SIAM J. IMAGING SCIENCES
Vol. 6, No. 1, pp. 1–31

Processing Terrain Point Cloud Data*

Ronald DeVore†, Guergana Petrova†, Matthew Hielsberg‡, Luke Owens§, Billy Clack¶, and Alok Sood¶

Abstract. Terrain point cloud data are typically acquired through some form of Light Detection And Ranging sensing. They form a rich resource that is important in a variety of applications including navigation, line of sight, and terrain visualization. Processing terrain data has not received the attention of other forms of surface reconstruction or of image processing. The goal of terrain data processing is to convert the point cloud into a succinct representation system that is amenable to the various application demands. The present paper presents a platform for terrain processing built on the following principles: (i) measuring distortion in the Hausdorff metric, which we argue is a good match for the application demands, (ii) a multiscale representation based on tree approximation using local polynomial fitting. The basic elements held in the nodes of the tree can be efficiently encoded, transmitted, visualized, and utilized for the various target applications. Several challenges emerge because of the variable resolution of the data, missing data, occlusions, and noise. Techniques for identifying and handling these challenges are developed.

Key words. surface reconstruction, point clouds, Hausdorff metric, compression, adaptive splines

AMS subject classifications. 65D17, 65D18, 41A15

DOI. 10.1137/110856009

1. Introduction.

Terrain point clouds are now quite ubiquitous and are used in a variety of applications including autonomous navigation, change detection, and field of view calculations. The point clouds themselves are too cumbersome and large to be used for these purposes. They need to be converted to a simpler platform that is more efficient and still contains all of the features of the terrain, present in the point cloud, that are needed for these applications. A naive approach would be to take local averages of data heights to obtain pixel intensities (and therefore a pixelized image) and then employ the techniques of image processing to make a conversion into a wavelet or other multiscale representation. However, this approach is not successful for several reasons. Foremost among these is that terrains are not images. They have certain topology and geometry that must be extracted and maintained for successful

*Received by the editors November 21, 2011; accepted for publication (in revised form) August 20, 2012; published electronically January 10, 2013. This research was supported by the ARO/DoD contract W911NF-07-1-0185; the NSF grants DMS-0810869 and DMS-0900632; the Office of Naval Research contracts ONR-N00014-08-1-1113, ONR-N00014-09-1-0107, and ONR-N00014-11-1-0712; the AFOSR contract FA9550-09-1-0500; and the DARPA grant HR0011-08-1-0014. This publication is based in part on work supported by award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST).
http://www.siam.org/journals/siims/6-1/85600.html
†Department of Mathematics, Texas A&M, College Station, TX 77843 (rdevore@math.tamu.edu, gpetrova@math.tamu.edu).
‡Institute of Scientific Computation, Texas A&M, College Station, TX 77843 (hielsber@tamu.edu).
§Automated Trading Desk, Mount Pleasant, SC 29464 (lowens@atdesk.com).
¶Department of Computer Science and Engineering, Texas A&M, College Station, TX 77843 (clac81@neo.tamu.edu, alok169@tamu.edu).

applications. A second related point is that the usual least squares metrics used in image processing do not match the intended applications for terrain maps. For example, capturing long thin structures such as poles, towers, and wires is essential for navigation, but is not given priority in Peak Signal-to-Noise Ratio (PSNR) metrics employed for images. Another important point is that terrain point clouds usually have missing data, occlusions, and noise which do not appear in most other applications.

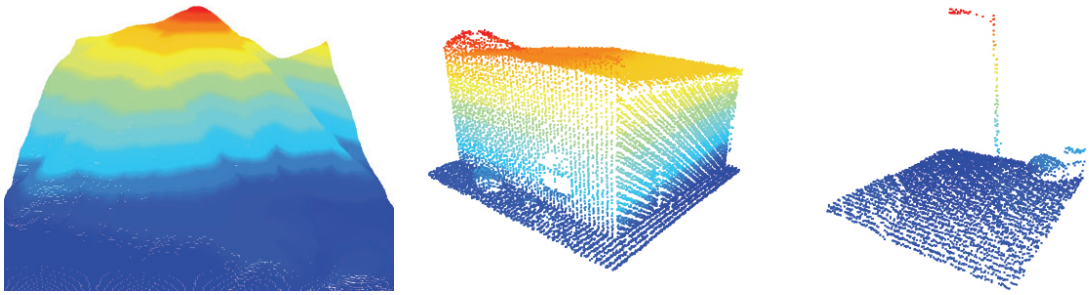
While surface reconstruction from point clouds is now a dominant theme in computer graphics, very little of this work addresses terrain data per se. The notable exception is the paper [26], which proposes a Morse tree structure to represent terrain but falls short of providing an implemented processing platform. Of course, one could argue that one can simply apply one of the vast number of surface processing algorithms in computer graphics. However, these algorithms are typically built for high resolution data, which is not the case for general terrain data, which suffers from occlusions, missing data, noise, and variable resolution.

The purpose of the present paper is to give a terrain point cloud processing algorithm. The algorithm outputs both a piecewise polynomial surface and the global Hausdorff distance between this surface and the (denoised) point cloud. It is based on the following basic principles:

- Distortion is measured in the Hausdorff metric, which we argue closely matches the intended applications.
- The decomposition is organized in a multiscale octree giving coarse to fine resolution.
- Each node of the tree corresponds to a dyadic cube and is ornated with a low degree polynomial fit to the point cloud on this cube and with other attributes, such as the local Hausdorff error of this fit.
- The tree is organized into subtrees each of which corresponds to a certain accuracy of resolution in the Hausdorff metric.
- The tree and nodal information can be efficiently encoded using predictive encoding.
- Upon receiving the tree and nodal information, the user can easily convert this information to a format that matches the intended application.
- Primitives such as normals, curvature, and other information can be extracted from the tree and nodal information.

Multiscale decompositions and local polynomial fits are often used in surface fitting (see [8, 15, 16, 17, 19, 25, 27, 28]). Among the things that separate our work from others are the measure and guarantee of performance in the Hausdorff metric and the fact that our methods can be applied to nonhomogeneous and nondense data.

A terrain point cloud D is a finite set of three-dimensional data points $\mathbf{x} := (x, y, z)$. Such point clouds are typically obtained from a sensor or from images using Structure From Motion calculations. In contrast to image processing or computer graphics, a canonical collection of point clouds that could be used to develop and test algorithms for terrain data is not available. We propose such a collection here, which can be used and added to by other researchers. These data sets as well as an implementation of the proposed algorithms can be downloaded from the web site http://www.math.tamu.edu/∼hielsber/MURI_PointCloud/index.html (some data sets are not available to the general public because of priority restrictions). All terrain point clouds in the collection are obtained via Light Detection And Ranging (LiDAR) sensing (real data) or a simulation of LiDAR sensing (synthetic data). Some of the data sets are part of



Maple Mountain Building Light Pole

Figure 1. *Three examples of denoised LiDAR point clouds.*

the AFRL/MNG VEAA Data Set #1, obtained by taking a LiDAR scan of some real world topology. Others are derived using a synthetic LiDAR sensor which allows us to create data sets with representative features. Our data sets contain high-altitude terrain scan (Test 1: Maple Mountain), portion of building (Test 2: Eglin data), light pole (Test 3: Eglin data), and several complicated point clouds containing buildings, light poles, fences, and wires (Tests 4–6). The purpose of the first three sets is to test the algorithm on standard and simple mountain and urban (buildings and thin structures) landscape. The last three data sets test the ability of the algorithm to represent and reconstruct a complicated urban scene without special parameter tuning for thin structure detection. An illustration of the point clouds that we use in order to demonstrate the results of our algorithm is given in Figure 1. Details for each tested data set are provided in the description of our numerical results (see Tests 1–6).

Note that our algorithms are developed for data sets which contain only the point cloud. In some settings one also has available the position and orientation of the sensor, which makes the task of processing the point cloud much simpler.

This paper is organized as follows. In sections 2–6, we shall discuss the various tasks that have to be completed in our terrain processing engine. For each of these tasks, we describe the current algorithm which completes the task. After this, in section 7, we describe our terrain processing algorithm. In sections 8–9, we discuss how the terrain processing algorithm is used for some of the directed applications.

2. Dyadic cubes and octrees. Given a point cloud D, we process D by using a dilation (with each coordinate dilated by the same factor) and a shift of the data in D so that the output data D' lies in the unit cube $\Omega = [0,1]^3$. There are many such transformations from which we choose the following one. After the transformation, the range of the new x_i should be $[1/2 - \delta_1, 1/2 + \delta_1]$ with $0 \le \delta_1 \le 1/2$. Similarly, the range of the y_i should be $[1/2 - \delta_2, 1/2 + \delta_2]$ with $0 \le \delta_2 \le 1/2$. For the new z_i, we require that the range be of the form $[2^{-\kappa-1} - \delta_3, 2^{-\kappa-1} + \delta_3]$, where $2^{-\kappa-2} < \delta_3 \le 2^{-\kappa-1}$. Thus the z values will all lie in $[0, 2^{-\kappa}]$. Finally, we require one of the δ_i, $i = 1, 2, 3$, to equal $1/2$. We denote by **INIT** the subroutine which takes as input D and has as output $\mathbf{INIT}(D) = (D', \kappa)$.

A dyadic subcube Q of Ω of side length 2^{-j} (volume 2^{-3j}) is of the form $2^{-j}([k_1, k_1+1) \times [k_2, k_2+1) \times [k_3, k_3+1))$ with the integers k_1, k_2, k_3 satisfying $0 \le k_1, k_2, k_3 < 2^j$. We use the

standard convention that the above intervals are closed on the right when the right endpoint is one. The children of Ω are the eight dyadic subcubes with side length 2^{-1}. Each of these children has eight children itself and so on. Thus, the dyadic cubes organize themselves into an octree with root Ω. We denote the set of all dyadic subcubes of Ω by $\mathcal{Q}$. Some of the cubes in $\mathcal{Q}$ will not contain data from D'. We denote by $\mathcal{Q}^*$ the set of all cubes in $\mathcal{Q}$ that are occupied and, for each cube $Q \in \mathcal{Q}^*$, we denote by $D_Q := D' \cap Q$ the set of all data points in Q. The cubes $Q \in \mathcal{Q}^*$ determine a subtree of the full dyadic octree.

A finite collection $\mathcal{T} \subset \mathcal{Q}$ is called an octree if whenever a cube is in $\mathcal{T}$, then its parent and all of its siblings are in $\mathcal{T}$. Our processing algorithm will input the data set D and a tolerance η and output a finite octree $\mathcal{T} = \mathcal{T}_\eta$.

3. Distortion metrics. The development and assessment of terrain processing algorithms requires a metric which measures the distortion between two surfaces. In the case of image processing, this metric is usually chosen as a least squares metric and is tabulated in the PSNR. We argue in this section that least squares metrics are not appropriate for the intended applications of terrain maps and that a more suitable way to measure distortion is through the Hausdorff metric.

Given two sets A, B in $\mathbb{R}^3$, the one-sided Hausdorff distance from A to B is given by

$$\delta(A,B) := \sup_{a\in A} \inf_{b\in B} |a-b|\,, \tag{3.1}$$

where $|\cdot|$ is the standard Euclidean distance in $\mathbb{R}^3$. Obviously, this guarantees that for any $a \in A$, there is $b \in B$ whose Euclidean distance to a does not exceed $\delta(A,B)$. The Hausdorff distance between two sets A and B is then defined as

$$\delta_H(A,B) := \max\{\delta(A,B), \delta(B,A)\}. \tag{3.2}$$

Notice that δ_H is a metric on sets and in particular satisfies the triangle inequality

$$\delta_H(A,C) \le \delta_H(A,B) + \delta_H(B,C) \tag{3.3}$$

for any sets A, B, C.

A depiction of the Hausdorff distance between a point cloud D and a curve C in two dimensions is given in Figure 2, where $\delta(D,C)$ is the one-sided distance from D to C and $\delta(C,D)$ is the one-sided distance from C to D.

Given two finite sets $A, B \in \Omega$, we denote by $\mathbf{DIST}(A,B)$ the subroutine that returns the one-sided Hausdorff distance from A to B, and by $\mathbf{HAUS}(A,B)$ the subroutine that computes $\max\{\mathbf{DIST}(A,B), \mathbf{DIST}(B,A)\}$, which is the Hasudorff distance between A and B. For our implementation of **HAUS**, we utilized the spatial searching features in the CGAL library [2] to reduce the computational complexity of this algorithm from quadratic to log-linear.

