4-2011

Design and Implementation of an Extensible Variable Resolution Bathymetric Estimator

Brian R. Calder
University of New Hampshire, Durham, brian.calder@unh.edu

Glen Rice
NOAA

Recommended Citation

B. R. Calder and G. Rice, "Design and implementation of an Extensible variable resolution Bathymetric Estimator," in PROC. U.S. HYDRO. CONF, 2011, pp. 25–28. [Online]. Available: http://ushydro.thsoa.org/hy11/0428P_01.pdf.

Design and Implementation of an Extensible Variable Resolution Bathymetric Estimator

B. R. Calder, LT(JG) G. Rice, NOAA

***Abstract*—For grid-based bathymetric estimation techniques, determining the right resolution at which to work is essential. Appropriate grid resolution can be related, roughly, to data density and thence to sonar characteristics, survey methodology, and depth. It is therefore variable in almost all survey scenarios, and methods of addressing this problem can have enormous impact on the correctness and efficiency of computational schemes of this kind.**

This paper describes the design and implementation of a bathymetric depth estimation algorithm that attempts to address this problem by combining the computational efficiency of locally regular grids with piecewise-variable estimation resolution to provide a single logical data structure and associated algorithms that can adjust to local data conditions, change resolution where required to best support the data, and operate over essentially arbitrarily large areas as a single unit. The algorithm, which is in part a development of CUBE, is modular and extensible, and is structured as a client-server application to support different implementation modalities. The algorithm is called "CUBE with Hierarchical Resolution Techniques", or CHRT.

***Index Terms*—Bathymetric Estimation, CUBE, CHRT, Variable Resolution, Client-Server Algorithm, Data Density Estimation, Data-driven Estimation**

I. Introduction

Rapid processing of Multibeam Echosounder (MBES) bathymetric data for hydrographic applications has been a popular research topic for over two decades. A number of different approaches have been proposed for the problem, from methods that apply simple pointwise tests to data [1] to methods that attempt to simulate the behavior of a human data processor [2], methods that apply statistical tests to collections of data points in an attempt to identify outliers [3]–[6], methods that attempt to form Triangulated Irregular Networks from data points directly [7], [8], and methods that attempt to compute best-estimated depths within a given region [9], [10], among many others.

Although comparisons between algorithms are rare, algorithms such as the Combined Uncertainty and Bathymetry Estimator (CUBE) [9] that estimate depths in a (most often regular) grid can have a number of advantages over methods based on selecting which measurements should be preserved in the output, which has to date been the most prevalent paradigm for processing algorithms (mainly because it better matches manual methods for data processing driven by a traditional desire to use individual measurements on a chart product). Grid-based methods using regular grids can be very computationally efficient, can be constructed to generate products that retain the most likely estimate of depth constructed from all available relevant data rather than the self-noise of the individual measurements, and automatically generate products that are easy to store, visualize, turn into other products and distribute in various forms. They have also traditionally had a key limitation: an essentially arbitrary choice of the resolution at which depth estimates are computed and retained.

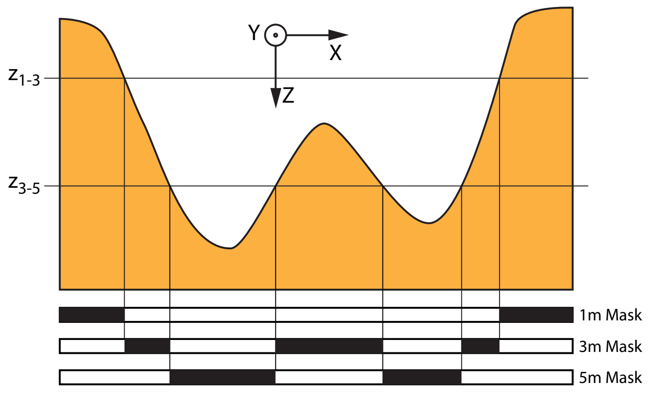
The *correct* choice of resolution is not, however, arbitrary. The seafloor in any particular area has some, generally unknown, level of detail in the topography, and a grid-based representation must be capable of achieving the Nyquist sampling rate [11, §1.4] for the area if an appropriate representation of the seafloor is to be preserved. This is modified in the case of a MBES system by the physical limitations of beamwidth, ping repetition rate, pulse-length and sampling rate built into the system so that the grid-based representation only has to meet the sampling rate of the surface *as observed* by the MBES, but this is still a complex function of seafloor character, MBES system in use, system parameters and, particularly, depth. To compensate for areas with extreme dynamic depth range and rapid, unpredictable changes in depth, therefore, a grid-based estimation method must provide some means to change resolution spatially and, preferably, in a manner that adapts to the data observed.

A trivial example of such a scheme is to split the depth range being considered into bands of constant resolution based on some estimate of depth and MBES abilities, and conduct the estimation at fixed resolution in all areas that fall within the appropriate band, Fig. 1. In this scheme, which was used in the research code-base for CUBE, each band is represented by a single resolution grid, and for robustness it is important that only one resolution is used in each physical location. Although simple, this can lead to difficulties in data management as multiple grids have to be coordinated at one time and then combined in some manner to provide the final output surface estimate for any given location. As a design goal, the data structure should be transparent to the user. The scheme can also result in large areas that are processed at either too high or too low a resolution due to the generally limited number of resolution bands that are considered.

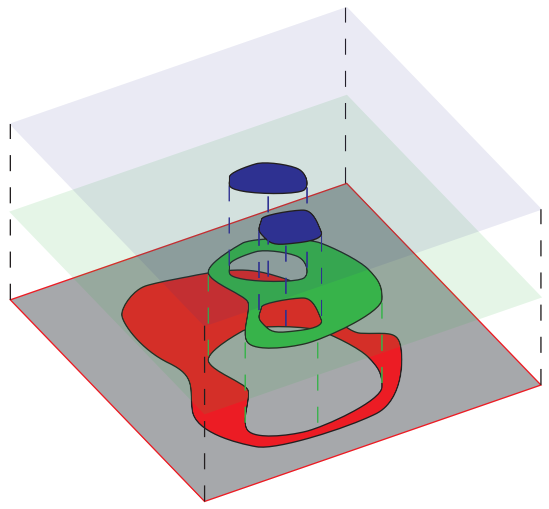
A common alternative is to utilize a quadtree approach [12], [13], where the grid cells or estimation nodes are spaced in dyadic multiples of a base resolution (i.e., $\Delta x = 2^n \Delta x_0,\ 0 \leq n < \infty$ for some arbitrary base resolution Δx_0), with density of nodes being determined by recursively sub-dividing the area into four equal sections until some measure of consistency

B. Calder is with the Center for Coastal and Ocean Mapping and NOAA-UNH Joint Hydrographic Center, University of New Hampshire. Address: Chase Ocean Engineering Lab, 24 Colovos Road, Durham NH 03824. E-Mail: brc@ccom.unh.edu. Phone: +1-603-862-0526.

G. Rice is with NOAA's Office of Coast Survey, Coast Survey Development Lab. Address: Chase Ocean Engineering Lab, 24 Colovos Road, Durham NH 03824. E-Mail: glen.rice@noaa.gov. Phone: +1-603-862-1397



(a) Depth transition points and resolution band masks.



(b) Composing resolution bands for the final surface representation.

Fig. 1. Illustration of the resolution-band approach to multi-resolution estimation. Based on an *a priori* estimate of depth, areas above depth z_{1-3} are estimated at 1m resolution; areas with depths between z_{1-3} and z_{3-5} are estimated at 3m resolution; and areas below z_{3-5} are estimated at 5m resolution. Only areas where the r-m resolution mask are black are estimated at r-m resolution, ensuring that only one resolution is used in each area.

within each section is satisfied. While this provides for a single data structure that allows changes in estimation resolution, it also, like the resolution band scheme, induces a structure in the data that is not necessarily supported by the data itself. That is, there is no reason why the data should be constrained to any specific resolution level due to the algorithm; induced structure usually reflects a limiting design decision.

More sophisticated methods, such as constrained irregular quadrilateral grids [14], [15] or constrained unstructured (triangulated) grids [16], [17] are also possible, and are used extensively in finite element computational methods, computer graphics, etc. While these more sophisticated methods certainly offer more nuanced descriptions of the required estimation resolution, and can be made to adapt to some metric derived from the problem being attempted, their subtlety in representation is bought at the expense of more complex implementations and higher data storage and computational cost per data point processed, particularly for the sorts of computations required for the types of depth estimation algorithms that we consider here. Given data volumes in the billions of individual measurements per survey, cost per measurement must always be a major driving factor in algorithm and data structure selection.

Consideration of these alternative schemes suggest a set of design goals for an algorithm to process data from raw measurements to a finite sampled representation. First, we require rapid, robust estimation of depth from raw data which takes into account typical failure modes for MBES systems. Second, we need some means to estimate the appropriate resolution at which to process the data that is driven by the data itself so that we adapt to the data, rather than inducing structure that is not reflected in the dataset. This resolution should correspond to the maximum resolution at which the data can be reliably processed while balancing robustness of estimation, fundamental resolution limitations of the data collection system, and fidelity of representation of the seafloor. Third, we need the data structure to support varying resolutions, and to be transparent to the user's requirements for area covered. Fourth, we require that the implementation be efficient and flexible in order to support potential differences in computational structures that might arise in practice.

We therefore propose here a scheme for achieving a piecewise-variable resolution representation of data within an essentially arbitrary large area (limited only by disc space available for storage), which adapts its resolution to local data conditions based on a preliminary estimate of data density whilst retaining the computational efficiency of a locally-regular grid representation and the statistical robustness of the CUBE algorithm, on which it is based. Although not necessarily as flexible as more general schemes, we believe that the algorithm will achieve the majority of the objectives of such schemes, whilst being significantly simpler and therefore more efficient. The algorithm is complemented by a memory management and spatial referencing scheme that supports efficient persistent handling of the data structures involved in the estimation, and is implemented in a modular fashion so that components of the algorithm can be replaced transparently.

This paper describes the design of the piecewise-variable resolution scheme, the estimation methods used to compute the resolution at which to work, and the workflow patterns for this algorithm in different data capture and availability scenarios. We also describe the implementation of the algorithm, and in particular a client-server structure that is intended to decouple the user interface and computational engine of the algorithm allowing more flexible implementations. Finally, the degree to which research-based code can, and should, be used for commercial-grade implementations of this type of algorithm is, we believe, still an open question, and we offer some perspectives on this based on our experience with prior algorithms.

