mathcal{B}_p$ contained in $p$
5:      for each class $\lambda$ within $p$ do
6:        Find blocks $\mathcal{B}_{p,\lambda}$ with label $\lambda$
7:        Let the cost for current best solution $f_* = \infty$
8:        if convention is DissolveSubBlockBoundaries then
9:          for each scan pattern $\pi$ do
10:           Populate occupancy map $\theta \in \{0,1\}^{K_x\times K_y\times K_z}$ s.t.
              active cells in $\mathcal{B}_{p,\lambda}$ are set to 1; 0 otherwise.
11:           Invoke coordinate-ascent (Algorithm 1) to
              obtain the consolidated blocks $\mathcal{M}_\pi$
12:           Compute the cost $f(\mathcal{M}_\pi)^\dagger$
13:           if $f_* > f(\mathcal{M}_\pi)$ then
14:             Set $f_* = f(\mathcal{M}_\pi)$ and $\mathcal{M}_* = \mathcal{M}_\pi$
15:           end if
16:         end for
17:         $coalesced\_blocks^{(t)}$.append( $\mathcal{M}_*$ )
18:       else if convention is PersistentBlockMemory then
19:         for each scan pattern $\pi$ do
20:           Populate occupancy map $\theta \in \mathbb{Z}^{K_x\times K_y\times K_z}$ s.t. all
              active cells in $\mathcal{B}_{p,\lambda}$ are set to the relevant sub-
              block index $b \in \mathcal{B}_{p,\lambda}$; -1 otherwise.
21:           Invoke coordinate-ascent (Algorithm 2) to
              obtain the consolidated blocks $\mathcal{M}_\pi$
22:           Compute the cost $f(\mathcal{M}_\pi)^\dagger$
23:           if $f_* > f(\mathcal{M}_\pi)$ then
24:             Set $f_* = f(\mathcal{M}_\pi)$ and $\mathcal{M}_* = \mathcal{M}_\pi$
25:           end if
26:         end for
27:         $coalesced\_blocks^{(t)}$.append( $\mathcal{M}_*$ )
28:       end if
29:     end for
30:   end for
31:   Signal when thread $t$ completes its task
32: end parallel for
33: Aggregate results: $solution \leftarrow \{coalesced\_blocks^{(t)}\}$
Output: solution
  NOTE:$^\dagger$ using the objective function based on
  volume-weighted block aspect ratio, for instance.
```

Appendix F. Octree decomposition and merging

The two octree schemes considered in the paper are the standard octree decomposition, and octree with intra-scale merging. Starting at full resolution ($d = 0$), at each level of the spatial hierarchy, a rectangular block with dimensions $\mathbf{\Delta}^{(d)} \in \mathbb{R}^3$ may be split into eight sub-blocks (or cells) called an octant, where the dimensions of each sub-block are essentially halved along each axis, yielding $\mathbf{\Delta}^{(d+1)} = \frac{1}{2}\mathbf{\Delta}^{(d)}$. A split is performed when 2 or more of its sub-blocks at resolution $\mathbf{\Delta}^{(d+1)}$ carry different labels. This decomposition is performed recursively and stops only when the 3D block region becomes homogeneous (all 8 cells have the same label) or when the maximum decomposition level D is reached. Such hierarchical structures are well studied in the literature, see (Samet, 1988) and (Tamminen and Samet, 1984) for instance. The purpose of this section is to clarify what intra-scale block merging means in this work, and how it relates to the standard octree.