We shall frequently need to compute the Hausdorff distance between a finite set A and a continuum surface S, such as a plane or quadric surface. There are fast algorithms for computing the Hausdorff distance between two surfaces [3, 6, 14, 29], most of which involve the discretization of the surfaces in question.

Normally, as is traditional in numerical analysis, we do not indicate the fact that a subroutine does not provide exact computation. However, we will make an exception in the

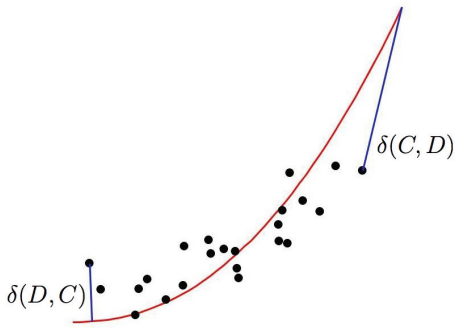


Figure 2. *Hausdorff distance from point cloud to curve and curve to point cloud.*

case of computing Hausdorff distances between a point set A and a surface S, since this subroutine plays a special role in our algorithm. With this in mind, we choose a numerical tolerance $\gamma = 2^{-m}$ with m a positive integer. We consider the tiling $\mathcal{Q}_m$ of Ω into dyadic cubes of side length γ and create the point set S_γ, which consists of all points $\mathbf{x}$ such that $\mathbf{x}$ is the center of a cube $Q \in \mathcal{Q}_m$ which contains points of the surface S; see Figure 3. Then $\mathbf{DIST}_\gamma(A, S) := \mathbf{DIST}(A, S_\gamma)$ takes as input the finite set A, the surface S, and the numerical tolerance γ and returns the distance between A and S_γ. Since $\delta_H(S, S_\gamma) \leq \frac{\sqrt{3}}{2}\gamma$, this computation is an approximation to $\delta(A, S)$, which is accurate to tolerance $\frac{\sqrt{3}}{2}\gamma$. We can similarly compute $\mathbf{DIST}_\gamma(S, A)$. The subroutine $\mathbf{HAUS}_\gamma$ takes as input any pair of A, S and the tolerance γ and returns $\mathbf{HAUS}_\gamma(A, S) := \max\{\mathbf{DIST}_\gamma(A, S), \mathbf{DIST}_\gamma(S, A)\}$.

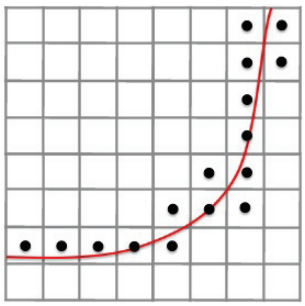


Figure 3. *Discretization of a two-dimensional curve for Hausdorff computation.*

To understand why the Hausdorff metric is appropriate for terrain applications, let us consider two such applications. First consider the problem of navigating an unmanned vehicle, for example, a Micro Air Vehicle. Sensors extract a point cloud, which describes surfaces that the vehicle must avoid (no fly zones). Suppose that we know that the true surface S has a Hausdorff distance ϵ from the point cloud; this is an assumption on the quality of the data which is necessary to proceed with any certainty. Suppose that we use the point cloud to find a surface $\hat{S}$ which is within η of the point cloud in the Hausdorff metric. Then, whenever the vehicle remains a distance greater than $\eta + \epsilon$ from the approximate surface, it is guaranteed

L_2 L_∞ Hausdorff

Figure 4. *Resulting line of sight between the true and approximate surfaces as observed from street level moving north along an urban canyon. Field of view metric comparison: false positives (yellow), false negatives (red), correctly classified (gray), unobserved regions (black); see* [30].

to avoid the true surface. Moreover, no weaker metric (such as least squares) can guarantee such a performance.

As a second example, consider observing a terrain surface S from an observation point $\mathbf{x}$. The field of view describes what the observer can see from this vantage point taking occlusions into consideration. If we construct an approximation $\hat{S}$ to S from given point cloud data, then the field of view will not be accurate but will have false positives and false negatives. The quality of the approximate field of view will depend on the metric used to compute the approximation $\hat{S}$. In Figure 4, we give a comparison of computing the field of view using three metrics with a comparable error. The left image uses the least squares metric, the center image uses a maximum z-value deviation, and the right uses the Hausdorff metric. The points colored yellow are false positives, and those colored red are false negatives. The gray colored points are correctly classified. One sees that when distortion is measured in the Hausdorff metric we obtain the largest agreement with the true field of view.

4. Multiscale decompositions. Our processing algorithm is based on a multiscale decomposition of the surface using low degree algebraic surfaces to locally approximate the given point cloud D. In the algorithms implemented in this paper, the algebraic surfaces are either planes (corresponding to linear polynomials) or certain quadric surfaces (corresponding to special choices of quadratic polynomials described below). In this section, we explain the multiscale structure which is based on dyadic cubes and also how we extract the polynomial fits.

We wish to associate a local polynomial fit to the point cloud on the cube Q. To do this, we need to assume that Q contains sufficiently many points from the point cloud. In our algorithms, we assign a tolerance K and require Q to have at least K points from D whenever we ask our algorithm to assign a polynomial fit. The value of K can be set by the user. Of course, it should be larger than the number of degrees of freedom in the local polynomial fits, but it also should be large enough to avoid fitting noise in the data. In our implementation, we set $K = 10$ for the examples in this paper. This value of K was motivated by considerations from learning theory; see [9, 10].

4.1. Planar fits to the data. Suppose that D_0 is a subset of D. We look to fit the data D_0 by a plane. Any plane can be described as the zero set of a linear function $L(\mathbf{x}) = \boldsymbol{n} \cdot \mathbf{x} + c$ on $\mathbb{R}^3$, where $\boldsymbol{n}$ is a unit normal to the plane and c is a suitable constant. Also $|L(\mathbf{x})|$ is the

distance of any given point $\mathbf{x}$ to this plane.

While we have emphasized that we measure the distortion of our data fitting in the Hausdorff metric, it turns out that finding the best planar fit to the data D_0 in the Hausdorff metric is too computationally intensive. So we will take another approach to finding a linear fit by using Principal Component Analysis (PCA). We emphasize, however, that we continue to evaluate the performance of this fit in the Hausdorff metric.

PCA finds a linear function $L(\mathbf{x}) = L_{D_0}(\mathbf{x}) = \boldsymbol{n}_{D_0} \cdot \mathbf{x} + c_{D_0}$ from the set $\mathcal{L}$ of all linear functions such that

$$L_{D_0} := \operatorname*{argmin}_{L \in \mathcal{L}} \sum_{\mathbf{x} \in D_0} |L(\mathbf{x})|^2. \tag{4.1}$$

Using Lagrange multipliers, one can solve this minimization problem by first finding the eigenvalues and eigenvectors of the covariance matrix for the components x, y, z of the data D_0:

$$\Sigma_{D_0} := \begin{pmatrix} \operatorname{cov}(x,x) & \operatorname{cov}(x,y) & \operatorname{cov}(x,z) \\ \operatorname{cov}(x,y) & \operatorname{cov}(y,y) & \operatorname{cov}(y,z) \\ \operatorname{cov}(x,z) & \operatorname{cov}(y,z) & \operatorname{cov}(z,z) \end{pmatrix}, \tag{4.2}$$

where $\operatorname{cov}(x,y) := \sum_{(x,y,z) \in D_0} (x - \bar{x}_{D_0}, y - \bar{y}_{D_0})$, with $\bar{x}_{D_0}$ the mean of the x components of the data D_0, and the sum is taken over all the data points in D_0. Then $\boldsymbol{n}_{D_0}$ is a multiple of the eigenvector corresponding to the smallest eigenvalue of Σ_{D_0}, and c_{D_0} is chosen so that $\sum_{\mathbf{x} \in D_0} L_{D_0}(\mathbf{x}) = 0$. Note that there is not necessarily a unique solution to (4.1) when the two smallest eigenvalues are equal. Since the matrix Σ_{D_0} is 3×3 and positive semidefinite, this eigenvalue problem can be solved efficiently using any standard solver such as LAPACK, CGAL, or Geometric Tools [1, 2, 4]. The plane associated to L_{D_0} is our default planar fit to the data D_0, and in most instances it performs satisfactorily.

In summary, we define a subroutine **PCA** that takes as input the data set D_0, with the property $\#(D_0) \geq K$, and returns $\mathbf{PCA}(D_0) = (\lambda_1, \lambda_2, \lambda_3; v_1, v_2, v_3)$, which are three eigenvalues (written in decreasing order) and corresponding eigenvectors of the matrix Σ_{D_0}.

From the algorithm **PCA**, we define a new algorithm **PLANE**, which takes as input any axis-oriented three-dimensional rectangular box B (parallelpiped). It outputs the set $\hat{S}_B := \mathbf{PLANE}(B)$, which is the restriction to B of the plane obtained from $\mathbf{PCA}(D_B)$, where $D_B := D \cap B$.

4.2. Quadratic fits to the data. The quality of approximation to the data on a box B can generally be improved by replacing planes by algebraic surfaces that are zero sets of higher degree polynomials. We use quadratic polynomials in our processing algorithm described below. If one utilizes general quadratic functions in three variables, then the zero sets are quadric surfaces that may have branches. To avoid this, we limit the types of quadratic polynomials, and hence the quadric surfaces that can be used in our algorithms. To describe this limitation, we return to the coordinate system given by $\mathbf{PCA}(D_B)$.

Given a box B, we use a change of coordinates that replaces the canonical x, y, z coordinates by the coordinates given by the basis of the three eigenvectors found by PCA written in order of increasing size of the eigenvalues (with ties handled arbitrarily). This maps the

coordinates $\mathbf{x} = (x, y, z)$ to new coordinates $\mathbf{u} = (u, v, w)$. We denote by $\bar{D}_B$ the transformed data points. We shall use quadratics $P(u, v) \in \mathcal{P}_2$, where $\mathcal{P}_2$ is the set of all quadratics in u, v. We define

$$P_B := \operatorname*{argmin}_{P \in \mathcal{P}_2} \sum_{(u,v,w) \in \bar{D}_B} |P(u, v) - w|^2, \tag{4.3}$$

which is a least squares fit to the data on B. We denote by $\hat{S}_B$ the quadric surface, described as the set of all $(u, v, w) \in B$ such that $w = P_B(u, v)$. The solution to the least squares problem (4.3) is easy to find from the Moore–Penrose formula for least squares problems.

We define a subroutine **QUAD** that takes as input a box B with $\#(D_B) \geq K$ and returns the quadric surface $\hat{S}_B$. We want to emphasize once again that $\hat{S}_B$ is not the best Hausdorff fit to the data from quadric surfaces of the above type. This would be too expensive to compute. However, we shall still measure the quality of fit of $\hat{S}_B$ to our point cloud by using the Hausdorff metric.

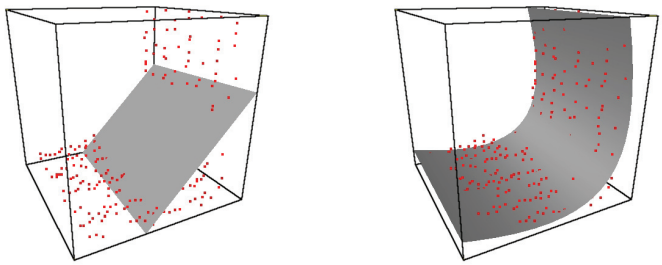


Figure 5. *Comparison of planar and quadratic fits.*

Figure 5 shows the planar and quadratic fits to a portion of test data from Figure 1, representing part of the building's wall and nearby ground. Table 1 gives the one-sided Hausdorff distances between the point cloud and the fits. Clearly, for these data points a quadratic fit substantially outperforms the planar fit.

Table 1

Comparison of one-sided Hausdorff distances for Figure 5 (values are given with respect to the original data units).

	PLANE	**QUAD**
$\delta(D_Q, \hat{S}_Q)$	0.9799	0.2722
$\delta(\hat{S}_Q, D_Q)$	0.4192	0.3164

4.3. General fitting. We found that in applications such as compression, the additional overhead required to encode the output of **QUAD** sometimes outweighed any benefit. It should be noted that while we found this to be true in several experiments, exhaustive testing has not been done. Our algorithms allow the user to specify the use of either **PLANE** or **QUAD** for any given experiment. We define the subroutine **FIT** as the generic delegate

for the **PLANE** and **QUAD** subroutines. It takes as input the box B and outputs $\hat{S}_B := \mathbf{FIT}(B)$.