II. Algorithm Design

A. Variable Resolution with Locally Regular Grids

We consider first the problem of efficiently varying resolution within some generic area. We establish first a coarse grid over the area of interest, with resolution R-m chosen such that

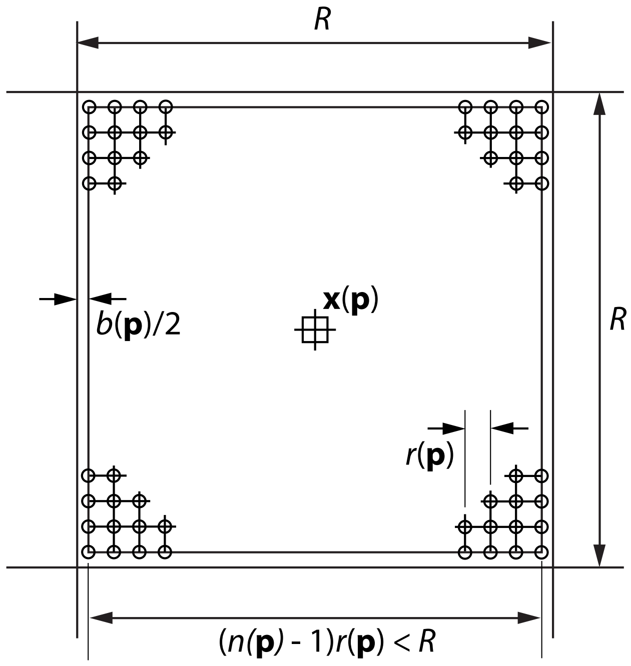


Fig. 2. Geometry of the hierarchical super grid cell refinement used in CHRT. The refined grid is constrained to be entirely within the cell, with maximum area not covered less than the estimate node spacing of the refined grid.

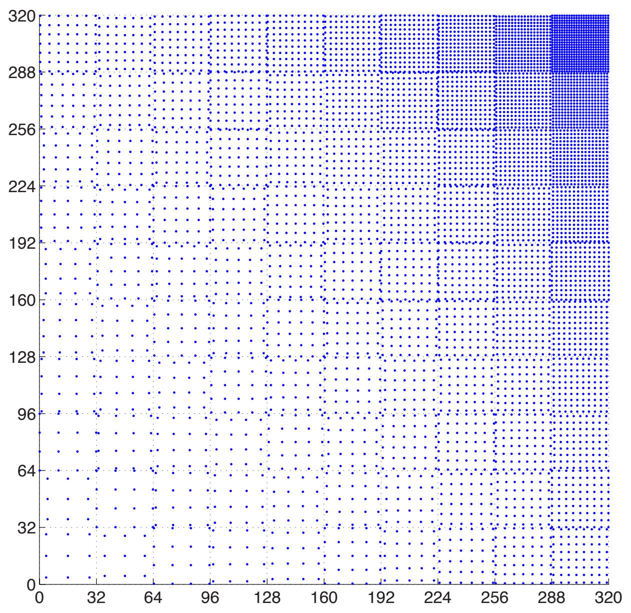


Fig. 3. An example of the estimation node locations within the super grid, using resolution estimates constructed from synthetic data. Note that the density of estimation nodes is always at least the designed density, and typically higher on the edges between the super grid cells. (Axes are arbitrary projected units; cell resolution is $R = 32$ units.)

$R \geq r_{\min}$, the coarsest resolution that we expect to use at any point in the area; we refer to this as the "super grid". (We assume that the cells are square in some suitable but arbitrary projected coordinate system.) Assume for the time being that, given a first pass over the data, we can estimate an appropriate resolution, $r(\mathbf{p})$, $\mathbf{p} = [u, v]' \in \mathbb{Z}^2$, for each cell in the super grid which we believe will be supported by the data. Within each cell, which covers area $\mathsf{A}(\mathbf{p}) \subset \mathbb{R}^2$, we can then construct another regular grid $\mathsf{G}(\mathbf{p})$ with resolution $r(\mathbf{p})$ such that the grid has

$$n(\mathbf{p}) = \lfloor R/r(\mathbf{p}) \rfloor + 1 \tag{1}$$

nodes on a side, and is centered at

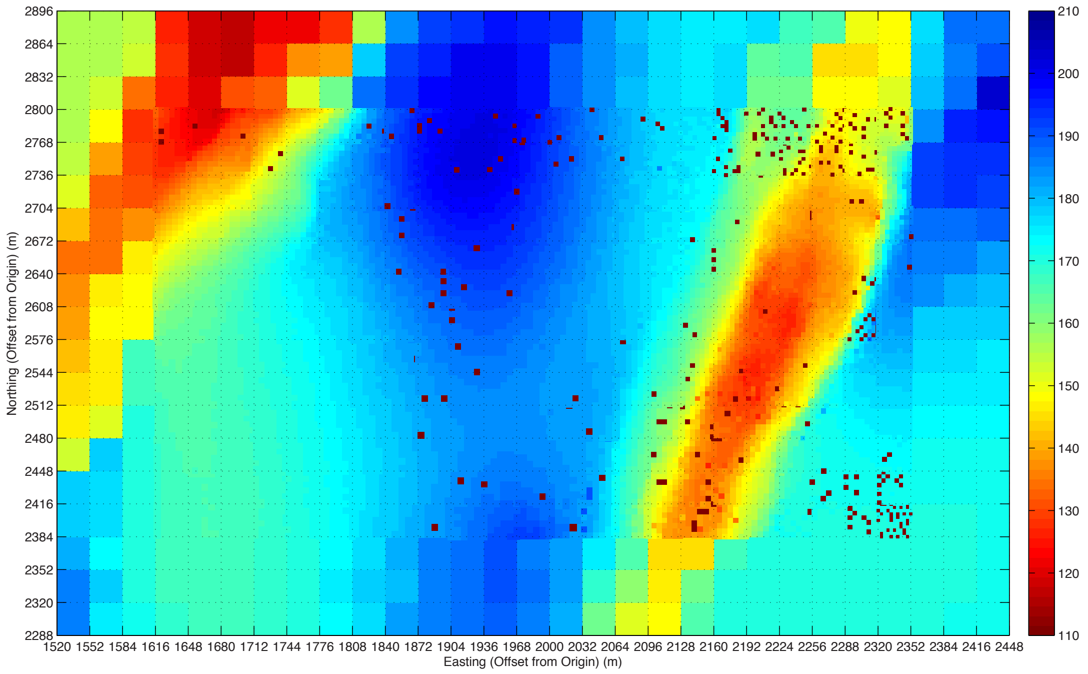
$$\mathbf{x}(\mathbf{p}) = (\mathbf{p} + 1/2)R \tag{2}$$

so that we are guaranteed that the coverage of the refined grid, $\mathsf{A}^r(\mathbf{p})$ is a proper subset of the cell, $\mathsf{A}^r(\mathbf{p}) \subset \mathsf{A}(\mathbf{p})$, Fig. 2, since $(n(\mathbf{p}) - 1)r(\mathbf{p}) \leq R$ by construction. We also have that $b(\mathbf{p}) = R - (n(\mathbf{p}) - 1)r(\mathbf{p}) < r(\mathbf{p})$ so that the spacing between outer estimation nodes in adjacent cells is at most $b(\mathbf{p}) < r(\mathbf{p})$ and consequently there are at least enough estimation nodes to maintain the designed estimation resolution, with typically slightly higher density of estimation nodes in the boundaries between cells. Fig. 3 shows an example of this in the case of synthetic resolution estimates, where the variation in resolution within an area is extreme enough to illustrate the change, while Fig. 4 shows the algorithm applied to some real data.

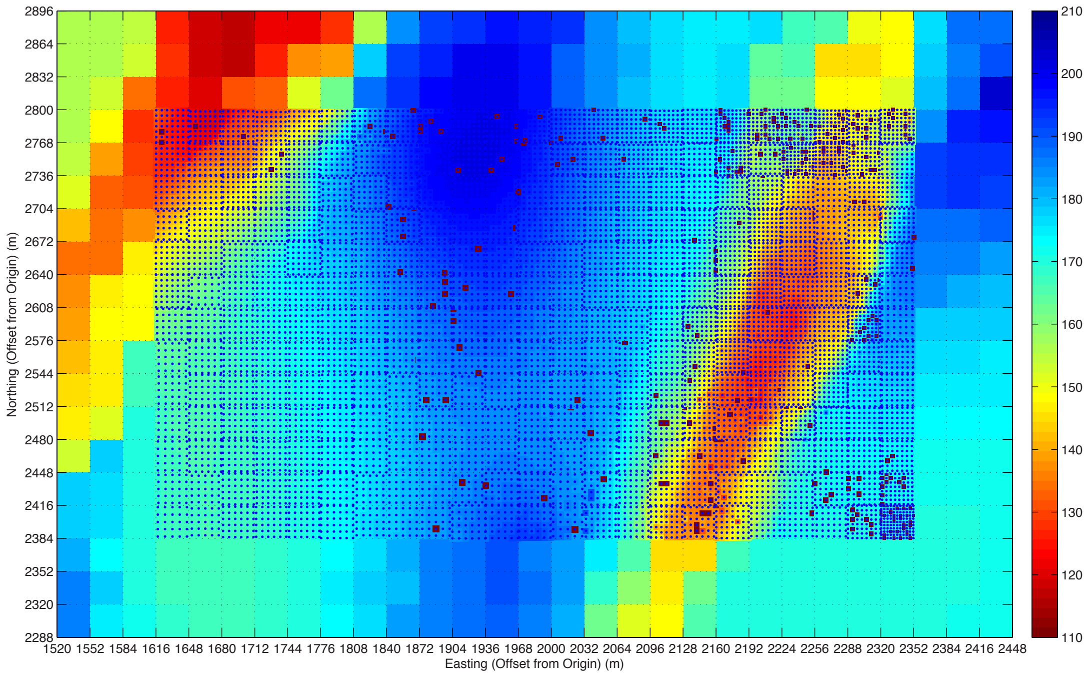
By hierarchically nesting regular grids, this scheme allows for piecewise-variable resolution estimation with the resolution tied to that estimated from the data. It also preserves, however, the use of locally-regular grids, since the estimation of stable resolution occurs on a regular grid of resolution R-m, and the estimation of depth occurs on a piecewise-regular grid $\mathsf{G} = \bigcup_{\mathbf{p} \in \mathbb{Z}^2} \mathsf{G}(\mathbf{p})$ of disjoint components (i.e., $\mathsf{A}^r(\mathbf{p}) \cap \mathsf{A}^r(\mathbf{q}) = \emptyset$, $\mathbf{p} \neq \mathbf{q}$), so only regular grid estimation techniques are required as long as the refined grids are treated individually.

The choice of resolution for the super grid cells is to some extent arbitrary, although there are some constraints. The coarsest possible resolution that can be sustained by the grid is R-m, and therefore we must ensure that, given the maximum depth expected in the area and the sonar being used, we do not expect to exceed this. At the other extreme, increasing R gives more individual measurements over which to compute the refinement resolution, but also means that we cannot change resolution as quickly, since the adaptation rate is limited to the cell size. If we allow the cells to become too large, therefore, we risk inducing structure in the data representation that is not supported by the data since the seafloor can change more rapidly than the algorithm can adapt. This may result in some proportion of the area being estimated at either too high or too low a resolution. (The former can result in noisy estimates or holes in the representation; the latter in poor fidelity of representation.) If the cells are not large enough, however, the estimation of resolution may become unstable. There are also questions of implementation efficiency with small cells.