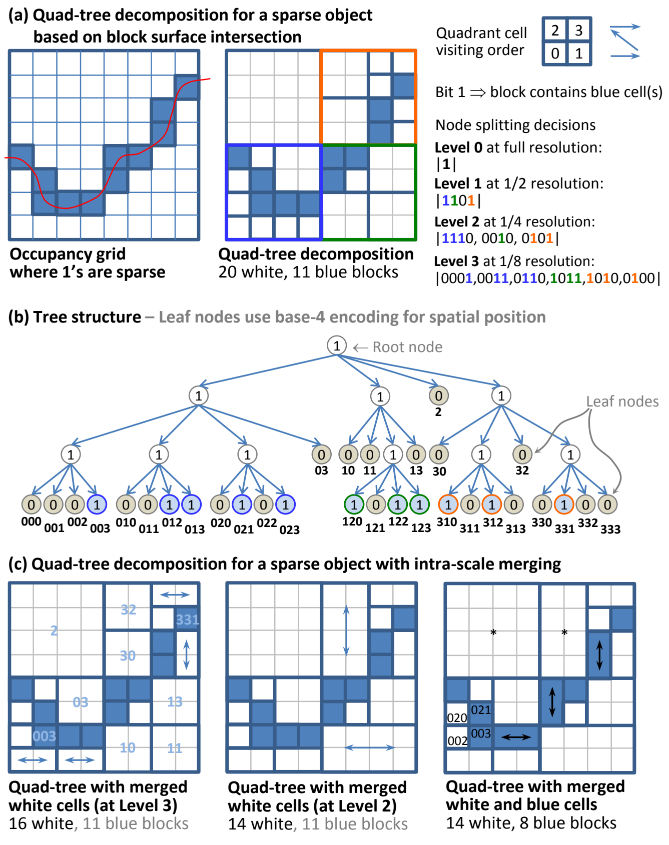

(a) Quad-tree decomposition for a sparse object based on block surface intersection
Quadrant cell visiting order
2 3
0 1
Bit 1 ⇒ block contains blue cell(s)
Node splitting decisions
Level 0 at full resolution: |1|
Level 1 at 1/2 resolution: |1101|
Level 2 at 1/4 resolution: |1110, 0010, 0101|
Level 3 at 1/8 resolution: |0001,0011,0110,1011,1010,0100|
Occupancy grid where 1's are sparse
Quad-tree decomposition 20 white, 11 blue blocks
(b) Tree structure – Leaf nodes use base-4 encoding for spatial position
← Root node
Leaf nodes
000 001 002 003 010 011 012 013 020 021 022 023 03 10 11 120 121 122 123 13 2 30 310 311 312 313 32 330 331 332 333
(c) Quad-tree decomposition for a sparse object with intra-scale merging
Quad-tree with merged white cells (at Level 3) 16 white, 11 blue blocks
Quad-tree with merged white cells (at Level 2) 14 white, 11 blue blocks
Quad-tree with merged white and blue cells 14 white, 8 blue blocks

Figure F.8: Octree for encoding sparse object such as edges

Octree decomposition is a popular technique for encoding sparse data such as edge pixels in an image array. Fig. F.8 provides an example whereby block surface intersections are localised by blue cells in (a-left). A complete quad-tree decomposition of this region into sub-blocks at $\frac{1}{2}$, $\frac{1}{4}$ and $\frac{1}{8}$ scale is shown in (a-middle). Following a particular quadrant cell scanning order, the 2D pattern may by represented by the tree-structure in (b). Intra-scale block merging has the specific meaning described in (c) where coalesced blocks are limited to adjacent cells within a quadrant; the arrows in (c-left) and (c-middle) show this happening at two spatial scales, $d = 3$ and $d = 2$. The final result after octree decomposition and intra-scale merging is shown in (c-right). This picture illustrates that further merging is in fact possible — for instance between the white cells 002 and 020, or blue cells 003 and 021 — if inter-scale merging is permitted. We opted not to challenge these rules for the octree approach since these merging opportunities have already been exploited by the proposed methods, and intra-scale merging has its place in our performance comparison. For simplicity, the region is treated as a 2D block, however all aspects generalise to three-dimensions (from quadrant to octant) and all processes involved in the actual experiments operate in 3D.