5. Improving Hausdorff fits. The planar and quadric surfaces $\hat{S}_Q$ on a cube $Q \in \mathcal{Q}^*$, described in the previous sections, are generated from least squares methods and may not be accurate in the Hausdorff metric. One way to remedy this situation is to subdivide Q into its children and repeat the fitting on each child. However, often the reason for the poor fit is simply because the planar/quadratic fit spreads over the entire cube, whereas the data is very localized on the cube. This manifests itself in having $\delta(D_Q, \hat{S}_Q)$ small, but $\delta(\hat{S}_Q, D_Q)$ large; see Figure 6. There are ways to remedy this situation by localizing the surface. In this section, we describe some methods for implementing this, as well as testing and applying such methods when $\delta(D_Q, \hat{S}_Q)$ is smaller than our preassigned Hausdorff error threshold, but $\delta(\hat{S}_Q, D_Q)$ is not.

5.1. Bounding boxes. Our first technique is to find one or two clusters of the data in Q, described by bounding boxes, in order to more accurately represent the underlying geometry. Certainly more than two clusters can exist in Q, and slight modifications in the proposed algorithm could be done to handle these cases. However, our experience shows that due to the nature of the terrain data and the high resolution of the underlying tree structure, these cases do not happen too often. This is the reason to limit our search to two such clusters.

The first step of the bounding box method is to find the smallest axis-aligned box that contains D_Q. The octree data structure keeps track of the minimum and maximum x, y, and z coordinates for D_Q in each cube Q dynamically as the tree is being built. These minimum and maximum coordinates provide three intervals I_x, I_y, I_z representing the smallest possible fit of an axis-aligned bounding box around the data. Thus, the single bounding box on Q is simply $B_Q := I_x \times I_y \times I_z$. We now compute $\delta(\hat{S}_{B_Q}, D_Q)$ and $\delta(D_Q, \hat{S}_{B_Q})$, where $\hat{S}_{B_Q} := \mathbf{FIT}(B_Q)$. If both quantities are smaller than our preassigned Hausdorff error tolerance, we accept $\hat{S}_{B_Q}$ and do not further refine Q. Figure 6 shows a synthetic example (in two dimensions), where the algorithm terminates with an accurate Hausdorff fit using the surface $\hat{S}_{B_Q}$ in B_Q; therefore in this case, the use of bounding boxes eliminates the need to continue subdividing Q.

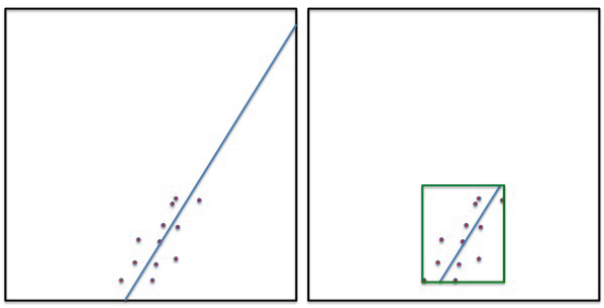


Figure 6. *An example of a single bounding box.*

We define the subroutine $\mathbf{BOX}_\gamma$ that takes as input a dyadic cube Q and the numerical tolerance γ. It finds the bounding box B_Q, the fit $\hat{S}_{B_Q}$, and the Hausdorff distance $\hat{\eta}_Q := \mathbf{HAUS}_\gamma(\hat{S}_{B_Q}, D_Q)$. Thus $\mathbf{BOX}_\gamma(Q) = (B_Q, \hat{S}_{B_Q}, \hat{\eta}_Q)$.

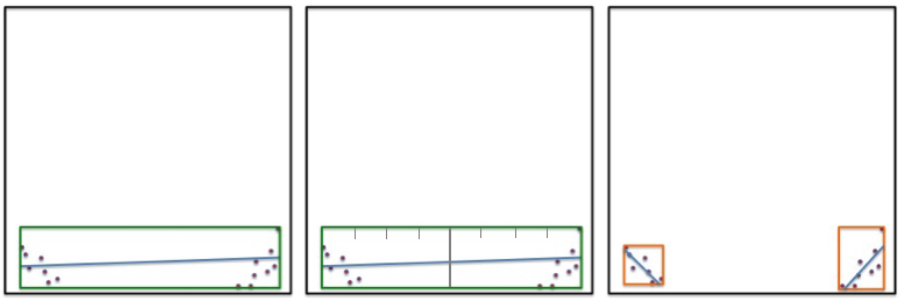


Figure 7. *An example of the sliding-partition method.*

If $\hat{\eta}_Q$ is bigger than the prescribed Hausdorff tolerance, then we attempt to cluster the data into two groups. There are several existing algorithms in the literature for clustering data sets that could be used for this purpose. We have implemented two methods: the *sliding-partition method* and the *k-means clustering algorithm*.

The *sliding-partition method* described here does not operate on Q, but instead on the single bounding box B_Q found in the previous step. The reason for doing this is that B_Q is already the tightest axis-aligned fit around the points, thus eliminating external empty space. This algorithm requires a user-defined parameter N specifying the number of partitions to use. The *sliding-partition method* proceeds as follows. Let x_i, $i = 1, \dots, M$, with $M = \#(D_Q)$, be the projection of the points in D_Q onto the x-axis written in increasing order $x_i \le x_{i+1}$, $i = 1, \dots, M-1$. Let $P_{\hat{d}} := [x_1, x_1 + \hat{d} \cdot L]$ and $P'_{\hat{d}} := (x_1 + \hat{d} \cdot L, x_M]$, $\hat{d} = 1, \dots, N$, be the interval choices along the axis, where $L := (x_M - x_1)/(N+1)$. If we fix a value of $\hat{d}$, then D_Q can be partitioned into two sets $D_1(\hat{d})$ and $D_2(\hat{d})$, where $D_1(\hat{d})$ contains all the data points $x \in D_Q$ whose projections are in $P_{\hat{d}}$, and $D_2(\hat{d})$ contains all the data points whose projections are in $P'_{\hat{d}}$. We find the two bounding boxes $B_1(\hat{d})$, $B_2(\hat{d})$ for $D_1(\hat{d})$, $D_2(\hat{d})$, respectively, each by the single bounding box method $\mathbf{BOX}_\gamma$. We also compute the best linear (or quadratic) fit to the data on each of these bounding boxes using **FIT**. The Hausdorff errors on each box are also computed, and the maximum of these two errors is stored as the error for this partition. We then choose the minimum of these errors over all partitions and denote this value by $\eta_Q^{(1)}$. We do a similar calculation starting with the y (respectively, z) values of the data and thereby obtain error $\eta_Q^{(2)}$ (respectively, $\eta_Q^{(3)}$). We now choose the bounding boxes corresponding to the smallest of the three errors $\eta_Q^{(1)}$, $\eta_Q^{(2)}$, $\eta_Q^{(3)}$, which we denote by $\hat{\eta}_Q$.

We denote by $\mathbf{BOX2}_\gamma$ the *sliding-partition method*, which takes as input a dyadic cube Q and the numerical tolerance γ and outputs two boxes $B_Q^{(1)}$, $B_Q^{(2)}$, their surface fits $\hat{S}_{B_Q^{(1)}} = \mathbf{FIT}(B_Q^{(1)})$, $\hat{S}_{B_Q^{(2)}} = \mathbf{FIT}(B_Q^{(2)})$, and the Hausdorff error $\hat{\eta}_Q$, which is the maximum of the $\mathbf{HAUS}_\gamma$ error on each of these boxes. Thus, we have $\mathbf{BOX2}_\gamma(Q) = (B_Q^{(1)}, B_Q^{(2)}, \hat{S}_{B_Q^{(1)}}, \hat{S}_{B_Q^{(2)}}, \hat{\eta}_Q)$. An example of the *sliding-partition method* for a synthetic two-dimensional data set is given in Figure 7, where the result of $\mathbf{BOX}_\gamma$ is shown on the left, an intermediate partition from the *sliding-partition method* is shown in the middle, and the result of $\mathbf{BOX2}_\gamma$ is shown on the right.

An alternative to the *sliding-partition method* is the *k-means* algorithm; see [22]. This is the most widely used unsupervised clustering algorithm, and it shows good results in our experiments with terrain data. However, it is more computationally intensive than the *sliding-partition method* described above.

The ideal *k-means clustering algorithm*, when applied to three-dimensional point cloud data, groups points into k distinct clusters based on their Euclidean distances. Given a set of data points $\mathbf{x} := (x, y, z)$ and a cluster count k (we choose $k = 2$ in our case to split the points into two groups), the algorithm outputs a cluster assignment where each input point is assigned to one of the k clusters. The mean of each cluster is computed. To evaluate a given cluster assignment, the sum of the squares of distances from the points to their corresponding cluster mean is computed. One then determines the k clusters that minimize this sum over all possible assignments.

The ideal *k-means* algorithm is too computationally intensive, so it is typically replaced by an iterative algorithm. A typical iterative algorithm begins by choosing k points $P_1, \ldots, P_k$ at random. The clusters ${C_1}^{(0)}, \ldots, {C_k}^{(0)}$ are determined by assigning a point from the data set to cluster ${C_j}^{(0)}$ if its distance to P_j is the smallest. Now the means ${M_j}^{(1)}$ of all clusters ${C_j}^{(0)}$ are computed, $j = 1, \ldots, k$. A new cluster assignment ${C_1}^{(1)}, \ldots, {C_k}^{(1)}$ is determined as above, where a point from the data set is assigned to cluster ${C_j}^{(1)}$ when its distance to ${M_j}^{(1)}$ is smallest. This cluster assignment and mean computation are iterated until a user-defined number of iterations is reached.

Our implementation uses the *k-means++* algorithm [5], where, rather than simply picking the points $P_1, \ldots, P_k$ at random, one uses a greedy method that guarantees their maximum separation. Our implementation of the above ideas is the subroutine $\mathbf{2MEANS}_\gamma$ (i.e., $k = 2$). This algorithm takes as input a dyadic cube Q and the numerical tolerance γ and applies the 2-*means++* algorithm to find two clusters, C_1 and C_2. We then define the smallest axis-aligned boxes $B_Q^{(1)}$, $B_Q^{(2)}$, their surface fits $\hat{S}_{B_Q^{(1)}} := \mathbf{FIT}(B_Q^{(1)})$, $\hat{S}_{B_Q^{(2)}} := \mathbf{FIT}(B_Q^{(2)})$, and the Hausdorff error $\hat{\eta}_Q$, which is the maximum of the result of $\mathbf{HAUS}_\gamma$ on each of these boxes. Thus, $\mathbf{2MEANS}_\gamma(Q) = (B_Q^{(1)}, B_Q^{(2)}, \hat{S}_{B_Q^{(1)}}, \hat{S}_{B_Q^{(2)}}, \hat{\eta}_Q)$. We also define the subroutine $\mathbf{CLUSTER}_\gamma$ as the generic delegate for the $\mathbf{BOX2}_\gamma$ and $\mathbf{2MEANS}_\gamma$ subroutines.

6. Preprocessing to remove noise and outliers. The LiDAR acquired terrain point cloud data are typically noisy. The type and amount of noise vary depending on the sensor and data collected. In this section, we discuss a few preprocessing algorithms we have implemented to identify and remove some of the present noise and outliers.

Locally optimal projection. We have implemented the locally optimal projection method from [21]. We refer the reader to [21] for its description and motivation. We have applied this method locally on D_Q. This method performs rather well on point clouds coming from a C^2 surface, but when this is not the case, e.g., for terrain surfaces, it leads to unwanted artifacts and missing detail.

Outlier measure. We have also implemented the outlier measure method from [32], where a measure for determining whether or not a data point is noise is presented. This method assigns to each point P a score $\chi(P)$. A simple thresholding of these scores produces the

denoised point cloud. The score χ for P is computed as $\chi(P) = \omega_1\chi_1(P)+\omega_2\chi_2(P)+\omega_3\chi_3(P)$, where ω_1, ω_2, ω_3 are scalar weights defined by the user, and χ_1, χ_2 and χ_3 are determined as follows. First, to compute $\chi_1(P)$, a value of k is chosen by the user, then a least squares plane is fit through the point's k-nearest neighbors, and the distances of all points in that neighborhood to the plane are calculated. Then, $\chi_1(P) = d_P(d_P + \bar{d})^{-1}$, where d_P is the distance from P to the plane and $\bar{d}$ is the mean distance of all points in that neighborhood to the plane. Next, χ_2 is computed by $\chi_2(P) = q_P(q_P + 2r/\sqrt{k})^{-1}$, where q_P is the distance from P to the center of the sphere, determining the k-neighborhood, and r is its radius. Last, $\chi_3(P) = N_P k^{-1}$, where N_P is the number of k-neighborhoods created from the k-neighbors of P that do not contain P. Our tests show that this method performs well at identifying noise along smooth surfaces as well as outliers. However, we have noticed that it may treat edges, corners, and boundaries between regions with different sampling densities as noise.