The optimal strategy for choosing the cell resolution may be subject to a number of competing requirements for any particular dataset. We expect, however, that for most cases the driver for faster change in resolution will naturally push the cell size towards the minimum constraint. This also has the significant benefit that it is more likely that the data density within the cell will be more closely a constant so that a regular grid is a better approximation everywhere in the cell



(a) Super grid background depth estimates with overlaid refinement resolution estimates at variable resolutions within the super grid cells. Small red dots are missing estimates due to lower than expected data density in the local area.



(b) As above, with overlaid points at the refined scale estimation node locations.

Fig. 4. Example of the refinement algorithm applied to standard hydrographic data (in this case a NOAA hydrographic survey courtesy of the NOAA Ship FAIRWEATHER). The super grid cells are here set to 32 m width, and refinement was computed based on the analysis of (5) [section II-B] with $n_{\text{req}} = 5$, $\epsilon = 0.05$ and $r_{\max} = 0.25$ m. The resolutions predicted vary from ≈ 3 m to ≈ 8 m in the ≈ 100 m dynamic range in the area. The variation is mostly driven by depth, but is modulated by density of data due to swath placement, etc.

to the optimal resolution at which to estimate. In areas where this was not the case, it is possible that we might require a more flexible refinement, using, e.g., a quadtree locally where required. The disjoint nature of the refined grids naturally leads to an encapsulated implementation, so there is no technical limitation on having different refinement strategies on a cell-by-cell basis if required. This would allow us to preserve efficiency by using a regular grid where the extra flexibility is not required.

B. Resolution Estimation

Although many of the techniques described here could be used for general depth estimation algorithms or even general gridding algorithms, we are interested particularly in application of the CUBE [9] algorithm both because we have access to the algorithm from previous work, and because of its efficiency and robustness with respect to MBES data. Although the algorithm itself attempts to estimate depths at a point, and therefore does not require any particular organization of estimation nodes, all current implementations use a regular grid of nodes to organize the computation. Addition of CUBE to the piecewise-variable resolution scheme outlined previously is therefore straightforward.

CUBE's basic assumption is that the goal of bathymetric data estimation is to determine as well as possible the depth at a given point. Since the positions of the individual measurements from the MBES are effectively random with respect to any given estimation node, however, a key component of the algorithm is to determine which individual measurements around a given node can be used to inform the depth estimation method; since we typically use data in a closed ball of radius c-m about the node location $\mathbf{x}$, $B_c(\mathbf{x})$, this distance is usually called the "capture radius". In previous versions of the algorithm the capture radius was essentially a free parameter, although it was typically set as $c = \max\{c_{\text{min}}, s_c z\}$ where c_{min} is a minimum radius to avoid the estimation collapsing in very shallow water, z is the expected depth in the area (usually approximated by the declared depth of each individual measurement for lack of further information) and s_c is a fixed constant, typically in the range $[0.01, 0.10)$.

We have previously considered the problem of predicting a resolution for CUBE nodes [18] where we suggested that the resolution be set in terms of the data density observed in the data, $\rho(\mathbf{p})\,\text{m}^{-2}$, so that a node will receive an average of $\pi c^2 \rho(\mathbf{p})$ observations. After some manipulations we show that this implies that we require

$$r(\mathbf{p}) \geq \gamma \sqrt{\frac{n_{\text{req}}(1+\epsilon)}{\pi \rho(\mathbf{p})}} \tag{3}$$

based on the assumption that we need n_{req} individual measurements for stable estimation, lose a proportion of the measurements, ϵ, to noise, and have a ratio $\gamma \triangleq r(\mathbf{p})/c$ of node spacing to capture radius. Concerns for our ability to represent an object with basic size L-m lead us to require the upper limit

$$r(\mathbf{p}) \leq \kappa L,\ 0 < \kappa \leq \frac{1}{2} \tag{4}$$

from the Nyquist theorem. The only remaining variable in the resolution computation is the ratio γ, which was previously considered independent. So that we do not miss any measurements between the $B_c(\mathbf{x})$, we arrange for $c = \min\{\delta : \mathsf{A}^r(\mathbf{p}) \subset \bigcup_{\mathbf{n} \in \mathsf{G}(\mathbf{p})} B_\delta(\mathbf{x_n})\}$ by setting $\gamma = \sqrt{2}$. ($\mathsf{G}(\mathbf{p})$ is the set of all nodes in the refined grid at $\mathbf{p}$.) We therefore have the requirement that

$$\kappa L \geq r(\mathbf{p}) \geq \sqrt{\frac{2n_{\text{req}}(1+\epsilon)}{\pi \rho(\mathbf{p})}},\ 0 < \kappa \leq \frac{1}{2} \tag{5}$$

and therefore all we require to compute the resolution, given suitable (user defined) values for n_{req} and ϵ is an estimate of the data density within each super grid cell, $\rho(\mathbf{p})$. Note that we may set the capture radius relative to the resolution in this case only because the resolution is computed from the data, rather than being independently chosen.

Under most circumstances, we expect refinements to occur at the finest resolution allowed by (5). It is possible, however, that in some cases resolutions selected from a user-specified set, or limited to a dyadic scale may be required. This is simply a matter of mapping the resolution appropriately after estimation, for example setting

$$r'(\mathbf{p}) = 2^{\lfloor \log_2(r(\mathbf{p})) + s_m \rfloor} \tag{6}$$

for dyadic levels, or constructing a lookup-table for a given user-specified set. There are, however, draw-backs to this approach, particularly that it again induces a structure not found in the data, Fig. 5, and that it can lead to some cells or parts of cells estimated at coarser or finer resolution than is appropriate. This can be partially controlled by appropriate choice of s_m in the mapping equation, which can be used to shift the transition points between resolution levels up to one full level, Fig. 6, although the flexibility of this scheme is limited in practice.

Linking the capture radius to the resolution allows for simplifications to the standard CUBE algorithm. In particular, since we know that the capture radius of any one node does not extend past any other node, an individual measurement can only be used by at most the four-nearest neighbors of its nominal location, reducing the computation for distribution of measurements to the nodes. In addition, any measurements farther than $\max_{\mathbf{q} \in \mathsf{N}(\mathbf{p})} r(\mathbf{q})/\sqrt{2}$ from the super grid cell boundary cannot be used by any node in an adjacent cell (here $\mathsf{N}(\mathbf{p})$ is the set of eight-nearest neighbor cells of the cell at $\mathbf{p}$), which limits how many measurements need to be propagated between cells and therefore can be used to speed up processing. In a tiled implementation of the algorithm (see section III-B), propagation limits for internal cells can be readily computed, but determining the minimal propagation limit for the cells on the edge of a tile cannot be done without access to the adjacent tiles. As a simplification, choosing a propagation distance of $R/\sqrt{2}$ limits data transport with minimal overhead in propagation costs.

C. Stabilization of Data Density Estimation

A simple, although naive, estimate of data density is to compute $\rho(\mathbf{p}) = N(\mathbf{p})/||\mathsf{A}(\mathbf{p})||$, where $N(\mathbf{p}) = \sum_{\forall i}[\mathbf{s}_i \in \mathsf{A}(\mathbf{p})]$

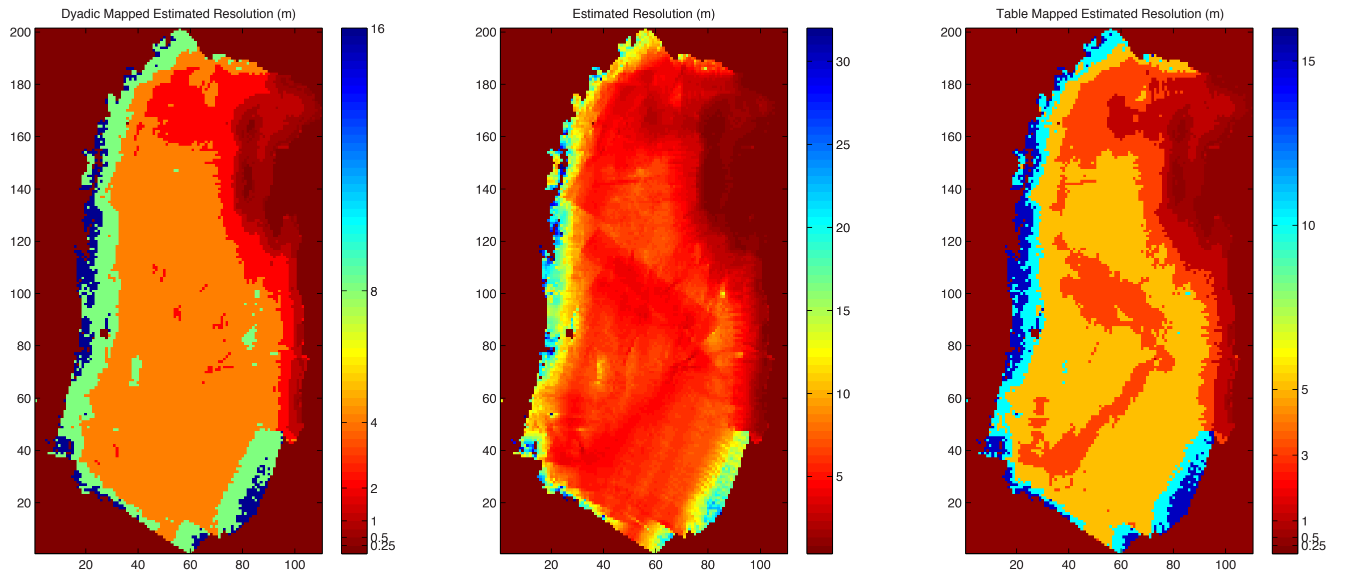

Dyadic Mapped Estimated Resolution (m)
Estimated Resolution (m)
Table Mapped Estimated Resolution (m)

Fig. 5. Example of mapped resolutions. The computed resolutions (center) are mapped to dyadic resolutions through (6) (left) or via a table of user-specified resolutions levels (right), here $\{0.25, 0.5, 1.0, 3.0, 5.0, 10.0, 15.0\}$-m. Mapping can possibly provide some advantages, but will also induce a structure in the representation of the seafloor that is not in the data and should be used with caution. Note that easting and northing scales here are in units of super grid cells, each of $R = 32$ m.

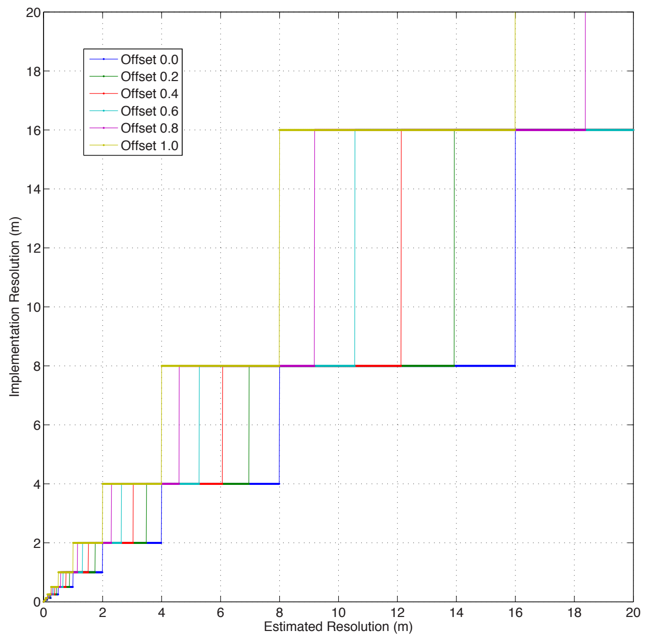


Fig. 6. Transition points for dyadic mapping of resolution levels. Without modification (offset 0), the mapping transitions to the next lowest level only after the estimated resolution drops below that level. Consequently, the data is treated at a higher than warranted resolution level for much of the time. This can be adjusted one full level by changing the offset value (s_m in (6)), although the non-linearity of the mapping equation also makes the response non-linear.

for measurement locations $\mathbf{s}_i \in \mathbb{R}^2$. Given the arbitrary size of the cells, however, we cannot guarantee that all of the cell will be occupied by measurements and therefore this estimate might underestimate the density of the data, and consequently predict (by (5)) a coarser resolution than is actually warranted by the data. This could readily result in under-representing potentially significant detail in edge areas, especially in shallow water.