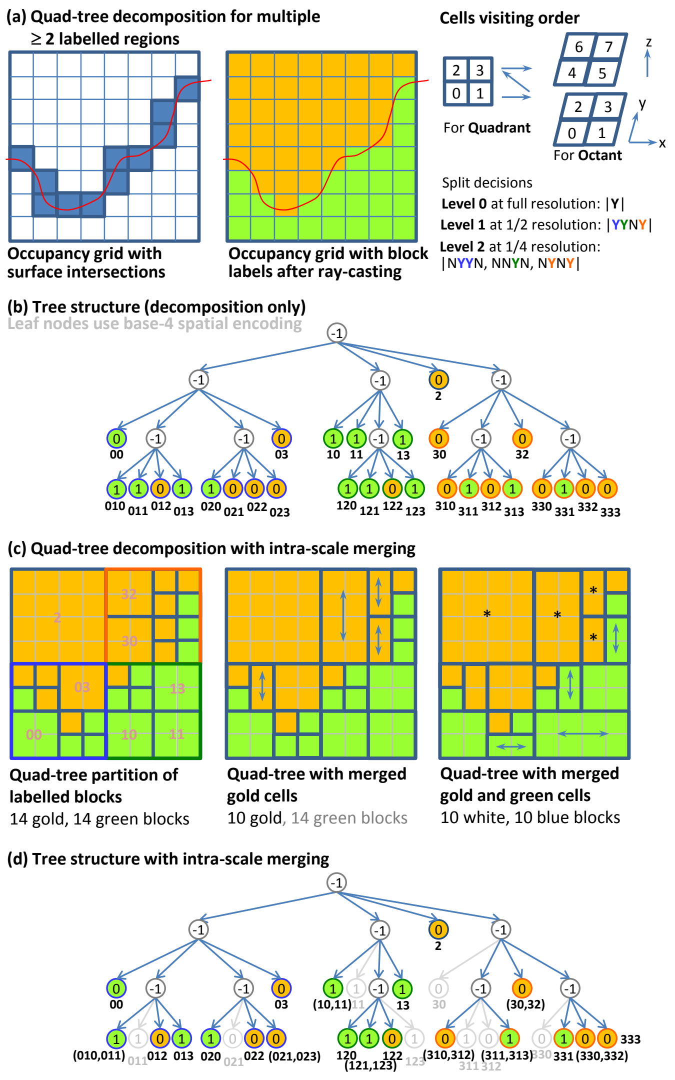

(a) Quad-tree decomposition for multiple ≥ 2 labelled regions
Occupancy grid with surface intersections
Occupancy grid with block labels after ray-casting
Cells visiting order
For Quadrant
For Octant
Split decisions
Level 0 at full resolution: |Y|
Level 1 at 1/2 resolution: |YYNY|
Level 2 at 1/4 resolution: |NYYN, NNYN, NYNY|
(b) Tree structure (decomposition only)
Leaf nodes use base-4 spatial encoding
(c) Quad-tree decomposition with intra-scale merging
Quad-tree partition of labelled blocks 14 gold, 14 green blocks
Quad-tree with merged gold cells 10 gold, 14 green blocks
Quad-tree with merged gold and green cells 10 white, 10 blue blocks
(d) Tree structure with intra-scale merging

Figure F.9: Octree decomposition and intra-scale merging for multiple regions

Extending these ideas to encode non-sparse regions, we observe that standard octree decomposition works in a top-down manner and has no innate ability for labelling cells at the minimum block size. Therefore, ray-tracing is used (since it forms part of the block model spatial restructuring workflow) to label cells as 0 or 1; colouring cells in gold or green in Fig. F.9(a-middle) depending upon which side of the surface they are on.

Applying octree decomposition produces the tree-structure shown in Fig. F.9(b) where split nodes are labelled -1, leaf nodes are labelled 0 or 1 (when there are two regions) and coloured gold or green accordingly. Result obtained with further intra-scale merging is shown in (c). As before, arrows indicate the blocks which have been merged within a quadrant at a given scale. The resultant tree-structure after intra-scale merging is depicted in (d). Comparing with (b), branches connecting with blocks which have been subsumed are evidently pruned with the corresponding nodes removed. Our earlier remarks on further inter-scale merging opportunities also exist here, for instance, blocks marked with asterisk in (c-right) can all potentially be combined into a single block. Although we focused our attention on two regions in this example, all relevant aspects generalise to three or more regions when multiple surfaces are involved.