One-sided Hausdorff outlier measure. To enhance denoising with respect to the Hausdorff method, we have developed our own denoising method called the *one-sided Hausdorff outlier measure*. It is based on the idea that outliers should be relatively far from any local fitting of the data. This method depends on a user-defined tolerance ζ and assigns to each point P a score $\xi(P)$. As in the previous method, a simple thresholding of the calculated scores produces the denoised point cloud.

To compute $\xi(P)$, we do the following with a user-defined value of k. For each point R we consider the set $N(R)$ of its k-nearest neighbors and we fit a least squares plane through this set. For every point $P \in N(R)$, we consider a sphere with center P and radius ζ. If this sphere intersects the plane, we assign $d_R(P) = 0$. Otherwise, we set $d_R(P)$ to be the reciprocal of the number of data points in this sphere. Thus, we assign a number $d_R(P)$ to each point P, viewed as a point from the k-nearest neighbors of R. In general, a point P belongs to several k-nearest neighbors and has associated to it several numbers $d_R(P)$. Now, the score $\xi(P)$ is the average of $d_R(P)$ taken over all k-nearest neighborhoods $N(R)$ that contain P.

This algorithm can be viewed as an extension of the outlier measure method with weights $(\omega_1, \omega_2, \omega_3) = (1, 0, 0)$. We have found in our tests that it performs similarly to the outlier measure method. However, it is more successful than the outlier measure method when processing corners or edges. It does not treat them as noise, especially when the user-defined tolerance ζ is bigger than the local sampling rate.

7. Processing algorithm. In this section, we shall describe our processing algorithm **MAIN**. The input to this algorithm consists of the output data set D' of **INIT**, a desired target Hausdorff error η^*, a tree depth ℓ, a numerical tolerance γ, such that $\gamma \leq \frac{1}{\sqrt{3}}\eta^*$, and a user choice of whether the algorithm is to use planar or quadric surfaces in the subroutine **FIT** and whether the algorithm is to use the *sliding-partition method* or *k-means* in the subroutine $\mathbf{CLUSTER}_\gamma$. The output is an octree $\mathcal{T}$ with depth at most ℓ. Every node of the tree is adorned with either a local polynomial fit to the data or a flag that says no local fit is available. We define the "surface" S_{out} to be the union of the local polynomial fits on the leaf nodes of $\mathcal{T}$. S_{out} is typically not the type of surface we see in terrain processing because it is fragmented (primarily due to the use of bounding boxes) and discontinuous and is not oriented with definable inside and outside. In section 9, we will explain how to obtain smooth oriented surfaces from the output of **MAIN**.

Each node of $\mathcal{T}$ that has a polynomial fit also contains an upper bound $\hat{\eta}_Q$ for the local Hausdorff error. The Hausdorff error $\hat{\eta}_Q$ will be less than our threshold η^* for each leaf node Q which contains a polynomial fit, except in the case where further subdivision of the node Q is artificially halted by the user's choice of ℓ. Note that the global Hausdorff error $\eta_{S_{\text{out}}}$ between S_{out} and D', computed with accuracy γ, is also an output of **MAIN**.

Our processing algorithm is quite simple; it recursively applies a single subroutine called $\mathbf{PROCQ}_\gamma$. To describe this subroutine, let us first define $\eta := \frac{\eta^*}{2}$ to be our computational tolerance. The subroutine $\mathbf{PROCQ}_\gamma$ takes as input a cube Q, the computational tolerance η, the numerical tolerance $\gamma \leq \frac{1}{\sqrt{3}}\eta^*$, and the specification of linear or quadric fit in **FIT**. It outputs the current best fit $\hat{S}_{D_Q}$ to the data D_Q on Q and the Hausdorff error $\hat{\eta}_Q$. Thus if Q is not flagged, then $\mathbf{PROCQ}_\gamma(Q, \eta^*) = (\hat{S}_{D_Q}, \hat{\eta}_Q)$, where $\hat{S}_{D_Q}$ is either $\hat{S}_Q$, $\hat{S}_{B_Q}$ or $(\hat{S}_{B_Q^{(1)}}, \hat{S}_{B_Q^{(2)}})$. The subroutine is described as follows:

if $\#(D_Q) < K$ **then**
  Flag Q and exit.
end if
Call $\mathbf{FIT}(Q) = \hat{S}_Q$, and $\mathbf{HAUS}_\gamma(\hat{S}_Q, D_Q) = \hat{\eta}_Q$, set $\hat{S}_{D_Q} = \hat{S}_Q$.
if $\mathbf{DIST}_\gamma(\hat{S}_Q, D_Q) > \eta$ and $\mathbf{DIST}_\gamma(D_Q, \hat{S}_Q) \leq \eta$ **then**
  Call $\mathbf{BOX}_\gamma(Q) = (B_Q, \hat{S}_{B_Q}, \hat{\eta}_Q)$, set $\hat{S}_{D_Q} = \hat{S}_{B_Q}$.
  if $\hat{\eta}_Q > \eta$ **then**
    Call $\mathbf{CLUSTER}_\gamma(Q) = (B_Q^{(1)}, B_Q^{(2)}, \hat{S}_{B_Q^{(1)}}, \hat{S}_{B_Q^{(2)}}, \hat{\eta}_Q)$, set $\hat{S}_{D_Q} = (B_Q^{(1)}, B_Q^{(2)})$.
  end if
end if
if $\hat{\eta}_Q > \eta$ **then**
  Set $\hat{S}_{D_Q} = \mathbf{FIT}(Q)$, and $\hat{\eta}_Q = \mathbf{HAUS}_\gamma(\hat{S}_Q, D_Q)$.
end if

Notice that if the local fit does not provide Hausdorff error $\leq \eta$, then an attempt is made to fit the data on Q through one or two bounding boxes. However, if these fail and we are not at finest level ℓ, then we return to the entire cube Q in the further processing. That is, we never subdivide bounding boxes, only the entire cube Q.

Given the above specification of $\mathbf{PROCQ}_\gamma$, the main processing algorithm $\mathbf{MAIN}(D', \eta^*, \ell, \gamma) = (\mathcal{T}, \eta_{S_{\text{out}}})$ maintains a set of cubes to be processed and a set of cubes that have been assigned to the octree. The method is described as follows:

Initialize the cube processing list as $\{\Omega\}$ and the octree $\mathcal{T}$ as empty.
for Each cube Q in the processing list **do**
  Call $\mathbf{PROCQ}_\gamma(Q, \eta^*)$.
  if Q is flagged by $\mathbf{PROCQ}_\gamma$ **then**
    Add Q and the flag to $\mathcal{T}$.
  else
    Add Q and its adornments $\hat{S}_{D_Q}$ and $\hat{\eta}_Q$ to $\mathcal{T}$.
    if $|Q| > 2^{-3\ell}$ and $\hat{\eta}_Q > \eta$ **then**
      Add the children of Q to the end of the processing list.

end if
end if
Remove Q from the processing list.
end for
Assign $\eta_{S_{\text{out}}} = \mathbf{HAUS}_\gamma(S_{\text{out}}, D')$.

We next make some observations on the output $\mathcal{T}$ of **MAIN**. We know that each non-flagged terminal node Q of $\mathcal{T}$ is adorned by a local polynomial surface produced by **FIT** whose Hausdorff distance to the data D_Q in Q is at most $\hat{\eta}_Q$. Generally, we have $\hat{\eta}_Q \le \eta$. The only exception to this is if Q is at dyadic level ℓ and the subdivision was stopped because of the user-imposed level restriction. In the case that no leaf cube containing points is flagged or artificially stopped by the condition on the maximal depth of $\mathcal{T}$, then the surface S_{out} will have Hausdorff distance $\eta_{S_{\text{out}}}$ to the point cloud, and $\eta_{S_{\text{out}}} \le \eta^*$, which is the original goal of the algorithm. We have found that with properly denoised data it is almost always the case that $\eta_{S_{\text{out}}}$ is close to the target accuracy η^*. Indeed, a cube containing points is flagged because it has too few points, but it generally has a neighboring cube which is not flagged and therefore has a local polynomial fit on that neighbor. When this is not the case, one could say that the points in the flagged cube are outliers and most of them have been removed by the preprocessing algorithms from section 6.

8. Encoding. We anticipate that one of the main applications of our fitting algorithm will be compression and encoding of the point cloud. The output of **MAIN** is an octree whose nodes are ornated with coefficients of polynomials. This is a common setting in image and surface compression, and there are several approaches to converting such an ornated tree to a bit stream which the user can decode to find the tree and quantized coefficients [25, 27]. For our implementation, we have utilized the predictive encoder developed by the Rice group [31]. Note that here a simple switch of data structures is necessary, since the Rice encoder works on an octree whose nodes are ornated with a single polynomial surface. In our case, the terminal nodes may have bounding boxes and may have two polynomials. We therefore have to encode this additional structure. The compression figures given below include the extra bits needed to encode this extra structure but done currently in a rather naive way. With Rice, we are improving on this encoder to include burn-in and progressivity and will report on this in a forthcoming paper.

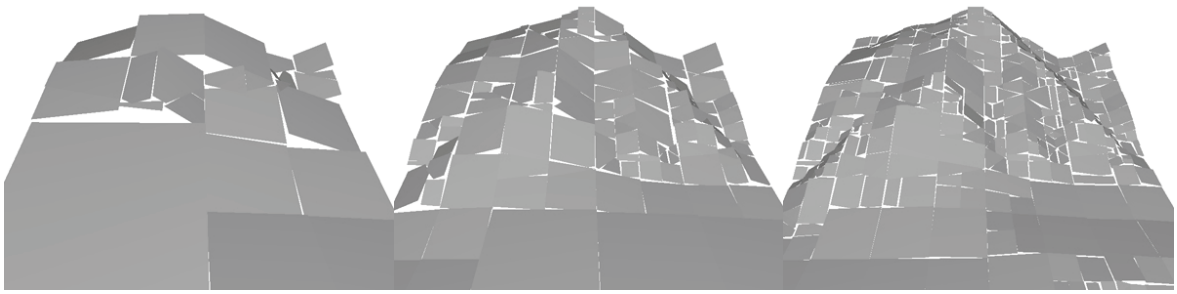
For large terrain data sets the ranges in x and y are typically significantly larger than the range in the vertical direction z. This is the reason we have introduced the integer κ and the initialization step **INIT**. This step shifts the data in the z direction and computes the largest nonnegative integer κ such that $D' \subset [0,1]^2 \times [0, 2^{-\kappa}]$. This fact was not utilized in **MAIN** but will be exploited in the encoding. Namely, if κ is large, then there are a lot of cubes in the tree at levels coarser than or equal to κ which have no data but would be encoded if we began the encoding at the root $[0,1]^3$. For this reason, we will instead encode a forest starting with the occupied cubes at level κ. This will improve the encoding because we remove κ levels from the encoding of the single octree $\mathcal{T}$, including the intermediate fits, and replace those by encoding κ and the root of each of the subtrees. This allows us to restrict the data to at most $2^{2\kappa}$ subtrees whose separate encoding results in higher compression rates. This step generally improves the compression rate over simply starting with $\Omega = [0,1]^3$ as the root of an octree.

The encoding of κ and the subtree roots is not free, and thus some overhead is required. To ensure that this overhead does produce lower overall compression than simply encoding the original octree, we impose a minimum value κ_0 and require $\kappa > \kappa_0$ before the subtrees are considered separately for encoding. In all our experiments, we have $\kappa_0 = 1$.

Next, we give some numerical results which tabulate the rate distortion performance of our algorithm **MAIN** when coupled with the Rice encoder. Note that one can utilize **MAIN** with different Hausdorff target tolerances η^* in order to create a progressive encoder analogous to wavelet-based image encoding; see, for example, [11, 18].

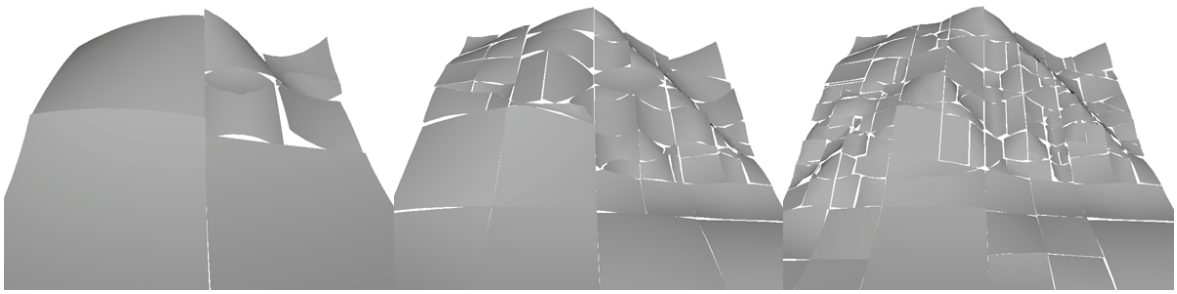
For our experiments, we first denoise all point cloud data sets, using the Outlier Measure algorithm described in section 6.