We therefore approach the data density estimation problem by computing an estimate of the area of each cell occupied by the data observed, a variant of spatial point process density estimation schemes based on nearest neighbor statistics [19]. Ideally, we would compute an effective area for each measurement, and associate with it $\alpha_i \subset \mathbb{R}^2$, the section of the plane occupied by the measurement. The effective area occupied within a cell would then be $\alpha(\mathbf{p}) \triangleq \mathsf{A}(\mathbf{p}) \cap \bigcup_{i:\mathbf{s}_i \in \mathsf{A}(\mathbf{p})} \alpha_i$, and we could estimate the density directly as $\hat{\rho}(\mathbf{p}) = N(\mathbf{p})/\alpha(\mathbf{p})$. The computation of area intersections is complex, however, and therefore computationally expensive.

As a more efficient, if approximate, solution, we compute an alternative to the effective area by treating the measurement locations as random data, and computing estimates of sample mean and covariance of the positions,

$$\mathbf{m}(\mathbf{p}) = \frac{1}{N(\mathbf{p})} \sum_{i:\mathbf{s}_i \in \mathsf{A}(\mathbf{p})} \mathbf{s}_i \qquad (7)$$

$$\mathbf{C}(\mathbf{p}) = \frac{1}{N(\mathbf{p}) - 1} \sum_{i:\mathbf{s}_i \in \mathsf{A}(\mathbf{p})} (\mathbf{s}_i - \mathbf{m}(\mathbf{p}))(\mathbf{s}_i - \mathbf{m}(\mathbf{p}))^T \qquad (8)$$

(to minimize the potential for numerical overflow during accumulation of the statistics, we equivalently compute relative positions with respect to $\mathbf{x}(\mathbf{p})$). The covariance matrix eigenvalues, (λ_1, λ_2), correspond to the variance along the major axes of a second-order approximation to the distribution of the measurement positions, and therefore with appropriate coverage factors s_λ, an approximation to the effective area of the cell covered by the measurements is

$$a(\mathbf{p}) = \pi s_\lambda^2 \sqrt{\lambda_1 \lambda_2} \qquad (9)$$

although in this case we can also compute the area directly as

$$a(\mathbf{p}) = \pi s_\lambda^2 \sqrt{c_{11}c_{22} - c_{12}^2} \tag{10}$$

$$= \pi s_\lambda^2 \sqrt{c_{11}c_{22}} \sqrt{1 - r_\lambda^2} \tag{11}$$

$$r_\lambda = c_{12}/\sqrt{c_{11}c_{22}} \tag{12}$$

and we estimate

$$\rho(\mathbf{p}) = N(\mathbf{p})/a(\mathbf{p}) \tag{13}$$

as the raw measurement density.

This estimate is straightforward to compute, and can be accumulated as the measurements are read without having to buffer them in memory. However it is also fragile when only a small number of measurements are available, and indeed degenerate when $N(\mathbf{p}) = 2$. We stabilize the estimation process in two ways. First, we observe that estimation of the second eigenvalue becomes unstable as $|r_\lambda| \to 1$. We can minimize this effect by constraining the aspect ratio of the area ellipse, setting

$$\lambda_2' = \lambda_2 + (s_r\lambda_1 - \lambda_2)f(r_\lambda) \tag{14}$$

with some appropriate blending function $f(x)$. The choice of $f(x)$ is arbitrary although it should be even symmetric and bounded such that $0 \leq f(x) \leq 1$, $-1 \leq x \leq 1$ and monotone increasing with $|x|$. In the implementation described here, we use $f_{s_b}(x) = |x|^{s_b}$, with $s_b \approx 5 - 15$ to give relatively rapid blending from the directly computed value to the conditioned value as $|r_\lambda| \to 1$.

Second, we observe that the areas estimated by the method when only small numbers of observations are available can significantly over-estimate the effective area because the variance estimates are poor. This manifests as a heavy-tailed left-skewed distribution of area estimates. We control for this by making an empirical estimate of the degree of over-estimation using a simulation of randomly distributed points within a unit area. From the empirical distribution of areas estimated for a given number of simulated observations, we compute the scale factor, $s_a(n)$, that would cause the upper 99% centile to match that from a sample with 100 measurements (for which we assume that the variance estimation is stable). We then compute the modified area as $a'(\mathbf{p}) = s_a(N(\mathbf{p}))a(\mathbf{p})$. The corrections are applied sequentially.

The stabilized area, and therefore density, estimates are observed empirically to reflect the density of the observed data so long as the size of the super grid cells is sufficiently large to allow for multiple beams from a swath to be observed. Otherwise, Fig. 7, the algorithm estimates the area occupied by a single beam along-track, and can compute erroneously high data densities. This is essentially a form of across-track sample aliasing, exacerbated by the beam geometry of equiangular MBES, but the simple expedient of increasing the size of the super grid cells is not always desirable due to concerns about variable resolution adaptation rate.

The problem is one of scale and we address it by aggregating, where required, the partial estimates of sample mean and covariance of a neighboring group of super grid cells to estimate the mean and covariance of the group, Fig. 8. (This is essentially a matter of refactoring (7)–(8) for the whole group into the mean and covariance matrices for the individual cells.) The resulting estimates of resolution are necessarily smoothed with respect to those done on each cell, although they are significantly more stable. We therefore blend the two estimates together, Fig. 9, by preferring the individual cell resolution estimates where there is little difference between them and the aggregated version, and replacing them with the aggregated version where there is. (This is a good indicator of instability in the initial estimates.) Our initial experiments indicate that one level of aggregation (i.e., aggregating the

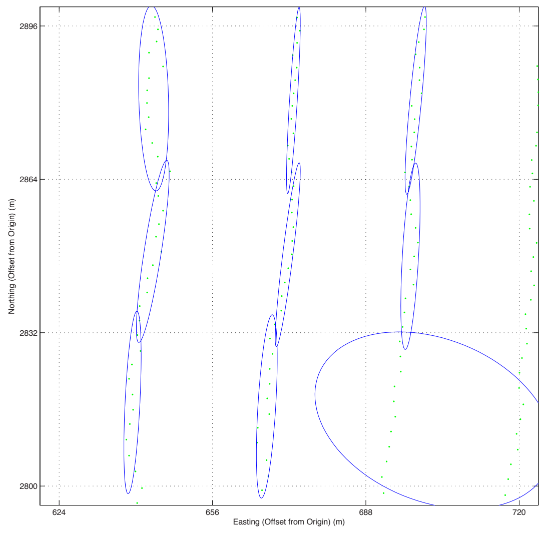


Fig. 7. Example of under-estimation of effective area due to super grid cell size. Increased spacing in the outer beams of an equiangular MBES mean that the data density drops low enough that only a single beam is observed within a cell. The algorithm estimates the effective area for the beam, but this does not reflect the actual area in use.

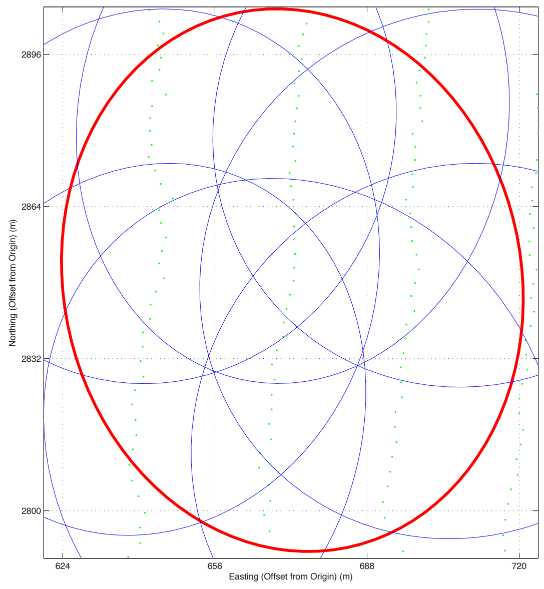


Fig. 8. Example of re-estimation of effective area by aggregation of individual cell estimates in the eight-nearest neighbors of each cell. Aggregation allows each cell to accumulate sufficient information to properly estimate the active area of the composite neighborhood (red ellipse), albeit at coarser resolution; estimates are still computed at each cell.

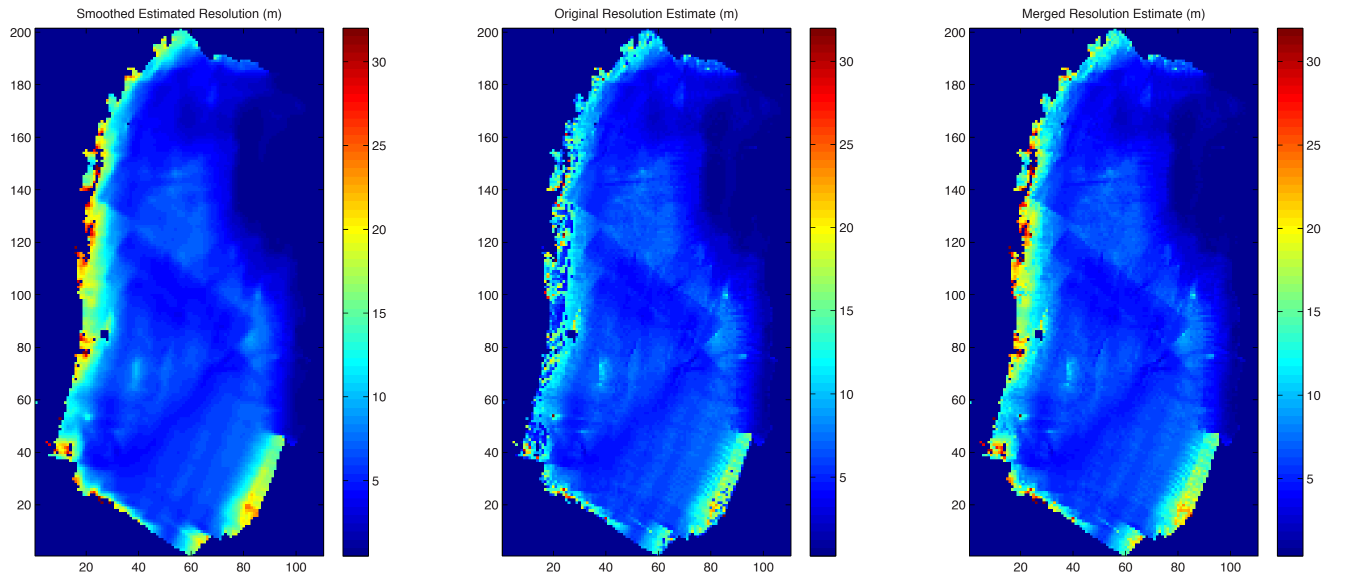


Fig. 9. Blended estimation of resolution. The resolution estimates at the cell level (center) are replaced by the aggregated estimates (left) to form the composite estimate (right) where there is a significant difference between the cell level and aggregated estimates. This preserves the cell level estimates (at finer resolution) where possible, but stabilizes the estimates where required.

eight-nearest neighbor cells for any given cell) produces stable enough estimates of resolution to use. Further aggregations could of course take place, with stability being assessed by tracking the change in resolution estimate across the different levels of aggregation.