Appendix F.1. Major difference between quadtree and octree

For an octree, the major difference with respect to quadtree are the candidates considered during intra-scale merging. Following the octant cell scanning order shown in Fig. F.9 (a-right), prospective 2-cell merge candidates include basically 12 edges: viz., $\{(0,1),(0,2),(1,3),(2,3)\}$ and $\{(4,5),(4,6),(5,7),(6,7)\}$ from the top and bottom sides, and similarly $\{(2,6),(3,7)\} \cup \{(0,4),(1,5)\}$ from the north and south sides of an octant. Prospective 4-cell merge candidates include 6 square faces: {(0,1,2,3), (4,5,6,7), (0,1,4,5), (2,3,6,7), (0,2,4,6), (1,3,5,7)}.

Appendix G. Detailed octree subblocking comparison

This section provides a more detailed breakdown of the model block count results presented in Sec. 8.1 of (Leung, 2020). Henceforth, we use the word ‘Octree’ to denote standard octree decomposition. When the ‘+Merge’ suffix is added, intra-scale merging is attempted between compatible cells within each octant. This means, edge-connected cells within the same octant may be combined in groups of two or four to form a rectangular or squared block as described above (in Appendix F.1). However, inter-scale merging across different decomposition levels is not permitted. ‘Proposed-P’ refers to the proposed block merging algorithm performed under the *persistent* block memory convention. ‘Proposed-D’ refers to the same under the *dissolved* subblock boundary convention. ‘Domain’ means geological domain and ‘% volume’ means percentage of the total volume in the modelled region.

To promote spatial awareness, an animated sequence of the test site’s domain structure is shown layer-by-layer in Table G.1 where the domain colour palette matches the colour labels used in the tables. A geology background is not required to understand this data. However, domain labels annotated by M, N and H may be interpreted as ‘mineralised’, ‘non-mineralised’ and ‘hydrated’ domains, respectively, by geologists.

References

Akenine-Möller, T., 2001. Fast 3D triangle-box overlap testing. Journal of graphics tools 6, 29–33.

Dietrich, A., Gobbetti, E., Yoon, S.E., 2007. Massive-model rendering techniques: a tutorial. IEEE Computer Graphics and Applications 27, 1–18.

Leung, R., 2020. Modelling orebody structures: Block merging algorithms and block model spatial restructuring strategies given mesh surfaces of geological boundaries. Journal of Spatial Information Science .

McGuire, M., 2017. Computer Graphics Archive (https://casual-effects.com/data/).

Möller, T., Trumbore, B., 2005. Fast, minimum storage ray/triangle intersection, in: ACM SIGGRAPH 2005 Courses, ACM. p. 7.

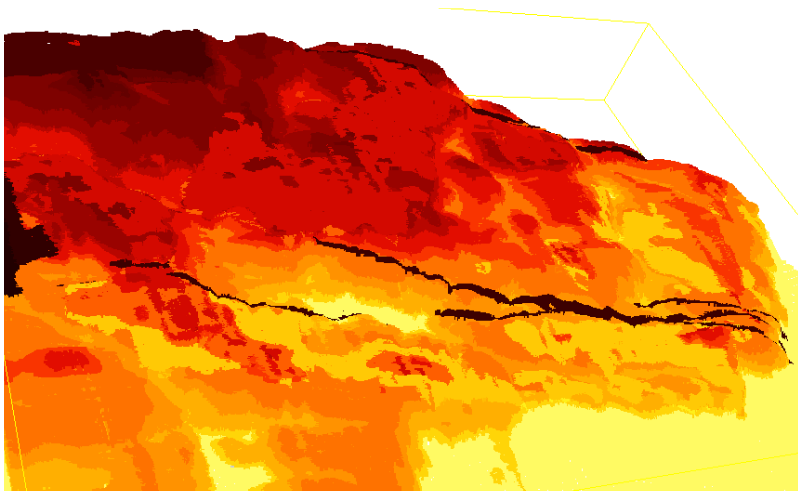
Samet, H., 1988. An overview of quadtrees, octrees, and related hierarchical data structures, in: Theoretical Foundations of Computer Graphics and CAD. Springer, pp. 51–68.