Test 1. Our first test is the Maple Mountain data set [7], generated by high-altitude terrain scans. The original data contains 62,500 points as 16-bit unsigned integer height values, with a file size of 125,000 bytes.



$\eta^* = 0.05$ $\eta^* = 0.02$ $\eta^* = 0.01$

Figure 8. *Maple Mountain reconstruction using* **PLANE** *for different values of* η^*.



$\eta^* = 0.05$ $\eta^* = 0.02$ $\eta^* = 0.01$

Figure 9. *Maple Mountain reconstruction using* **QUAD** *for different values of* η^*.

In Figures 8 and 9, we display the local plane and quadratic fits, respectively, that ornate the final leaves of the octree $\mathcal{T}$ for various values of the Hausdorff tolerance η^*. One observes the progressive structure of the octree, in that smaller η^* values correspond to more detail on the surface. In Table 2, we show various statistics for the corresponding octrees, such as tree depth, number of nodes, number of polynomial fits, boxes, and clusters.

Table 2

Tree statistics using $\mathbf{CLUSTER}_\gamma$ *= sliding-partition method, with parameters* $\gamma = 2^{-10}$, $N = 10$, $K = 10$, *and* $\ell = 20$.

Maple Mountain	η^*	Tree depth	# Nodes in tree	# Leaf nodes	# Basic fits	# BOX	# CLUSTER
PLANE	0.01	6	817	715	274	46	36
	0.02	6	273	239	98	17	12
	0.05	4	49	43	20	0	3
QUAD	0.01	6	321	281	101	23	23
	0.02	4	97	85	34	5	10
	0.05	3	17	15	9	1	0

In Table 3, we present the Hausdorff error between the points in the nodes of the octree $\mathcal{T}$ and the corresponding fit in these nodes (Hausdorff error on fits), and the Hausdorff error between the whole point cloud D' and the resulting fragmented surface S_{out} (Hausdorff error global).

Table 3

Maple Mountain. The Hausdorff errors and η^* *are given with respect to the unit cube.*

	PLANE		**QUAD**	
η^*	Hausdorff error on fits	Hausdorff error global	Hausdorff error on fits	Hausdorff error global
0.01	0.00997005	0.00997005	0.00987381	0.00987381
0.02	0.01992060	0.01953990	0.01899540	0.01849000
0.05	0.04957710	0.04897010	0.04132170	0.03401330

Notice that in this example, the global Hausdorff error is sometimes smaller than the local Hausdorff error. The reason for this is that the point cloud to surface distance, as well as the surface to point cloud distance, may improve globally by using the portions of the surface on neighboring cubes.

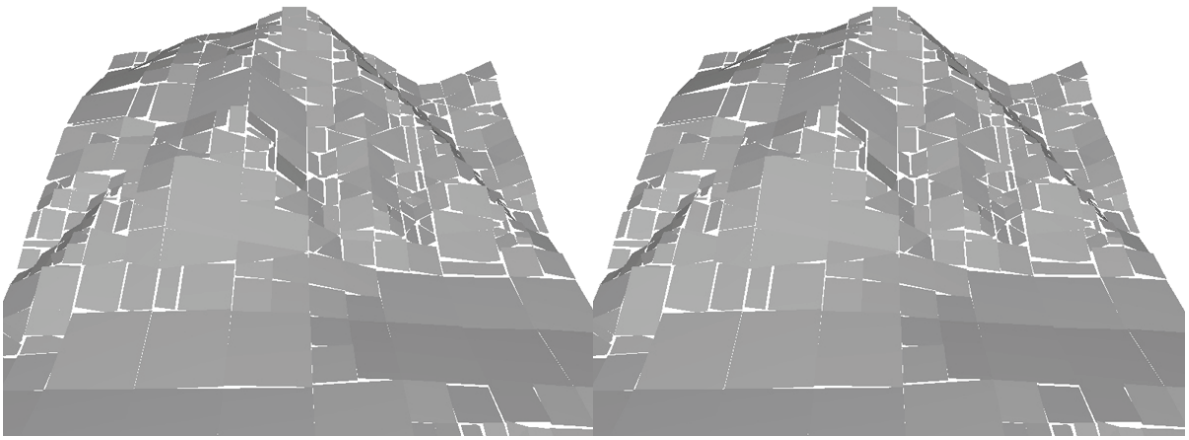
Table 4 contains the compression results obtained from the application of our algorithm and the Rice encoder (with parameters $Smoothness = s$, $WindowSize = 16$, and $TaperLevel = TreeDepth$). The computed Hausdorff error is the global error between the point cloud and the decoded planar fits. The compression ratios in the last two columns are the results reported by the Rice encoder without and with the additional bits needed for the encoding of the data structure switch, respectively.

Table 4

Compression results, Maple Mountain.

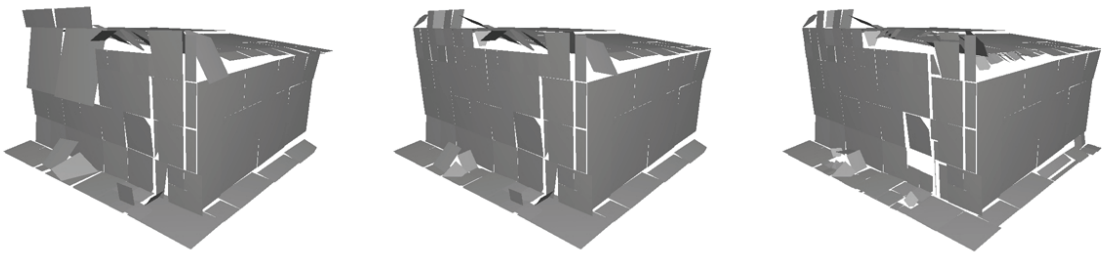
η^*	s	Encoded size (bytes)	Hausdorff error	Comp. ratio (Rice)	Comp. ratio (Rice and extra)
0.01	3	3922.25	0.01209860	44.71072163	31.86946268
0.02	3	1216.25	0.02352490	151.4921982	102.7749229
0.05	4	219	0.04855540	791.1392405	570.7762557

Figure 10 shows a comparison between the surface S_{out}, representing the Maple Mountain and its decoded counterpart, after the Rice encoder compression. Clearly, both Table 4 and Figure 10 demonstrate the high compression rates without sacrificing the quality of the surface.



Original surface Decoded surface

Figure 10. *Maple Mountain,* $\eta^* = 0.01$



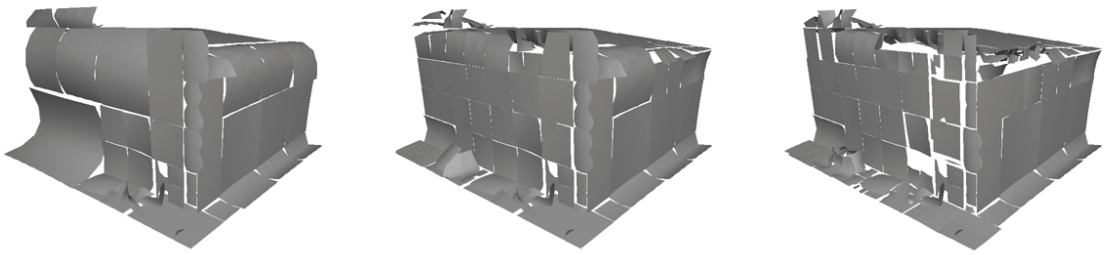
$\eta^* = 0.06$ $\eta^* = 0.04$ $\eta^* = 0.03$

Figure 11. *Building reconstruction using* **PLANE** *for different values of* η^*.

Test 2. Our next data set (see Figures 11–12) is a portion of a building taken from the Eglin data (AFRL/MNG VEAA Data Set #1). It contains typical features present in real world urban terrain, such as windows, doors, corners, thin structures, and bushes. Resolving these structures is a challenge for any reconstruction algorithm. Note that this is a true three-dimensional point cloud data, where each point coordinate is represented by a 32-bit floating point value. The denoised data contains 22,058 points and has a file size of 264,696 bytes.

Tables 5 and 6 describe the statistics and results for the Building data set. Note that, as shown in Table 6, the local Hausdorff error for quadratics could be larger than the error for planes due to the fact that these fits are obtained via L_2 and not Hausdorff optimization. Compression results for this example, as well as for those that follow, are given in Table 15.

Test 3. Next, we process another portion of the Eglin data (see Figures 13–14): a single light pole and a portion of a vehicle (in the upper right corner). This point cloud is a computational challenge since it contains a thin structure with no discernible interior. The denoised data consists of 5,901 points and has a file size of 70,812 bytes.



$\eta^* = 0.06$ $\eta^* = 0.04$ $\eta^* = 0.03$

Figure 12. *Building reconstruction using* **QUAD** *for different values of* η^*.

Table 5

Tree statistics using **CLUSTER** = *sliding-partition method, with parameters* $\gamma = 2^{-10}$, $N = 10$, $K = 10$, *and* $\ell = 20$.

Building	η^*	Tree depth	# Nodes in tree	# Leaf nodes	# Basic fits	# BOX	# CLUSTER
PLANE	0.03	6	521	456	149	25	5
	0.04	5	273	239	103	29	2
	0.06	5	177	155	80	18	2
QUAD	0.03	6	353	309	108	17	10
	0.04	5	201	176	82	13	4
	0.06	5	97	85	49	7	2

Table 6

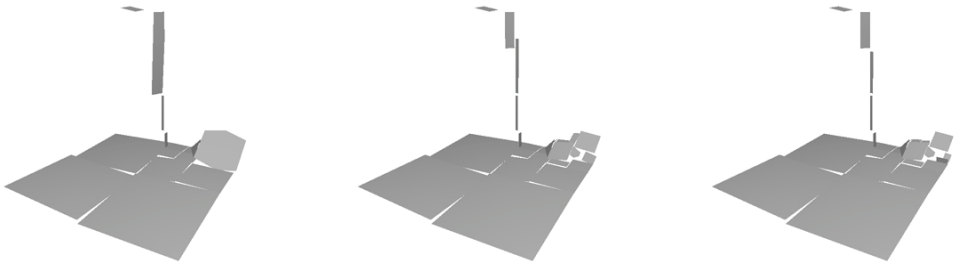
Building. The Hausdorff errors and η^* *are given with respect to the unit cube.*

	PLANE		**QUAD**	
η^*	Hausdorff error on fits	Hausdorff error global	Hausdorff error on fits	Hausdorff error global
0.03	0.02934980	0.03290050	0.02965160	0.02941150
0.04	0.03994440	0.03994440	0.03980440	0.03661180
0.06	0.05418520	0.05013430	0.05835300	0.05835300

Tables 7 and 8 show the octree statistics and Hausdorff errors for the Light Pole data. Notice the good approximation of the original point cloud using a succinct representation.

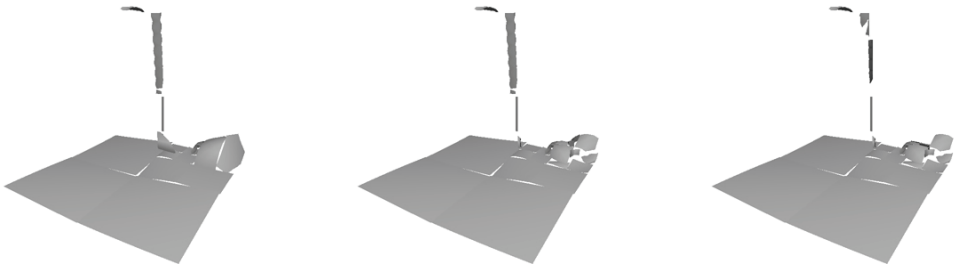
Test 4. We now consider a more complicated data set (see Figures 15–17), obtained using simulated LiDAR [7, 13]. It was created from a simulated low-altitude flight through a CAD representation of an actual Military Operations in Urban Terrain (MOUT) site. We present three pairs of images showing the point cloud and reconstructed surface from different vantage points. We have selected these views to emphasize the various urban terrain structures that are present in the data. The denoised data contains 632,448 points and has a file size of 7,589,376 bytes.