An immediate consequence of this aggregation scheme is that increasing the super grid cell sizes is not required to stabilize the resolution estimation scheme. That is, we can by design use smaller super grid cells, possibly down to the minimum limit outlined previously, and aggregate to generate the finest usable resolution estimate of data density. This may be advantageous in certain circumstances, since it could be used to improve the variable resolution adaptation rates for better modeling of complex areas.

D. Multipass Estimation Workflows

The requirement of a first pass through the data to estimate resolution means that the workflow for algorithms of this type is necessarily different than those of conventional grid-based algorithms. In the simplest case, this would require that all data is collected prior to any processing being conducted, Fig. 10, a workflow only suitable for post-processing of datasets. In the most general case of randomly distributed data, there would be no means to determine when all data had been collected, and therefore no means to determine when a particular cell was ready for density and resolution estimation, and thence refinement. Typical data capture methodologies are not random, however, since they are tied to at most a small number of capture platforms, and in most cases collect data consistently within a relatively small geographical area at any one time. This structure allows us to improve the efficiency of processing, although it also requires that we allow that only partial refinement of the grid may be possible at any given time.

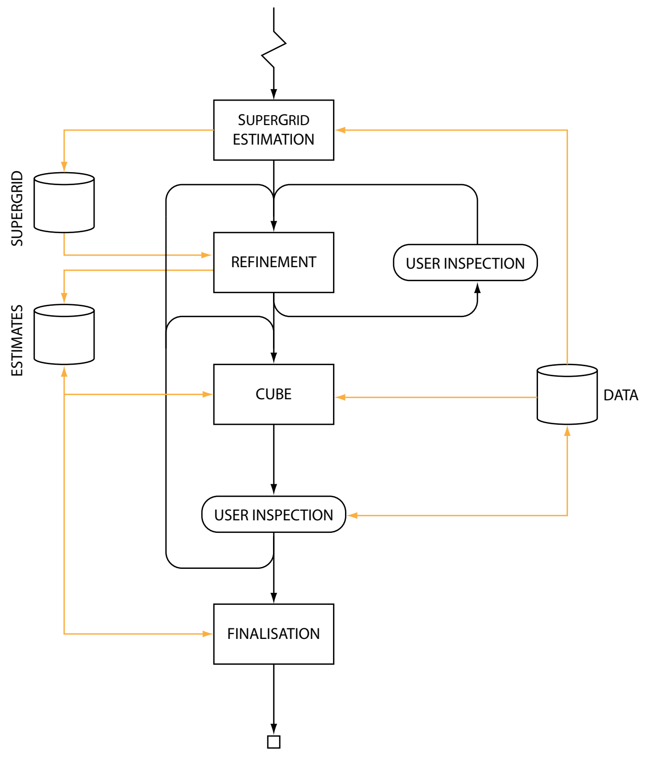


Fig. 10. Workflow where all data is available before processing begins. This allows for a simple two-pass system with a single resolution estimation component, although repeat processing might be required to adjust the resolution estimates or to re-CUBE the data to resolve signal-to-noise issues in the data.

A typical data capture methodology, for example, is the launch/mothership model where a number of hydrographic launches capture data within a local area during the day and return to a mothership at night to unload and process the data ready for the next day. This model can be readily

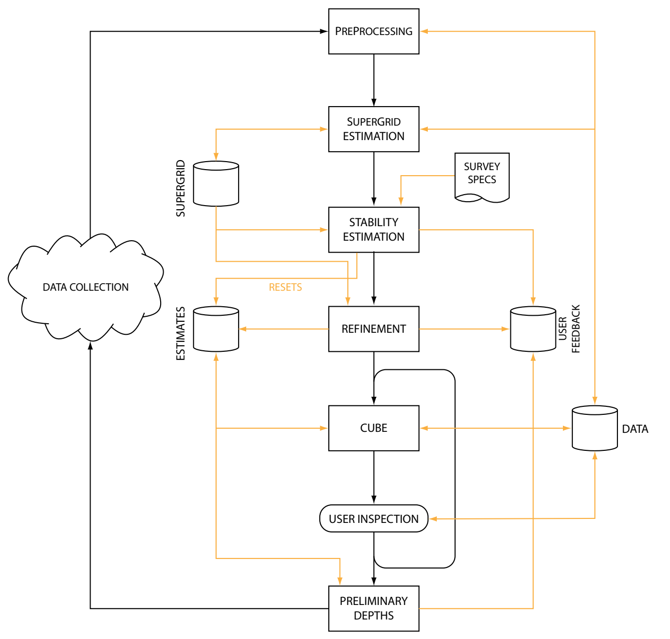


Fig. 11. Workflow where data becomes available in small chunks over an extended period of time. Spatial coherence of typical data collection methodologies mean that a limited amount of re-processing will be required each day, typically on the edges of the current coverage area.

accommodated in a workflow, Fig. 11, since in most cases the goal is that the area of coverage of each day's data should be considered "complete" at the end of the day. Since the coverage is simple to track, the meta-algorithm monitoring refinement can easily determine which cells have been modified and therefore require (re-)refinement. We expect that there would be some occasions where some cells would have to be revisited, for example on the edges of the extant survey area being re-covered on the next day, or if the current coverage was considered inadequate. Since each cell is independent of the others with respect to the refined grid, however, local effects remain local and easy to track, with limited recomputation of fine scale resolution estimates.

A more complex situation is implementing an algorithm such as CUBE in real time. The estimation algorithm itself was designed to be a real time-capable system, meaning that it can start to generate estimates of depth without having all of the data available, and then refine those estimates as more data is observed. The requirement for a preliminary pass through the data makes this more complex, but a potential workflow is sketched in Fig. 12. Here, suitable buffering is used to capture sufficient data to allow for estimation, and stability monitoring is used to determine when it is probably safe to trigger a refinement and start fine scale estimation. Such stability monitoring could be accomplished by observing that no data had been added to a cell for some significant time (e.g., by looking at how much data had been added to the system as a whole since the last data point added to the cell) or by looking for consistency in the estimates of effective area within the cell as data is being added. The burdens of data indexing and re-computation make this a potentially complex, although possible, meta-algorithm to implement.

There is a hidden benefit in the requirement for a first pass through the data. Since the CUBE algorithm does not require any *a priori* setup to generate depth estimates, if we co-locate a CUBE estimation node with the center of each super grid cell, we can estimate an approximate depth associated with each cell. Although these estimates will not be useful for hydrographic purposes, they do provide valuable aiding information, for example in estimating slope, determining areas of particular depth complexity, or determining likely bounds for egregious blunder detection and remediation, if required. These estimates of depth may also be used to provide aiding information in the data density estimation problem, since an approximate estimate of density can be constructed purely from depth given some appropriate assumptions about the collection system and typical survey methodologies.

III. Algorithm Implementation

A. *Memory Management*

Efficient memory management is an essential component of algorithms of this kind. We take a fairly conventional tile-based approach to management of a large computational data structure, since we expect that it will always be the case that the whole data structure will not be able to fit into core memory. Logically, our approach here uses a low-level memory map-based management layer to abstract the system-specific requirements in a type-safe manner, and provides a middleware layer to proxy high level requests for memory management of a particular data type within a tile. The middleware layer also handles splitting the data structures across multiple files, if required, and to provide for virtual address space management, demand paging of persistent storage, and least-recently-used cache replacement. The management system also provides a global memory manager that monitors the composite memory footprint of the whole system, and triggers requests for middleware managers to unload mapped resources if required in order for new objects to be constructed. The structure is designed to ensure that mapping requests are deferred as long as possible, and to minimize overhead when the mapping has already happened.

The principal data structures used in the algorithm are an object to accumulate the statistics associated with the resolution estimation sub-algorithm, and an object to represent the state required for a CUBE estimate node. The former is small (generally 92B) and is only required at the level of the super grid cells, so the memory overhead is small. This is managed as a single file with 32-bit address space since it only has to apply to a single tile. This configuration could allow tiles up to 6832 cells on a side, or perhaps 50–200 km wide (total area $\approx 730 - 11,662\,\mathrm{nm}^2$) for practical cell sizes in shallow water (and larger areas in deeper water), and is therefore not expected to be a limiting factor for implementations.

The data structure for a CUBE node consists of a core component of $m_c = 24\,$B/node, a data queue with elements of $m_q = 8\,$B/elements/node and typical lengths of $Q = 1 - 11$ elements for 8–88 B/node, and a collection of zero or more hypotheses of depth for $m_h = 40\,$B/hypothesis/node. The core and queue data structures are fixed at construction, but the

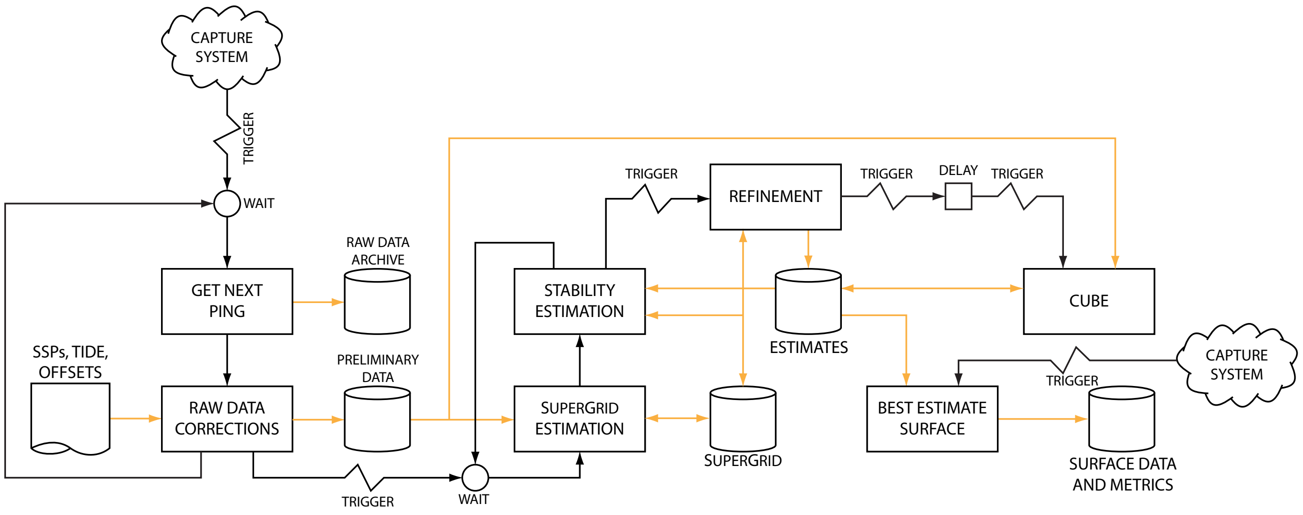


Fig. 12. A potential real time processing workflow. Causality requirements are met by buffering within the system and stability-driven triggering of refinement. Efficiency of such a scheme would depend on careful control of the stability estimation system and efficient indexing of the input data to allow controlled minimal re-estimation.