Tamminen, M., Samet, H., 1984. Efficient octree conversion by connectivity labeling. ACM SIGGRAPH Computer Graphics 18, 43–51.

Turk, G., Levoy, M., 2014. The Stanford 3D Scanning Repository. URL: `http://www.graphics.stanford.edu/data/3Dscanrep/`.

Table G.1: Block model statistics: proposed methodology vs octree (with D=3 decomposition levels)

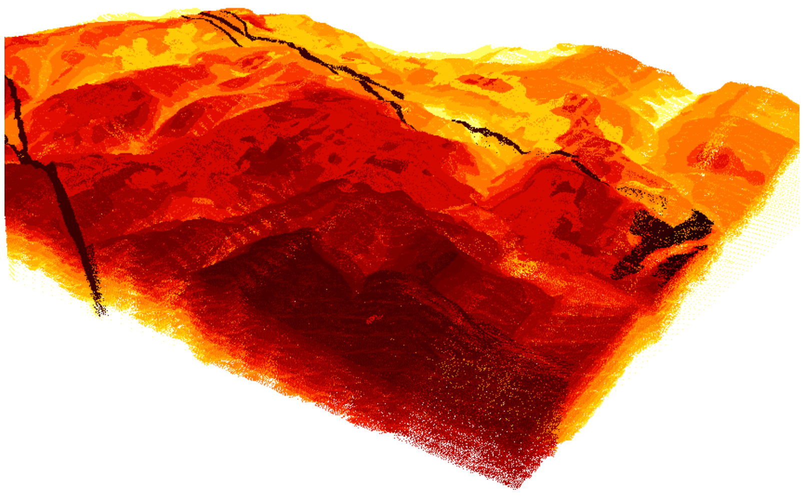
Domain	% volume	block count				volume-weighted block aspect ratio			
		Octree	Octree + Merge	Proposed-P	Proposed-D	Octree	Octree + Merge	Proposed-P	Proposed-D
⬛ N_0	0.012340	2644	1154	772	704	2.5	3.954896	9.251789	3.685130
🟫 M_0	0.012216	2422	1057	791	594	2.5	3.923701	6.925476	3.811883
🟫 N_1	2.071587	326463	123708	59908	52925	2.5	2.435113	3.252310	2.386576
🟫 N_2	0.571025	27523	11878	7256	7679	2.5	3.301445	4.297926	2.631313
🟫 M_1	0.000045	17	12	12	12	2.5	3.529412	3.088235	3.088235
🟥 N_3	0.247183	27769	12558	8132	8728	2.5	3.954645	6.093452	2.532378
🟥 N_4	0.318811	34279	15510	10080	10732	2.5	3.954323	6.095554	2.586614
🟥 N_5	1.036601	80587	35968	23032	23655	2.5	3.552698	5.208733	2.565268
🟥 M_2	0.074106	13096	5958	3710	3788	2.5	3.723641	5.582627	3.214119
🟥 N_6	1.996944	158170	70237	44619	45183	2.5	3.610081	5.044647	2.649020
🟥 M_3	0.426214	53367	24250	15604	15490	2.5	3.638962	5.590088	2.862343
🟥 N_7	1.058833	166835	75399	46713	47876	2.5	3.871672	5.621888	2.996686
🟥 M_4	0.112265	23454	11036	7113	7121	2.5	3.780942	5.277800	3.288215
🟥 H_0	0.332034	64151	29953	21128	19535	2.5	3.784879	7.178617	3.074582
🟥 N_8	2.363637	241316	106972	67694	69211	2.5	3.942829	5.903363	2.878910
🟥 M_5	0.035386	8595	4190	2824	2828	2.5	3.720205	6.229485	3.344340
🟥 N_9	1.500652	249267	111807	67270	68843	2.5	3.928390	5.649683	3.162396
🟧 M_6	0.005508	1579	811	576	568	2.5	3.527364	4.199520	3.340855
🟧 H_1	0.052937	12489	6035	4357	4132	2.5	3.818116	6.895195	3.289241
🟧 N_{10}	5.062708	394979	173877	108587	110817	2.5	3.756453	5.372227	2.753414
🟧 M_7	0.230237	31048	13958	8972	8687	2.5	3.753732	4.868516	2.861059
🟧 N_{11}	4.327004	450788	198725	123289	126389	2.5	3.966091	5.891937	2.966619
🟧 M_8	0.119425	21038	9726	6387	6238	2.5	3.602132	5.310738	3.053729
🟧 N_{12}	6.059302	517338	225301	138735	141283	2.5	3.893502	5.591457	2.858506