Tables 9 and 10 show the octree statistics and Hausdorff errors for the MOUT data. Let us note that, in this example and some of the ones that follow, the global Hausdorff distance is larger than the local distance. The reason for this is the appearance of flagged cubes which



$\eta^* = 0.08$ $\eta^* = 0.06$ $\eta^* = 0.05$

Figure 13. *Light pole reconstruction using* **PLANE** *for different values of* η^*.



$\eta^* = 0.08$ $\eta^* = 0.06$ $\eta^* = 0.05$

Figure 14. *Light pole reconstruction using* **QUAD** *for different values of* η^*.

Table 7

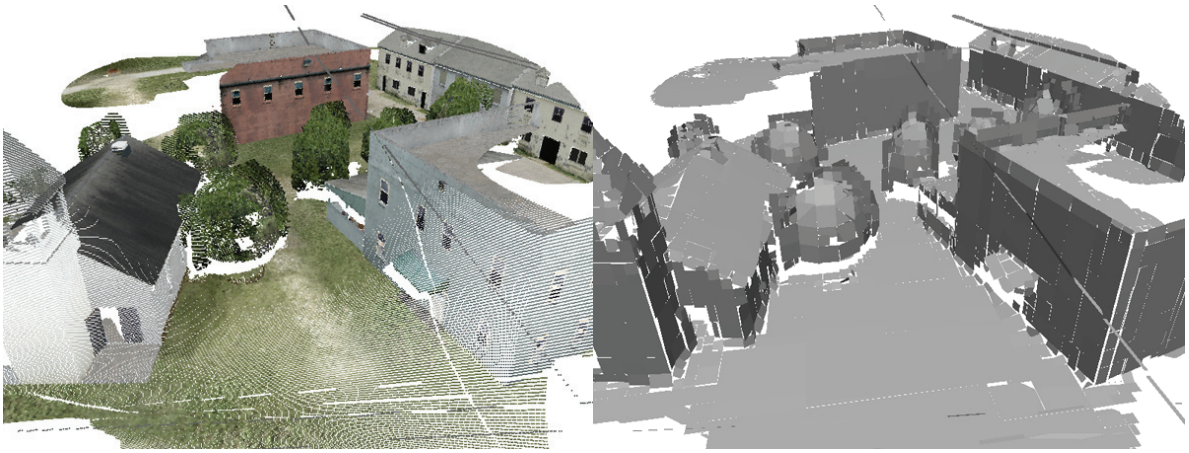
Tree statistics using **CLUSTER** = *sliding-partition method, with parameters* $\gamma = 2^{-10}$, $N = 10$, $K = 10$, *and* $\ell = 20$.

Light Pole	η^*	Tree depth	# Nodes in tree	# Leaf nodes	# Basic fits	# BOX	# CLUSTER
PLANE	0.05	5	49	43	14	9	1
	0.06	4	33	29	13	7	2
	0.08	4	25	22	11	5	0
QUAD	0.05	5	41	36	14	8	1
	0.06	4	33	29	13	8	1
	0.08	4	25	22	11	4	1

Table 8

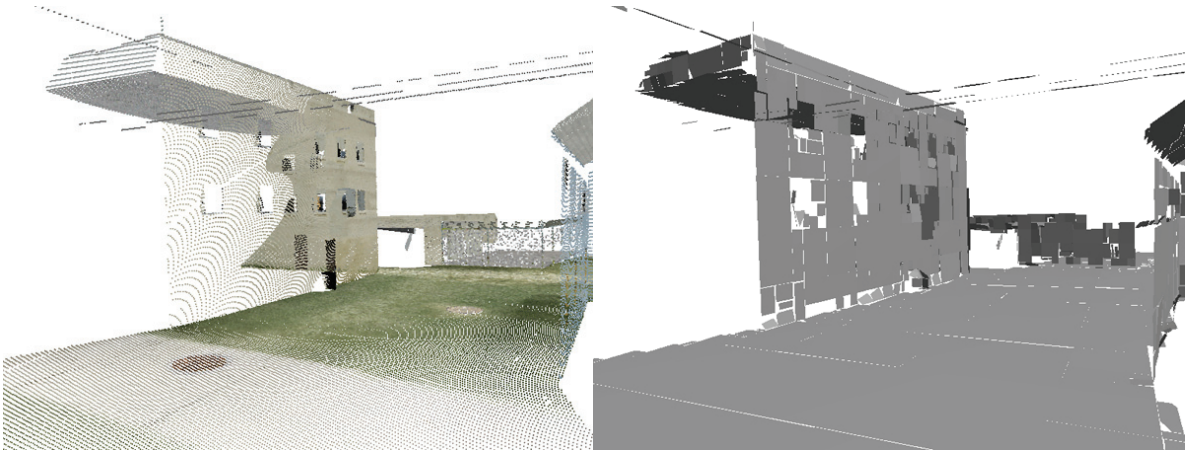
Light Pole. The Hausdorff errors and η^* *are given with respect to the unit cube.*

	PLANE		**QUAD**	
η^*	Hausdorff error on fits	Hausdorff error global	Hausdorff error on fits	Hausdorff error global
0.05	0.04897710	0.04897710	0.04795200	0.04795200
0.06	0.05851370	0.05851370	0.05693050	0.05306850
0.08	0.07074950	0.07074950	0.07045780	0.06543520



Point Cloud Reconstruction using **PLANE**, $\eta^* = 0.002$

Figure 15. *MOUT, View* 1.



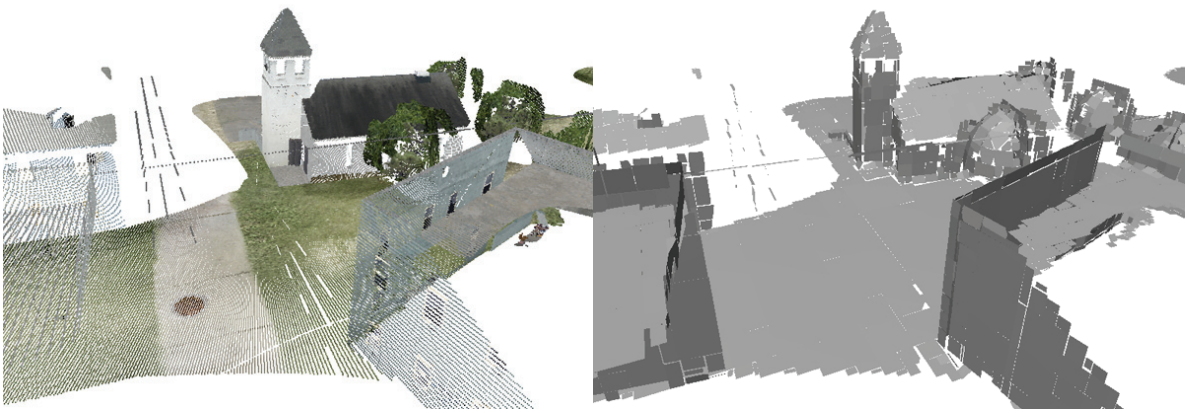
Point Cloud Reconstruction using **PLANE**, $\eta^* = 0.002$

Figure 16. *MOUT, View* 2.

contain points but do not have a local surface fit. The points in these flagged cubes have to be included when computing the global distance but, of course, were not included in the computation of the local distance.

Test 5. We now consider a larger portion of the Eglin data set containing both thin structures and buildings; see Figure 18. The purpose of this test is to demonstrate that no special tuning of the algorithm is needed to handle both thin structures and buildings. The denoised data contains 382,143 points and has a file size of 4,585,716 bytes. Tables 11 and 12 show the octree statistics and Hausdorff errors for the Eglin1 data.

Test 6. We now consider a second portion of the Eglin data set consisting of several portions of multiple buildings; see Figure 19. The denoised data contains 192,021 points and has a file



Point Cloud Reconstruction using **PLANE**, $\eta^* = 0.002$

Figure 17. *MOUT, View* 3.

Table 9

Tree statistics using **CLUSTER** = *sliding-partition method, with parameters* $\gamma = 2^{-12}$, $N = 10$, $K = 10$, *and* $\ell = 20$.

MOUT	η^*	Tree depth	# Nodes in subtrees	# Leaf nodes	# Basic fits	# BOX	# CLUSTER
PLANE	0.002	10	15718	13757	3056	1096	407
	0.004	9	6782	5938	1744	686	156
	0.006	9	4086	3579	1162	437	107
QUAD	0.002	10	14142	12378	2607	938	390
	0.004	9	5478	4797	1209	599	174
	0.006	9	3230	2830	837	385	116

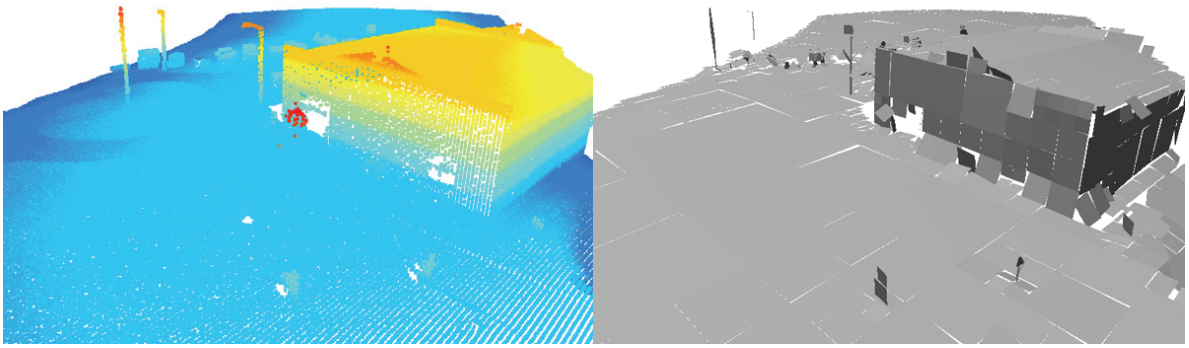
Table 10

MOUT. The Hausdorff errors and η^* *are given with respect to the unit cube.*

	PLANE		**QUAD**	
η^*	Hausdorff error on fits	Hausdorff error global	Hausdorff error on fits	Hausdorff error global
0.002	0.00199971	0.01965250	0.00199971	0.01979920
0.004	0.00399864	0.01402490	0.00399933	0.00796615
0.006	0.00599457	0.01574560	0.00599397	0.00767231

size of 2,304,252 bytes. Tables 13 and 14 show the octree statistics and Hausdorff errors for the Eglin2 data.

Next (see Table 15), we show the compression ratios for the smallest values of η^* for all the above data. The values are produced by the same method used to create Table 4 (with parameters $Smoothness = s$, $WindowSize = 16$, and $TaperLevel = TreeDepth$). One can obtain higher compression ratios if larger values of η^* are used, but this would result in less accurate surfaces. The reported Hausdorff errors in Table 15 are the global Hausdorff errors



Point Cloud Reconstruction using **PLANE**, $\eta^* = 0.006$

Figure 18. *Eglin*1.

Table 11

Tree statistics using **CLUSTER** = *sliding-partition method, with parameters* $\gamma = 2^{-12}$, $N = 10$, $K = 10$, *and* $\ell = 20$.

Eglin1	η^*	Tree depth	# Nodes in subtrees	# Leaf nodes	# Basic fits	# BOX	# CLUSTER
PLANE	0.006	8	2801	2456	678	199	98
	0.008	8	1777	1560	483	139	63
	0.010	8	1201	1056	328	114	47
QUAD	0.006	8	2273	1994	505	184	95
	0.008	8	1457	1280	356	131	64
	0.010	8	977	860	253	101	42

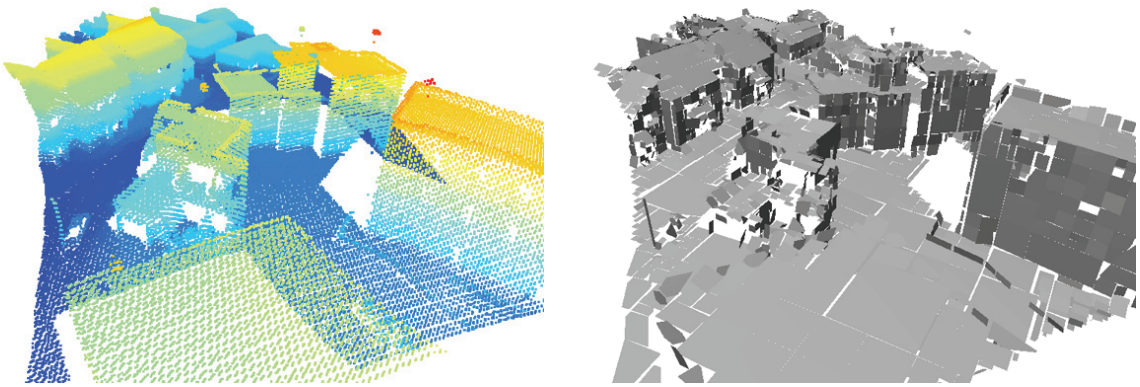
Table 12

*Eglin*1. *The Hausdorff errors and* η^* *are given with respect to the unit cube.*

	PLANE		**QUAD**	
η^*	Hausdorff error on fits	Hausdorff error global	Hausdorff error on fits	Hausdorff error global
0.006	0.00599877	0.01640190	0.00599398	0.01666800
0.008	0.00799094	0.00814113	0.00799883	0.01580340
0.010	0.00996485	0.01065530	0.00997917	0.00997825

between the corresponding point clouds D and the decoded planar surfaces that represent them.