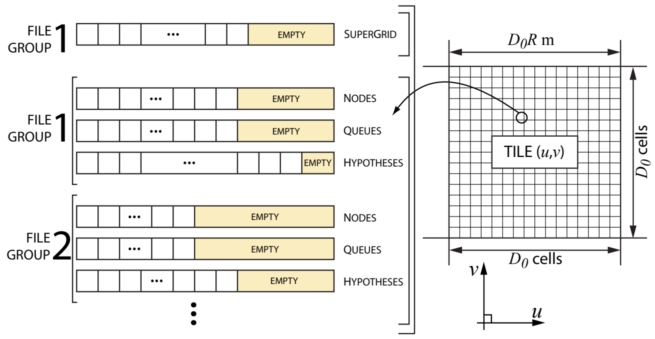


Fig. 13. Structure of the persistent files used for memory management. Each tile maintains one cell file, and one or more file-groups for the CUBE nodes, which consists of a file each for core data structures, queues and hypotheses.

number of hypotheses associated with a node is monotone increasing with data. The three data structures are mapped in individual files to improve the efficiency of new hypothesis creation, but are handled as a file-group with linear indexing within the group, so that all of the components are mapped at the same time, Fig. 13. Core and queue data structures for the same node have the same index number in a 32-bit space, and the core data structure contains the index within the hypothesis file for its first hypothesis; each hypothesis contains the index for the next, forming a singly-linked list. Hypothesis indices are also 32-bit integers, but the file is mapped in a 32-bit address space, so at most $\approx 10^8$ hypotheses could be supported per file and other limits generally reduce this even further.

Depending on the finest refinement resolution allowed by the user (typically 0.25–0.5 m), each tile may require more CUBE nodes than may fit into a single 32-bit address space, and hence into a single file; in order to preserve linear indexing we require that all the nodes associated with a single request appear in the same file-group. In addition, we do not know *a priori* how many hypotheses will be required for any node, and it is a requirement for efficient implementation that all hypotheses associated with a node occur within a single file-group. These constraints require that the middleware memory manager split the requests for CUBE nodes among multiple file-groups, generating new files automatically where there is insufficient space for the expected number of nodes or hypotheses within the current file-group. We implement this by splitting the uniform 32-bit index space into a b-bit file-group number and a $(32-b)$-bit index within the file-group. Including the CUBE nodes at the super grid cell centers, if we assume that in the worst case each cell will be refined to the finest possible resolution, r_{max}, we will require

$$N = D_0^2(1 + \lceil R/r_{\text{max}} \rceil^2) \text{ nodes} \tag{15}$$

per tile. (We assume that each tile is square with D_0 cells on a side.) We require a single value to represent “not valid” in both parts of the index, and therefore we can address at most

$$N_{\text{max}} = (2^b - 1)\left(2^{32-b} - 1 - \max\left\{D_0^2, \lceil R/r_{\text{max}} \rceil^2\right\}\right) \tag{16}$$

in all file-groups within one tile in the worst case. (We have to allow that we might lose up to $\max\left\{D_0^2, \lceil R/r_{\text{max}} \rceil^2\right\}$ indices at the end of each file-group due to the requirement that all nodes from one request are in one file-group.) We therefore require $N_{\text{max}} < N$ or that

$$D_0 \leq \sqrt{\frac{(2^b - 1)(2^{32-b} - 1 - \max\left\{D_0^2, \lceil R/r_{\text{max}} \rceil^2\right\})}{1 + \lceil R/r_{\text{max}} \rceil^2}} \tag{17}$$

Assume the value of b is such that this equation has a solution $D_0 \in \mathbb{R}$, i.e., $2^{32-b} - 1 \geq \max\left\{D_0^2, \lceil R/r_{\text{max}} \rceil^2\right\}$. This necessarily implies $2^{32-b} - 1 \geq \lceil R/r_{\text{max}} \rceil^2$ and therefore that the solution to (17) depends on D_0, but that the solution only exists if

$$2^{32-b} - 1 \geq \lceil R/r_{\text{max}} \rceil^2 \tag{18}$$
$$\Rightarrow \qquad b \leq 32 - \log_2(\lceil R/r_{\text{max}} \rceil^2 + 1) \tag{19}$$

It is clear that if the solution exists, it only exists for $D_0 > \lceil R/r_{\text{max}} \rceil$, and therefore (after some simplification), we can

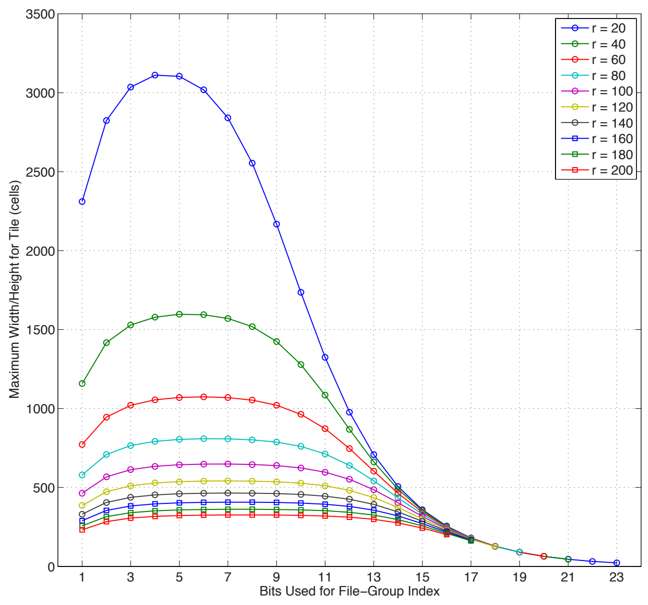


Fig. 14. Example of maximum tile sizes, D_0, in one dimension for refinement ratios $\lceil R/r_{\max}\rceil \in [20, 200]$, assuming no other constraints. Increasing refinement ratios generally reduce the maximum size of tile that can be accommodated, although the effect is not expected to be as large in practice due to other limits on allowable bit distribution within the file indices.

satisfy (17) if

$$D_0 \leq \sqrt{\frac{(2^b-1)(2^{32-b}-1)}{2^b+\lceil R/r_{\max}\rceil^2}} \tag{20}$$

and therefore guarantee that, given the user-defined refinement ratio $\lceil R/r_{\max}\rceil$, the tiles are sized such as to avoid the potential for index exhaustion, Fig. 14.

To establish a lower limit on b, note that the expected size of a node is $m_n = m_c + m_q Q + m_h \mathbb{E}[n_h]$ for the mean number of hypotheses, n_h. If we assume that conditions are just right, a file-group could contain $2^{32-b}-1$ of them, so that the total size of the virtual memory map for a file-group is $S = m_n(2^{32-b}-1)$ B/file-group. The worst case for the number of file-groups that need to be mapped simultaneously is generally when data occurs on the corner of four tiles, when we will need to have at least four, and more likely eight file-groups mapped simultaneously to avoid cache-swapping. We also require a ninth file-group space to support load-before-delete semantics, and therefore require $9m_n(2^{32-b}-1) < 2^{32}$ to successfully place all of the file-groups in a single 32-bit virtual memory map. Consequently, we require

$$b > 32 - \log_2\left(\frac{2^{32}}{9m_n}+1\right) \tag{21}$$

or, given that $2^{32}/(9m_n) \gg 1$,

$$b > \log_2(9m_n) \approx 3.17 + \log_2 m_n. \tag{22}$$

For the current configuration $m_n = 212$ B ($Q = 11$, $\mathbb{E}(n_h) = 2.5$), giving $b > 10.90$ bits irrespective of other parameters.

Configuration of the indexing, then, is a matter of ensuring that the maximum and minimum limits on the index partition point are satisfied. That is, we require

$$32-\log_2\left(\frac{2^{32}}{9m_n}+1\right) < b < 32-\log_2(\lceil R/r_{\max}\rceil^2+1) \tag{23}$$

to satisfy both requirements. Clearly, we require

$$32-\log_2(\lceil R/r_{\max}\rceil^2+1) \geq 32-\log_2\left(\frac{2^{32}}{9m_n}+1\right) \tag{24}$$

$$\Rightarrow \qquad \lceil R/r_{\max}\rceil \leq \sqrt{\frac{2^{32}}{9m_n}} \tag{25}$$

or $\lceil R/r_{\max}\rceil < 1500$ with $m_n = 212$ B. For a fairly typically hydrographic choice of $R = 30$ m, $r_{\max} = 0.25$ m, $\lceil R/r_{\max}\rceil = 120$, and for a slightly more extreme case of $R = 50$ m, $r_{\max} = 0.10$ m, $\lceil R/r_{\max}\rceil = 500$; it is therefore unlikely that most refinement scenarios will have difficulties in achieving a stable configuration, and in most cases we expect that there will be some range of index partition schemes that could be chosen. Note of course that the lower limit is a consequence of choosing a 32-bit memory model. If required, we could readily adopt a 64-bit memory model (at modest runtime costs), which would result in no effective lower limit, and provide access to larger tile sizes.

The sizes of tiles in the stable zone range from 319–1324 cells at $b = 11$ bits, for refinement ratios in the range $[20, 200]$. For the scenario previously, then, we might expect a tile of ≈ 510 cells (≈ 15.3 km) on a side, and for a more aggressive refinement of $R = 20$ m, $r_{\max} = 0.1$ m, we might expect ≈ 319 cells (≈ 6.4 km). It is therefore likely that any reasonable surveys will contain only a few active tiles, and since tiles are indexed by a 32-bit signed integer, the maximum addressable area within the data structure is many orders of magnitude larger than the circumference of the earth. Consequently the proposed data structure can readily represent arbitrary sized pieces of the surface of the earth within a single consistent data structure.

B. Global Grid Management

Any gridded data representation requires that we establish a local coordinate frame indexed by the row-column indices of the grid, and make a connection back to an absolute coordinate frame with respect to a suitable datum. Often this is done by defining a fixed bounding box with known georeference offset with respect to the absolute coordinate frame, usually because it is simpler to code algorithms with a fixed domain of support.