🟧 M_9	0.092165	15999	7252	4671	4601	2.5	3.480721	5.185372	2.839088
🟨 H_2	0.193627	42416	20254	14320	13380	2.5	3.691417	6.955868	3.105813
🟨 N_{13}	3.049206	474133	208258	121712	124319	2.5	3.935326	5.502560	3.186988
🟨 M_{10}	0.005196	1692	822	563	526	2.5	3.379135	5.267176	3.388906
🟨 N_{14}	0.001007	339	180	138	133	2.5	3.366142	5.218110	3.547769
🟨 N_{15}	68.631674	797448	323952	175932	174585	2.5	2.599801	2.867128	2.506040
🟨 M_{11}	0.000127	48	28	20	23	2.5	3.645833	5.781250	3.564583
Total (avg. by volume)		4241289	1830826	1094917	1100585	2.5	2.958828	3.668188	2.615373
Total (avg. by block count)		same	same	same	same	2.5	3.535403	5.068056	2.839873
Ratio		100.000	43.167	25.816	25.949				



Animated sequence — birds eye view of Site 8's spatial structure
Geological domains are peeled back layer by layer in this animation

Table G.2: Block model statistics: proposed methodology vs octree (with D=4 decomposition levels)

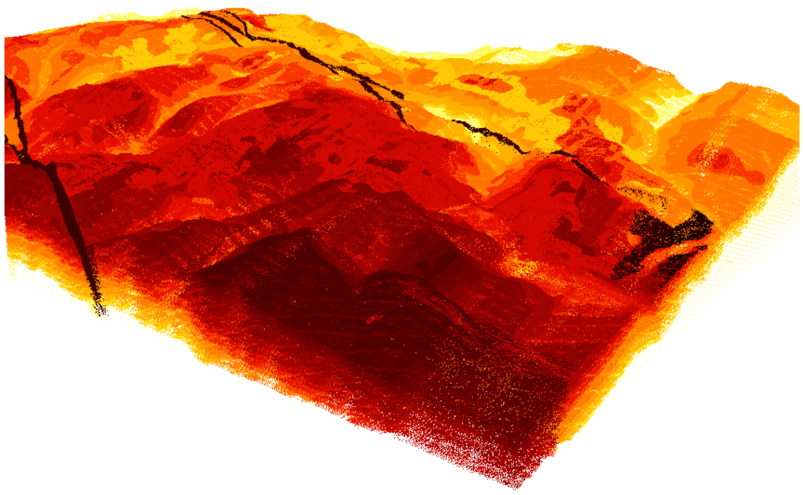
Domain	% volume	block count				volume-weighted block aspect ratio			
		Octree	Octree + Merge	Proposed-P	Proposed-D	Octree	Octree + Merge	Proposed-P	Proposed-D
■ N_0	0.012367	11742	5065	3317	2746	2.5	3.900329	15.549850	4.059046
■ M_0	0.012224	10946	4650	3420	2201	2.5	3.909655	10.466585	4.674493
■ N_1	2.071568	1429469	533237	223538	185677	2.5	2.438087	4.852193	3.363062
■ N_2	0.571042	108496	47062	26560	28328	2.5	3.330618	6.133705	2.987043
■ M_1	0.000039	112	62	45	44	2.5	3.592437	4.118487	3.170588
■ N_3	0.247097	116150	52202	31105	33811	2.5	3.882805	9.738552	3.590038
■ N_4	0.318835	143282	64352	38541	41093	2.5	3.928394	9.994956	3.620897
■ N_5	1.036702	342263	150890	87578	89680	2.5	3.623427	8.200493	3.193486
■ M_2	0.074196	60521	26754	14949	15198	2.5	3.768124	8.326254	3.820477
■ N_6	1.996801	667620	292632	169023	170337	2.5	3.644469	7.759336	3.312802
■ M_3	0.426296	235590	104610	60747	59749	2.5	3.610870	8.889136	3.776128
■ N_7	1.058725	728816	324163	182289	186255	2.5	3.892632	8.553657	3.970692
■ M_4	0.112501	110475	49990	28969	28582	2.5	3.779309	7.580900	4.003741
■ H_0	0.331797	298282	135474	88804	78141	2.5	3.779701	11.899441	3.762755
■ N_8	2.362666	1022988	450290	259668	261922	2.5	3.936937	9.418536	3.851701