Remark. Note that the local Hausdorff error on a single cube and the global Hausdorff error may be quite different since the first one depends heavily on the structure of the octree. When a cube has been chosen for subdivision, the choice of how to create the child cubes is automatic and does not take completely into account the structure of the point cloud within that cube. This could lead to the introduction of a false noise or outliers in the child cubes. A similar problem occurs when the octree creates nodes that contain too few points for a fit to be computed. The latter often happens when the sampling density of the point cloud varies



Point Cloud

Reconstruction using **PLANE**, $\eta^* = 0.006$

Figure 19. *Eglin*2.

Table 13

Tree statistics using **CLUSTER** = *sliding-partition method, with parameters* $\gamma = 2^{-12}$, $N = 10$, $K = 10$, *and* $\ell = 20$.

Eglin2	η^*	Tree depth	# Nodes in subtrees	# Leaf nodes	# Basic fits	# BOX	# CLUSTER
PLANE	0.006	8	7262	6356	1596	587	208
	0.008	8	4454	3899	1046	435	149
	0.010	8	3038	2660	790	319	116
QUAD	0.006	8	5606	4907	1066	606	221
	0.008	8	3334	2919	688	410	156
	0.010	8	2238	1960	473	304	128

Table 14

*Eglin*2. *The Hausdorff errors and* η^* *are given with respect to the unit cube.*

	PLANE		**QUAD**	
η^*	Hausdorff error on fits	Hausdorff error global	Hausdorff error on fits	Hausdorff error global
0.006	0.00599929	0.04639610	0.00599929	0.04639270
0.008	0.00799705	0.01619260	0.00799947	0.01555480
0.010	0.00998930	0.01619260	0.00999902	0.01555480

substantially from region to region. We illustrate this phenomenon in Figure 20, produced using the MOUT data with $\eta^* = 0.002$. The red planes in this figure correspond to fits for cubes that do not meet the error threshold η^* and, if subdivided further, would have produced all children each with fewer than K points.

9. Implicit surface representations. The surface S_{out}, generated by the output of **MAIN**, and representing the point cloud D, is a piecewise (discontinuous) polynomial surface. Direct display of S_{out} is possible but not appealing (see Figures 8–19 in section 8) due to the large number of discontinuities and missing regions. In this section, we present alternative methods

Table 15
Compression results.

Data	η^*	s	Encoded size (bytes)	Hausdorff error	Comp. ratio (Rice)	Comp. ratio (Rice and extra)
Maple	0.01	3	3922.25	0.01209860	44.71072163	31.86946268
Building	0.03	4	2165.375	0.03149410	146.2913990	122.2402586
Light Pole	0.05	5	317.5	0.05292810	332.2557185	223.0299213
MOUT	0.002	5	85129.625	0.01996640	113.5764328	89.15082147
Eglin1	0.006	6	16675.5	0.01620250	355.3200384	274.9972115
Eglin2	0.006	3	30037	0.04627640	112.6932026	76.71378633

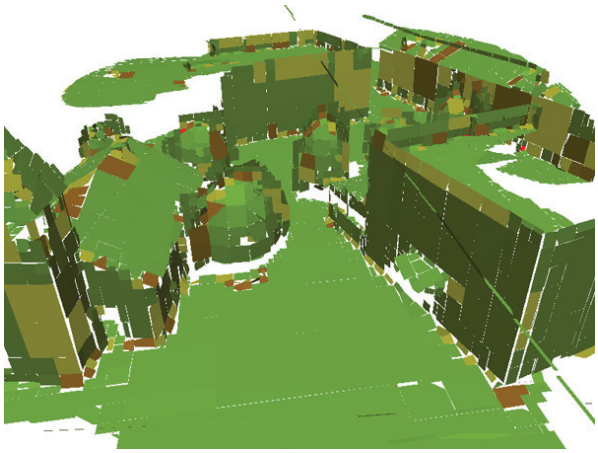


Figure 20. *Planes colored according to their Hausdorff error η^*: green $\eta^* \le 0.001$, yellow $\eta^* \le 0.0015$, brown $\eta^* \le 0.002$, and red $\eta^* > 0.002$.*

that take S_{out} and create a more appealing mathematical surface S_{math}. Our main goal is to retain the fidelity of the representation but to have S_{math} be a smoother connected surface. We shall discuss our method for generating an S_{math} that can be displayed using standard graphics hardware.

The surface S_{math} that we generate is given implicitly as the solution set of $F = h$, where F is a function defined on $[0,1]^3$ and h is a real number. There are many possible choices for such a function F. The typical implicit representations of surfaces use signed or unsigned distance functions. The main drawback of distance methods is that the associated isosurface depends unstably on the parameter h: the resulting surface either lacks detail (if $|h|$ is too big) or has artificial holes (if $|h|$ is too small). For this reason, we shall use an alternative method based on multiscale (wavelet) decompositions, originally developed in [23] for laser scan data. This method views the surface as boundary of a three-dimensional body M. Notice that the surface is then the level set of the indicator function of M, χ_M. The method takes the wavelet decomposition of χ_M and a truncation F of this decomposition. Then S_{math} is a level set of F (typically we use $F = 0.5$ in our algorithms).

The wavelet-based methods require knowledge of the surface orientation (the identification of the inside and outside of the surface) in order to define M. This orientation is generally

not available to us. Determining the surface orientation is a well-studied topic in geometric modeling and is typically derived from the point cloud data directly; see [15, 16, 20]. The existing surface orientation methods are successful when the sensor produces a point cloud that either includes normals, the point cloud has such high resolution that normals can be accurately numerically computed, or the surface to be reconstructed is very smooth. These ingredients are usually not available for terrain point clouds. Therefore, we shall develop a surface orientation algorithm that is based only on the concise tree representation $\mathcal{T}$ and without any additional information. We discuss this orientation algorithm in the following subsection and then discuss how we find F in subsequent sections.

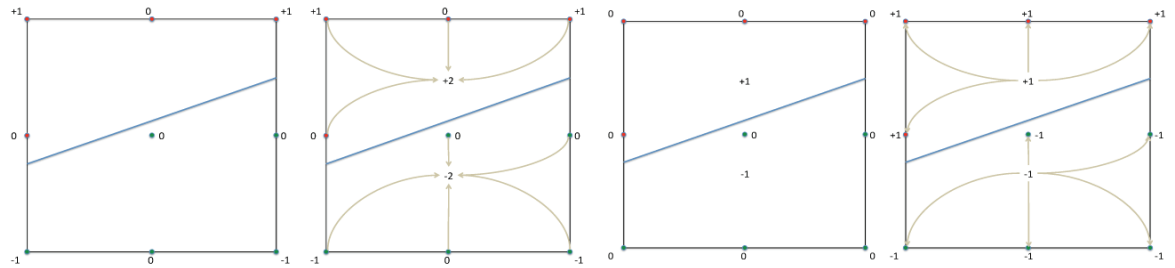
9.1. Identifying the orientation of the terrain surface. In this section, we propose the subroutine **ORIENT** that orients S_{out} without the knowledge of the normals associated to the point cloud data D, or any sensor-related information such as position or orientation. The algorithm uses a progressive coarse to fine prediction and voting scheme that utilizes the local polynomial fits from each level of the octree to determine a globally consistent orientation. The algorithm **ORIENT** has nothing to do with the encoding algorithm of the preceding section. The encoding proceeds without it. We employ **ORIENT** as a postprocessing algorithm only when we want to visualize the output of **MAIN**. We shall describe **ORIENT** only in the case when the polynomial fits are planes. Substantial modifications to the algorithm are needed when the fits are quadratic functions.

We first preprocess the octree $\mathcal{T}$, produced from **MAIN**, using the subroutine **CUT**. This subroutine replaces with flagged nodes the terminal leaves Q of $\mathcal{T}$ that contain planes $\hat{S}_{D_Q}$ obtained from $\mathbf{BOX}_\gamma$ or $\mathbf{CLUSTER}_\gamma$, and outputs the resulting tree $\mathcal{T}_c$ along with the list $\mathcal{N}$ of replaced nodes.

The subroutine **ORIENT** takes the output $\mathcal{T}_c$ from **CUT** and the integer κ from **INIT** and creates an octree $\widetilde{\mathcal{T}}$, that is, $\mathbf{ORIENT}(\mathcal{T}_c, \kappa) = \widetilde{\mathcal{T}}$, which has the same structure as $\mathcal{T}_c$, and in addition has the values -1, 1, or 0 assigned to the vertices of each of its cubes as well as the vertices of each of their children; -1 meaning inside the surface, 1 outside the surface, and 0 on the surface. Note that the children of a cube Q in $\mathcal{T}_c$ are not in $\mathcal{T}_c$ when Q is a terminal leaf, but we will still define a value to the vertices of all these children. Let us denote by V the set of all vertices of all cubes Q such that either Q is in $\mathcal{T}_c$ or Q is a child of a cube in $\mathcal{T}_c$.

Next, we describe how to assign values to the vertices in V as we march through the levels of the octree from coarse to fine. Given an integer $m \geq \kappa$, we denote by $\mathcal{T}_m$ the truncation of $\mathcal{T}_c$ to this level. Given any cube Q we denote by V_Q the set of all vertices in $V \cap Q$ that are either vertices of Q or vertices of one of its children. We say that a vertex has level m if it is a vertex of a cube of dyadic level m (i.e., side length 2^{-m}) but not of a coarser cube. We define V_m to be the set of all vertices in V of level m, and W_m the set of all vertices in V of level $\leq m$. To start our labeling, we take advantage of the fact that $D' \subset [0,1]^2 \times [0, 2^{-\kappa}]$ and assign values $s_\kappa(v) \in \{\pm 1, 0\}$ to all $v \in V$ as follows. We assign -1 to all the vertices $v = (x, y, 0) \in V$ which are of level $\leq \kappa$ and assign $+1$ to all other vertices $v \in V$ of level $\leq \kappa$. All other vertices $v \in V$ are initially assigned the value $s_\kappa(v) = 0$.

Let us now assume that we have assigned values $s_m(v)$ to all vertices $v \in V$ and explain how we determine the updated values $s_{m+1}(v)$. Let Q be any leaf cube from $\mathcal{T}_m$ (Figure 21(a)).



(a) (b) (c) (d)

Figure 21. *Label assignment (two dimensions):* (a) *An arbitrary square Q with vertex values $s_m(v)$.* (b) *The values of the vertices are summed for each group (V_+ and V_-).* (c) $s_{m+1}(v, Q)$ *is found.* (d) $s_{m+1}(v, Q)$ *is redistributed to the vertices. For each vertex v, a second pass over all squares that share v will take place to determine $s_{m+1}(v)$ (not shown).*

Note that this cube can be of any level $\leq m$. If Q is flagged and Q has level q, then for any vertex $v \in W_{q+1} \cap Q$, we define $s_{m+1}(v, Q) := \text{sign}(\sum_{w \in V_Q} s_m(w))$, where $\text{sign}(a)$ is the function that gives the sign of the number a with $\text{sign}(0) = 0$. Note that in this case, all the vertices in V_Q are assigned the same label. If Q is not flagged, then it has a fit $\hat{S}_{D_Q}$ that separates the vertices v of V_Q into two groups, V_+ and V_-, corresponding to the intersection of V_Q with the closed half-spaces generated by the plane $\hat{S}_{D_Q}$. If $v \in V_+$ and does not lie on $\hat{S}_{D_Q}$, then we define $s_{m+1}(v, Q) = \text{sign}(\sum_{w \in V_+} s_m(w))$ (Figures 21(b) and 21(c)). We make the corresponding assignment when $v \in V_-$ and does not lie on $\hat{S}_{D_Q}$. If $v \in \hat{S}_{D_Q}$, we assign the value $s_{m+1}(v, Q) = 0$.