The combination of tiled data management, large addressable domain and commit-delayed resource allocation implemented here mean that this simplification is not required. Instead, we define *a priori* the relationship between the tile locations and the local projected coordinate frame so that the tile at (u, v) has projected coverage of $\{[u, u+1) \times [v, v+1)\}D_0R$, and all tiles are square with area $D_0^2R^2$ m^2. This global grid thus requires no user-defined bounding box or coordinate frame, and can be established without reference to the data.

We cannot in general construct all portions of such a representation, and in any case would likely prefer not to do so due to the distortions that would occur in outlying areas from any reasonable projection. In almost all cases, however, this will be unnecessary. The nature of data collection means that most surveys are conducted within a small, geographically coherent region, and therefore at most a small number of physically close tiles will become active in the course of data ingestion. We therefore monitor the tiles becoming active and maintain a persistent estimate of a rectangular bounding box on the active tiles as data is added. The rectangular bounding box is wasteful in the sense that depending on the orientation of the data collection to the primary axes of the projected coordinate frame there may be many tiles in the bounding box that are not actually active. Careful resource allocation, in particular delaying the commitment of resources until data availability is confirmed, minimizes this overhead. These principles are also applied at the tile level, so that unused "edge" space that derives from the relationship of the globally defined tile frame to the data position consumes minimal resources. This empty space can be readily detected and trimmed from the output data estimates, at least at the level of the low-resolution cells, due to the simple hierarchical relationship of the refined CUBE node grids within the super grid cells. Sub-cell trimming would be more difficult and computationally expensive, but with suitably sized super grid cells we expect the overhead to be sufficiently small to make this unnecessary.

C. Modularity and Extension

We intend the implementation of the estimator to be as modular as possible, so that it may be extended with as little disruption as possible. Object-oriented design that breaks the problem down to the main piecewise-variable resolution grid, a single tile within that grid, and a refined grid within one cell of that tile provides encapsulation of methods as might be expected. We have also established a common abstract interface for CUBE disambiguation methods, and another for resolution determination methods to allow new algorithms to be added with no other changes to the core code. All resolution computation sub-classes inherit the ability to map resolution (Fig. 5) from the base class.

The fine scale estimator interface is currently fixed, since we expect in almost all cases to use CUBE. Since the estimator is implemented in its own object, however, and has an associated memory manager that is derived from an abstract base class, it would be possible to re-base the estimator and have higher levels of the algorithm use the abstract interface instead. This would allow for run-time selection of different algorithms, as long as they supported the same basic geospatial data organization. We have refrained from doing so currently only due to efficiency concerns.

D. Client-Server Network Protocol

Code built in a research environment is typically designed to illustrate the concept of the algorithm being demonstrated, and is generally not required to be either efficient or stable enough for general use. In the hydrographic context, however, it is perhaps a legitimate question to ask whether it should. That is: if the algorithm is going to be used in the field to process data with inevitable safety of navigation concerns, how do we otherwise demonstrate that the algorithm implemented really is the algorithm that was demonstrated (and tested) at the research level?

A simple expedient would be to have the research code used for the field implementation, but the skill-set of most researchers is generally not the same as software engineers writing commercially-stable code, and although researchers may pay close attention to the science, it would be unusual for their code to be entirely supportable. At the same time, effort expended on developing code that addresses only implementation details might be considered wasted: if it must be re-implemented anyway, why bother? We would argue that, like it or not, research code should have a role in field implementations, and that the effort is not wasted—and on the contrary is actually required—for two primary reasons.

First, unless we consider the details of how the system might be implemented for field use we are unlikely to flush out any design problems that would adversely affect the efficiency of the algorithm. For example, if we expect the algorithm to be implemented on a multi-threaded CPU, or in a multi-processor compute cluster but do not consider issues of code factorization that minimize interaction between threads/processes and limits inter-process communication as much as possible, it will likely be difficult to effectively retrofit the abilities and maintain efficiency. Although we might conclude these in the abstract, only an implementation (even if it is not ultimately used) is likely to actually select appropriate algorithms.

Second, unless we approach a version of the algorithm that could be used as a field-ready implementation, we are unlikely to be able to guarantee traceability of the algorithm in the future without really extensive effort and collaboration with implementation partners. In effect, it is more efficient to work on the implementation details only once.

The extent to which this model can be used in practice is still to be determined. However, we have used the development here to consider the question in a practical example, designing in addition to the core estimator libraries a client-server architecture that allows clients to access computational resources in a network-aware manner, Fig. 15. The client-server pair implement an example of the command design pattern [20], where the objects that represent commands (e.g., construct/load/save a grid, add data, refine the grid, extract results) are constructed at the client side, passed across the network, reconstituted at the server side and then queued for execution when resources become available; acknowledgements are returned to ensure reliable transport. When results become available they are stored internally at the server and an alert message is sent back to the client, triggering optional user-defined actions. The whole process is event driven so that it better maps to modern user interface design, and is made opaque to the client application through a local stub-object on which the user interface code calls local methods. Both client and server are multi-threaded, which allows for staged resource commitment at the server: commands can be executed at reception, immediately on inspection at the executive, or in parallel on the compute thread pool, as is appropriate for the nature of the command and expected execution duration. The server keeps statistics of execution time, memory committed, etc. to allow for better real time adjustment of resources.

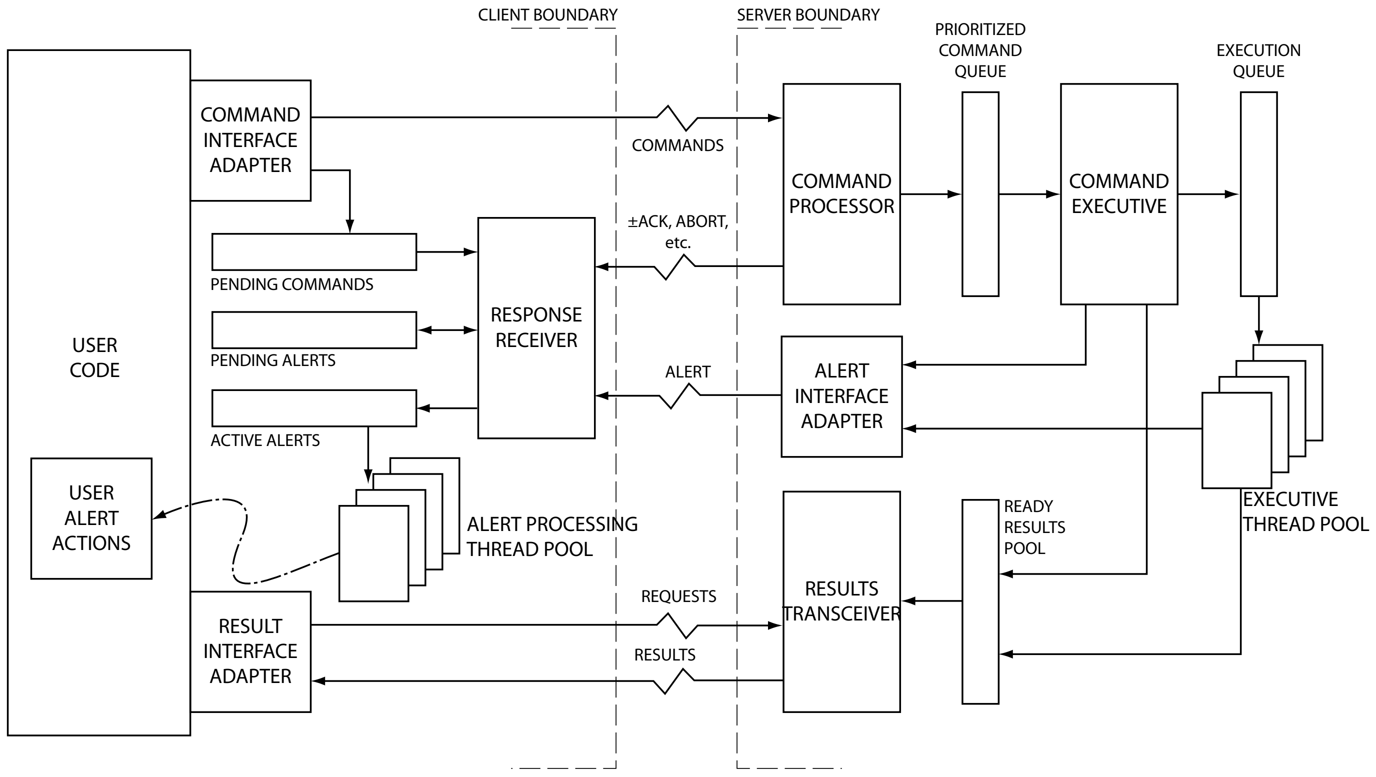


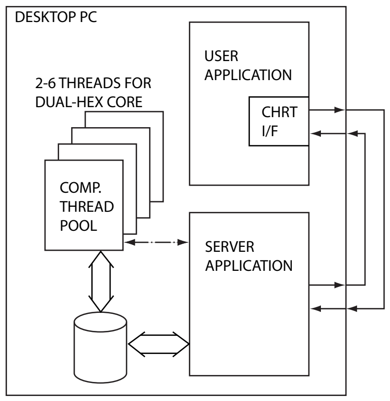
Fig. 15. Client-server architecture to support the CHRT estimation algorithm. The client interface provides an object on which the user code calls local methods for service; these are translated into network packets, transferred to the server, and executed at some later time either in the executive module, or in the computation thread pool. When a command completes, the server sends an alert to the client, triggering a user-defined action; results are buffered at the server until requested.

This implementation structure has a number of benefits. First, it exploits and benefits from modern processor architectures that support multi- and hyper-threading. Second, the separation of client and server at the network protocol means that the same client interface can be maintained across multiple different server back-end implementations, so long as they implement the protocol correctly. It is therefore possible to have the same client interact with a local server, remote computer, or computational cluster, Fig. 16, with no client modification. In fact, the current implementation supports a broadcast discovery of local network computational resources and can select the resource most appropriate for its needs based on metrics provided by the server in its response packet. Third, a standard network protocol mandated by the code means that implementations may choose to support just the client, the server, or both, depending on needs and (commercial) goals. By the same token, users may choose to select a client application because of its user interface, and a separate server application because of its efficiency or supported hardware, rather than be tied to a tightly coupled implementation from a single vendor. Finally, and perhaps most importantly, establishing a network protocol means that it is possible to observe the operation of both sides of the algorithm without requiring any details of the internal implementation, and it is possible to exercise the implementation directly and in isolation. These capabilities allow for external debugging and algorithm verification, even after field deployment.

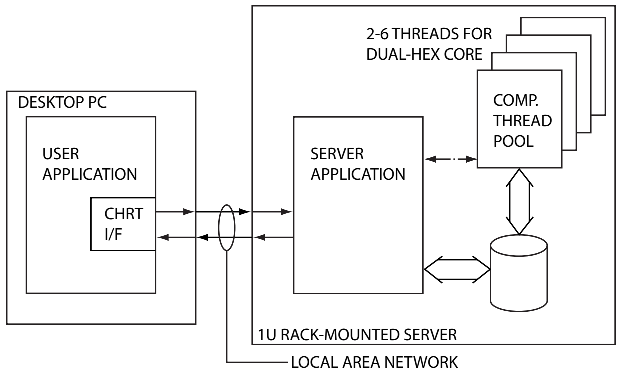
The arguments for this approach to implementation testing are, unfortunately, somewhat circular. That is, we assume that the research code may not necessarily be sufficiently robust for field use and might need to be modified; modified code, however, may not necessarily implement the same algorithm as the research code, and might be difficult to verify. There are several possible solutions to this, including obtaining third-party expertise within the research environment to "industrialize" the code-base before field use, and partnering with one or more implementation entities to reach a common consensus on the code-base as part of the later stages of development. Which is likely to be more successful has yet to be determined.