■ M_5	0.035805	45500	21242	12706	12578	2.5	3.677538	9.116177	3.699823
■ N_9	1.501456	1098197	485024	264369	268390	2.5	3.963615	8.493589	4.080947
■ M_6	0.005730	9258	4734	2985	2929	2.5	3.564548	5.640211	3.770489
■ H_1	0.053013	62734	29214	19290	17460	2.5	3.701885	10.991420	3.706879
■ N_{10}	5.064220	1663577	727924	410459	414897	2.5	3.794050	8.463884	3.543067
■ M_7	0.228323	142484	61726	35310	33610	2.5	3.726269	7.352394	3.805164
■ N_{11}	4.325235	1895420	828568	466319	473527	2.5	3.942704	9.354241	4.004534
■ M_8	0.119550	99113	44273	25985	24900	2.5	3.579750	7.963333	3.817275
■ N_{12}	6.059267	2143315	928956	519128	522118	2.5	3.921219	8.812658	3.766239
■ M_9	0.092176	73872	32446	18975	18255	2.5	3.478816	8.269290	3.744605
■ H_2	0.193962	201251	92622	60682	54519	2.5	3.722921	11.130710	3.710936
■ N_{13}	3.050039	2038137	885891	469694	475684	2.5	3.970251	8.126027	4.180359
■ M_{10}	0.005201	9332	4259	2611	2405	2.5	3.427114	7.226263	3.727820
■ N_{14}	0.001046	2086	1004	647	610	2.5	3.463970	7.412168	3.983059
■ N_{15}	68.632008	2643443	1065825	525508	501270	2.5	2.605537	3.283658	2.576202
■ M_{11}	0.000115	284	143	89	102	2.5	3.832853	8.900865	2.621326
Total (avg. by volume)		17414755	7455284	4053310	4007018	2.5	2.968178	4.901418	2.941797
Total (avg. by block count)		same	same	same	same	2.5	3.585874	7.861600	3.650470
Ratio		100.000	42.810	23.275	23.009				



Animated sequence of Site 8's spatial structure
X cross-sections of the same geological domains

Table G.3: Block model statistics: proposed methodology vs octree (with D=5 decomposition levels)

Domain	% volume	block count				volume-weighted block aspect ratio			
		Octree	Octree + Merge	Proposed-P	Proposed-D	Octree	Octree + Merge	Proposed-P	Proposed-D
⬛ N_0	0.012187	54650	22512	13500	10598	2.5	3.937759	28.207846	5.272790
🟫 M_0	0.012225	47666	19995	14373	8581	2.5	3.880556	17.203926	6.303496
🟫 N_1	2.069887	5966122	2206483	856985	691068	2.5	2.448336	8.103377	5.186617
🟫 N_2	0.569810	465501	196686	101427	106403	2.5	3.341298	9.875999	3.403828
🟫 M_1	0.000040	646	333	225	211	2.5	3.638946	7.322417	2.796074
🟫 N_3	0.246775	482523	214397	120786	128908	2.5	3.872228	17.465091	5.425162
🟫 N_4	0.318412	593948	264225	149991	157914	2.5	3.893686	18.004315	5.075332
🟥 N_5	1.035730	1441073	626411	340497	345133	2.5	3.648167	14.088016	3.945783
🟥 M_2	0.074129	266189	114990	59476	59692	2.5	3.725392	14.391746	5.381985
🟥 N_6	1.994655	2809366	1215883	656664	653464	2.5	3.665340	13.196620	4.206202
🟥 M_3	0.426116	994488	437691	238458	232800	2.5	3.634994	15.740771	5.344710
🟥 N_7	1.057534	3076723	1352508	715135	728884	2.5	3.909520	14.793386	6.200759
🟥 M_4	0.112500	484807	215842	116435	114125	2.5	3.801134	12.542677	5.895113
🟥 H_0	0.331752	1295410	578825	360946	308160	2.5	3.779019	21.440046	5.554469
🟥 N_8	2.361292	4261395	1856015	1009360	997440	2.5	3.932490	16.538731	5.283464
🟥 M_5	0.035589	209842	94582	51685	50865	2.5	3.681174	15.741827	5.199280
🟥 N_9	1.500113	4615285	2017890	1041881	1049804	2.5	3.979482	14.656347	6.338921
🟧 M_6	0.005824	53198	25599	14448	14215	2.5	3.544744	8.629379	5.305402
🟧 H_1	0.052644	302851	133967	79652	70437	2.5	3.725562	19.455017	5.425445
🟧 N_{10}	5.059155	6903632	2989708	1582574	1573708	2.5	3.815540	14.647774	4.527278
🟧 M_7	0.229129	615542	261940	139458	129839	2.5	3.719217	12.449978	5.553382
🟧 N_{11}	4.322323	7841613	3398938	1800920	1801115	2.5	3.935243	16.360033	5.554397
🟧 M_8	0.119576	434074	190030	104029	98436	2.5	3.584765	13.754793	5.630620
🟨 N_{12}	6.055483	8847559	3800130	1995917	1975291	2.5	3.923066	15.285411	4.941555
🟨 M_9	0.092180	322413	139347	76104	71667	2.5	3.464583	14.724576	5.738100
🟨 H_2	0.193918	892552	402913	249416	217807	2.5	3.716055	19.805877	5.544980
🟨 N_{13}	3.048503	8456955	3650182	1837766	1853527	2.5	3.977186	13.698514	6.438980
🟨 M_{10}	0.005207	45786	20074	11478	10381	2.5	3.470138	11.670511	4.750513
🟨 N_{14}	0.001024	10736	4770	2806	2567	2.5	3.561028	13.049818	4.983007
🟨 N_{15}	68.656172	12271192	4599110	1865318	1741347	2.5	2.608728	4.109259	2.667184
🟨 M_{11}	0.000117	1564	760	452	457	2.5	3.635483	12.809443	2.827696
Total (avg. by volume)		74065301	31052736	15608162	15204844	2.5	2.972378	7.395642	3.456381
Total (avg. by block count)		same	same	same	same	2.5	3.585541	13.523983	5.078064
Ratio		100.000	41.926	21.074	20.529				



Animated sequence of Site 8's spatial structure
Y cross-sections of the same geological domains