IV. Conclusions

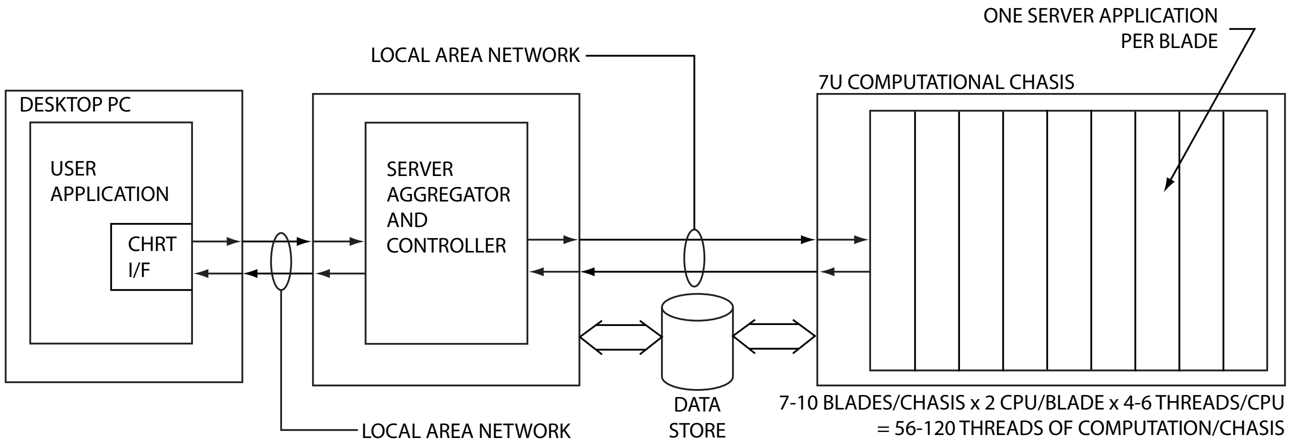
We have outlined the requirement for, and the design of, a piecewise-variable resolution bathymetric estimator, and considered some of the questions inherent in implementation of same. The core observation here is that the data structure should be optimized for computational efficiency, which here means controlling the cost per sounding processed, but must still support resolution changes at arbitrary resolutions as indicated by the data itself. We approach the problem with a hierarchically nested grid scheme where initially coarse cells are refined to the appropriate resolution supportable by the data based on an initial estimate of data density extracted from the observations. The data structure is therefore a data-driven union of disjoint locally-regular grids, allowing variable resolution while maintaining computational efficiency. Use of locally-regular grids also allows us to use common robust



(a) Local implementation model for client and server on the same computer.



(b) Remote implementation model for server on a separate computer to offload computational effort.


ONE SERVER APPLICATION PER BLADE
LOCAL AREA NETWORK
7U COMPUTATIONAL CHASIS
DESKTOP PC
USER APPLICATION
CHRT I/F
SERVER AGGREGATOR AND CONTROLLER
DATA STORE
LOCAL AREA NETWORK
7-10 BLADES/CHASIS x 2 CPU/BLADE x 4-6 THREADS/CPU = 56-120 THREADS OF COMPUTATION/CHASIS

(c) High-performance implementation model for a compute cluster with separate aggregator.

Fig. 16. Possible models for client-server implementation. The local model, (a), is appropriate for stand-alone computers, laptops, etc.; the remote model, (b), allows for one machine to offload the computational costs (for example where it is also responsible for data capture); the high-performance model, (c), is appropriate for post-capture computations in a shared environment and has the advantage that it allows for optimization of I/O bandwidth and computational load through a dedicated connection to the data store. Note in all cases that the application interface to CHRT at the user's local machine is identical: to the user, the back-end implementation can be transparently changed, and indeed could be changed dynamically in real time if required.

estimation algorithms designed for this configuration; in this case, we use a re-implemented version of CUBE. The algorithm is supported by a network-aware client-server implementation that is multi-threaded in both control and processing.

Variable resolution estimation comes at a cost. Apart from the slightly increased complexity of the data structure, we require a full first pass through the data to determine the data density and therefore the appropriate estimation resolution. Although we simultaneously make a low-resolution estimation of depth to assist the algorithm, this is a non-trivial cost. We are convinced that the benefits justify the expense, however, and it is hard to imagine how an algorithm of this type could be made data adaptive without a similar effect.

Estimation of data density is theoretically simple, but is made more complex by edge cases where cells are only partially filled by the data, or where only a few data observations are available. This can be exacerbated by choice of cell size and wide dynamic depth range (or equivalently data density) within the survey area. Estimation of effective area within a cell, combined with multi-scale analysis of the data density estimates, can tackle these problems, although we are actively pursing alternative techniques and co-estimation strategies to robustify the estimates.

We have hypothesized that the piecewise-variable resolution grid is sufficient representation for the sort of data that we typically have to process in the hydrographic context, but accept that this may not be the case everywhere. The independence of the file scale refinements of the cells, however, means that more complex but flexible representations could be substituted where required without changing the fundamental structure of the algorithm. How often this is likely to be necessary is an interesting, although still open, question.

The proposed algorithm's use of a two pass approach to data processing will require some reconsideration of typical data workflows. We have shown, however, that the most common data capture and processing scenarios can be relatively readily incorporated, and real time processing is still possible, although the control meta-algorithm might prove complex to implement.

Research and development of these sorts of algorithms necessarily involves some consideration of the implementation details of the algorithm. Serious consideration of these problems, however, has traditionally been deferred until after the research phase has completed, and are often passed along to the field implementation stage. We believe, however, that they may need to be tackled at an earlier stage of the process if the algorithms being developed are to avoid later difficulties due to design decisions that ignore the eventual implementation. A more stable research code-base, and such features as a client-server decoupling of control interface and computational structure, could result in better traceability of algorithms from research to field and ease all stages of testing, but likely requires resources in excess of those typically available at the research stage. It is possible therefore that some form of robustified research code-base might be a better approach for field implementation, although this also implies questions of code-sharing, cooperative development and stability that are themselves challenging. The optimal interaction pattern remains, therefore, an open—and very interesting—question.

ACKNOWLEDGEMENTS

This work is supported by NOAA grant NA10NOS4000073 (`brc`) and NOAA's Coast Survey Development Lab (`gr`). The authors would also like to acknowledge the contributions of James Hiebert (previously with NOAA's Hydrographic Survey Technology Program), who contributed to the core idea of locally-regular, but piecewise-variable grid representations in what has become CHRT.

REFERENCES

[1] C. B. Lirakis and K. P. Bongiovanni, "Automated multibeam data cleaning and target detection," in *Proc. IEEE Oceans*, vol. 1, Marine Technology Society/IEEE Ocean Engineering Society. Providence, RI, USA: MTS/IEEE, September 2000, pp. 719–723.

[2] Z. Du, D. E. Wells, and L. A. Mayer, "An approach to automatic detection of outliers in multibeam echosounder data," *Hydro. J.*, vol. 79, pp. 19–25, 1996.

[3] C. Ware, L. Slipp, K. W. Wong, B. Nickerson, D. E. Wells, Y. C. Lee, D. Dodd, and G. Costello, "A system for cleaning high volume bathymetry," *Int. Hydro. Review*, vol. 69, pp. 77–94, 1992.

[4] N. Debese and P. Michaux, "Détection automatique d'erreurs ponctuelles présentes dans les données bathymétriques multifaisceaux petits fonds," in *Proc. Canadian Hydro. Conf.* Toronto, Canada: Can. Hydro. Soc., 2002.

[5] N. Debese, "Multibeam echosounder data cleaning through an adaptive surface-based approach," in *Proc. US Hydro. Conf.* Norfolk, VA: The Hydrographic Society of America, May 2007.

[6] J. Eeg, "On the identification of spikes in soundings," *Int. Hydro. Review*, vol. 72, pp. 33–41, 1995.

[7] G. Canepa, O. Bergem, and N. G. Pace, "A new algorithm for automatic processing of bathymetric data," *IEEE J. Ocean. Eng.*, vol. 28, no. 1, pp. 62–77, 2003.

[8] L. Arge, K. G. Larsen, T. Mølhave, and F. van Walderveen, "Cleaning massive sonar point clouds," in *Proc. ACM Int. Conf. on Advances in GIS*, A. el Abbadi, D. Agrawal, M. Mokbel, and P. Zhang, Eds., Assoc. Comp. Machinery. San Jose, CA, USA: Assoc. Comp. Machinery, November 2010, pp. 152–161.

[9] B. R. Calder and L. A. Mayer, "Automatic processing of high-rate, high-density multibeam echosounder data," *Geochem., Geophys. and Geosystems (G3) DID 10.1029/2002GC000486*, vol. 4, no. 6, 2003.

[10] B. R. Calder, "Automatic statistical processing of multibeam echosounder data," *Int. Hydro. Review*, vol. 4, no. 1, pp. 53–68, 2003.

[11] J. G. Proakis and D. K. Manolakis, *Digital Signal Processing*, 4th ed. Prentice Hall, 2006.

[12] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. San Fransisco, CA: Morgan Kaufmann, 2006.

[13] A. Fischer and P. Z. Bar-Yoseph, "Adaptive mesh generation based on multi-resolution quadtree representation," *Int. J. Num. Methods in Eng.*, vol. 48, pp. 1571–1582, 2000.

[14] H. Borouchaki and P. J. Frey, "Adaptive triangular-quadrilateral mesh generation," *Int. J. Num. Methods in Eng.*, vol. 41, no. 5, pp. 915–934, March 1998.

[15] W. Cao, W. Huang, and R. D. Russell, "A study of monitor functions for two-dimensional adaptive mesh generation," *SCIAM J. Sci. Comput.*, vol. 20, no. 6, pp. 1978–1994, 1999.

[16] H. Borouchaki, P. L. George, F. Hecht, P. Laug, and E. Saltel, "Delaunay mesh generation governed by metric specifications. Part I. Algorithms," *Finite Elem. Anal. Design*, vol. 25, pp. 61–83, 1997.

[17] H. Borouchaki, P. L. George, and B. Mohammadi, "Delaunay mesh generation governed by metric specifications. Part II. Applications," *Finite Elem. Anal. Design*, vol. 25, pp. 85–109, 1997.

[18] G. Rice and B. Calder, "A quantitative approach to the resolution of bathymetric representation," in *Proc. US Hydro. Conf.* Norfolk, VA: The Hydrographic Society of America, May 2009.

[19] N. A. C. Cressie, *Statistics for Spatial Data*. Wiley, 1993.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vissides, *Design Patterns*. Addison-Wesley Professional, 1994.