Finally, we shall assign the value $s_{m+1}(v)$. Given any vertex $v \in V_Q$ of a cube Q of dyadic level $\leq m$, we define $s_{m+1}(v) := \text{sign}(\sum_{R \in \mathcal{T}_m | v \in V_R} s_{m+1}(v, R))$ (Figure 21(d)). For any vertex $v \in V$ whose value has not been updated, we define $s_{m+1}(v) := s_m(v)$. The entire process halts when we have $m = \ell$, the finest allowable level. In this case, we define the final values $s(v) := s_{\ell+1}(v)$ for all $v \in V$.

The subroutine **ORIENT**, performing the above-mentioned procedure, is as follows:

for Each vertex $v = (x, y, z) \in V$ **do**
  if Level of $v \leq \kappa$ **then**
    if $z > 0$ **then**
      $s_\kappa(v) = 1$.
    else
      $s_\kappa(v) = -1$.
    end if
  else
    $s_\kappa(v) = 0$.
  end if
end for
for $m = \kappa \ldots \ell$ **do**
  for Each $v \in V$ **do**
    $s_{m+1}(v) = s_m(v)$

end for
for Each leaf cube Q of $\mathcal{T}_m$ **do**
if Q is flagged **then**
for Each vertex $v \in V_Q$ **do**

$$s_{m+1}(v, Q) = \text{sign}\left(\sum_{w \in V_Q} s_m(w)\right).$$

end for
else
Find V_+ and V_-.
for Each vertex $v \in V_Q$ **do**
if $v \in \hat{S}_{D_Q}$ **then**
$s_{m+1}(v, Q) = 0.$
else if $v \in V_+$ **then**

$$s_{m+1}(v, Q) = \text{sign}\left(\sum_{w \in V_+} s_m(w)\right).$$

else

$$s_{m+1}(v, Q) = \text{sign}\left(\sum_{w \in V_-} s_m(w)\right).$$

end if
end for
end if
end for
for Each vertex $v \in W_{m+1}$ **do**

$$s_{m+1}(v) = \text{sign}\left(\sum_{R \in \mathcal{T}_m | v \in V_R} s_{m+1}(v, R)\right).$$

end for
end for
for Each $v \in V$ **do**
$s(v) = s_{\ell+1}(v)$
end for

9.2. The solid region M. In this section, we shall describe how we generate a solid M from the output of **MAIN**. Recall that the role of M is that it gives a solid whose boundary is the surface which represents the data. Given the tree $\mathcal{T}$ which is part of the output of **MAIN**, we apply **ORIENT** and receive a labeling of the vertices in V. If Q is a terminal cube of $\mathcal{T}$, we now describe how we define the portion $M_Q = M \cap Q$ of M on Q. The cube Q is one of three types. If Q is not flagged in $\mathcal{T}_c$, then Q has a linear fit $\hat{S}_{D_Q}$ on Q which separates Q into two regions Q', Q''. We examine the $s(v)$, $v \in Q' \cap V$, and determine that Q' is in M if $\text{sign}(\sum_{v \in Q' \cap V} s(v)) \leq 0$; otherwise Q' is outside M. We do the same test for Q''. We then define M_Q as the union of the regions that are inside. Typically, only one of Q', Q'' is inside M, but it could happen that both are labeled inside or both are labeled outside. The second

possible case is that Q is flagged in $\mathcal{T}$ because it had too few data points. In this case, if $\text{sign}(\sum_{v\in V_Q} s(v)) \le 0$, we define $M_Q := Q$; otherwise $M_Q := \emptyset$. The third and last possibility is that Q is flagged in $\mathcal{T}_c$ because it has bounding boxes (more precisely, Q is an element of the output $\mathcal{N}$ from **CUT**). In this case, we define M_{B_Q} as the union of the $\hat{S}_{B_Q}$ that live in Q (there will be one or two of these). We then define M_Q as the union of the M_{B_Q} associated to Q. This method for defining M_Q in the case of bounding boxes tends to preserve thin (lower dimensional) structures such as guy-wires, cables, poles, and fences.

Note that since the point clouds we work with may be incomplete, contain holes or occlusions, and possibly be of lower dimension, finding a globally consistent orientation is challenging and sometimes impossible. However, our algorithm, when tested on various point cloud data, demonstrates robustness and typically produces reasonable solutions to quite complicated data.

9.3. Wavelet display. In this section, we discuss the wavelet method for creating S_{math}. First, we introduce some basic facts about wavelets and refer the reader to [12] for details on this topic. We use the standard construction of three-dimensional wavelet bases and the notation $\psi^0 = \varphi$ for the scaling function φ, $\psi^1 = \psi$ for the corresponding wavelet, E' for the set of vertices of the cube $[0,1]^3$, and E for the set of vertices excluding the origin (i.e., $E = E' \setminus \{(0,0,0)\}$). For each $e = (e_1, e_2, e_3) \in E'$, $j \in \mathbb{N}$, and $\mathbf{k} = (k_1, k_2, k_3)$, we define the ($L_2$ normalized) wavelet

$$\psi^e_{j,\mathbf{k}}(\mathbf{x}) = 2^{3j/2}\psi^{e_1}(2^j x - k_1)\psi^{e_2}(2^j y - k_2)\psi^{e_3}(2^j z - k_3).$$

It is known that every locally integrable function f has the expansion

$$f(\mathbf{x}) = \sum_{\mathbf{k}\in\mathbb{Z}^3} c^{(0,0,0)}_{0,\mathbf{k}}\psi^{(0,0,0)}_{0,\mathbf{k}}(\mathbf{x}) + \sum_{j\in\mathbb{N}}\sum_{\mathbf{k}\in\mathbb{Z}^3}\sum_{e\in E} c^e_{j,\mathbf{k}}\psi^e_{j,\mathbf{k}}(\mathbf{x}), \tag{9.1}$$

with coefficients

$$c^e_{j,\mathbf{k}} = \int_{\mathbb{R}^3} f(\mathbf{x})\psi^e_{j,\mathbf{k}}(\mathbf{x})\,d\mathbf{x}.$$

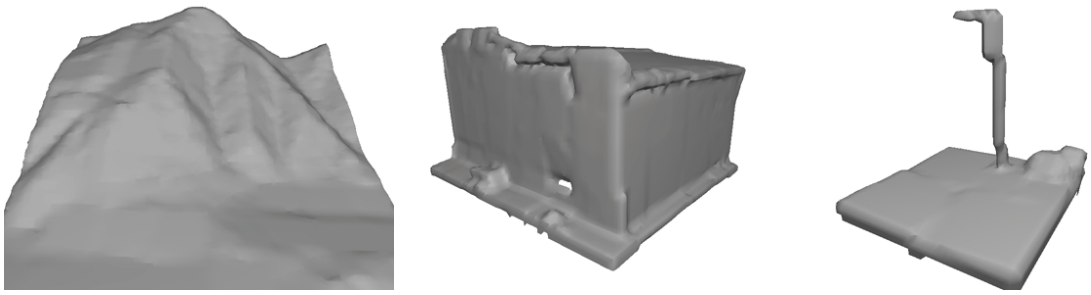
In numerical implementation, one has to take a finite portion of the wavelet sum appearing in (9.1) by retaining only the terms corresponding to $0 \le j \le d$. Here d is a user-defined parameter, but in our numerical experiments we have always chosen d to be the same as the finest level of the octree $\mathcal{T}$. We choose a smoothing parameter t (needed to "inflate" thin structures in order to apply the algorithm from [24]), which we take as 2^{-d} in all of our experiments, and $M(t)$ as the set of all points whose signed distance from M is t. We then construct the wavelet expansion of the function $f = \chi_{M(t)}$ and F_d, which is the partial sum of (9.1) obtained by selecting only those summands for which $j \le d$. We output and display the surface S_d, which is the level set $F_d = 0.5$.

Several computational issues arise in implementing the above display method. The first is to find a numerical approximation to $M(t)$. One could use computationally efficient octree-based algorithms for this, but we decided to take a fine voxelization and define a piecewise constant distance function whose value at each voxel is the signed distance of the center of

the voxel to M. We then subtract t from this distance function and use marching cubes to find the zero set. Next, we need to compute the wavelet coefficients

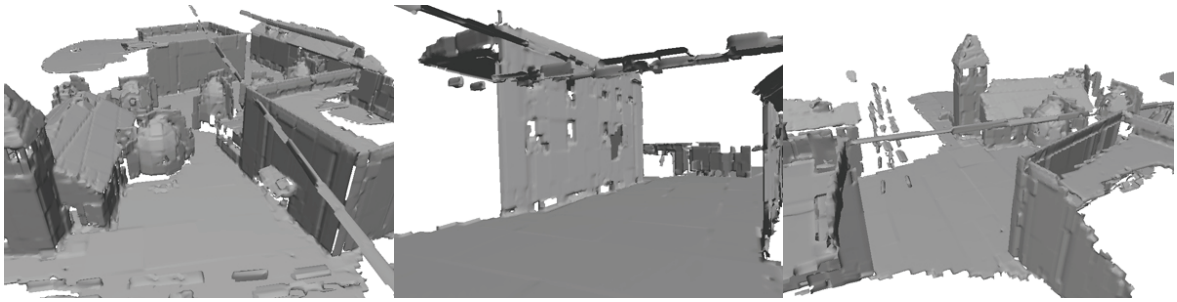
$$2^{3j/2} \int_{M(t_d)} \psi^{e_1}(2^j x - k_1)\psi^{e_2}(2^j y - k_2)\psi^{e_3}(2^j z - k_3)\, dxdydz \tag{9.2}$$

of the series (9.1). In this paper, we use the idea from [23] to compute the volume integral (9.2) as a surface integral using the divergence theorem. Another computational issue is how to find the level set $F_d = 0.5$. For this we use the method described in [24]. In this way we obtain our numerical display of the level set S_d as shown in Figures 22–24 that follow.



Maple Mountain Building Light Pole

Figure 22. *Wavelet processed decoded surfaces.*

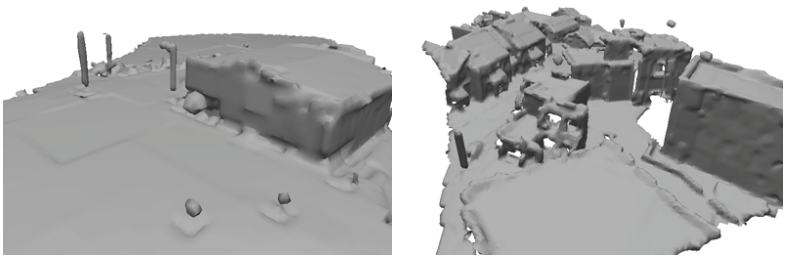


View 1 View 2 View 3

Figure 23. *Wavelet processed decoded surfaces from the MOUT data.*

Note that since the processed point cloud D comes from terrain that may contain holes or occlusions, or may simply be incomplete, we remove the portions of S_d that are unreliable and have been filled in by our algorithm in order to create a globally consistent surface. To perform this operation, we introduce the subroutine **TRIM** that simply removes portions from S_d that are further than $2\eta^*$ from all fits $\hat{S}_{D_Q}$ in the leaf nodes of $\mathcal{T}$. The images that follow show the resulting surfaces for the data sets in section 8 when using the Haar wavelets for ψ.

10. Concluding remarks. An algorithm for a succinct representation of terrain/urban point cloud data is given. This algorithm is based on using local polynomial fits to the



Eglin 1 Eglin 2

Figure 24. *Wavelet processed decoded surfaces from the Eglin data.*

data and measures distortion in the Hausdorff metric. While there are now many methods for processing point cloud data, some using multiscale piecewise polynomial decompositions, these algorithms have not been directed to terrain/urban data, which offers unique challenges due to the irregular sampling, occlusions, and significant noise. The paper offers ad hoc solutions to some of these challenges.

Some applications of our algorithm to the problems of encoding, compression, and visualization are presented. Several examples of rate distortion performance and display are given and can serve as a benchmark for other researchers.

REFERENCES

[1] *Geometric Tools: Wild Magic* 5.9, http://www.geometrictools.com (5 August 2012).

[2] *CGAL, Computational Geometry Algorithms Library*, http://www.cgal.org (24 October 2012).

[3] H. Alt and L. Guibas, *Discrete geometric shapes: Matching, interpolation, and approximation*, in Handbook of Computational Geometry, North–Holland, Amsterdam, 2000, pp. 121–153.

[4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999.

[5] D. Arthur and S. Vassilvitskii, `k-means++`: *The advantages of careful seeding*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2007, pp. 1027–1035.

[6] N. Aspert, D. Santa-Cruz, and T. Ebrahimi, *Mesh: Measuring errors between surfaces using the Hausdorff distance*, in Proceedings of the IEEE International Conference on Multimedia and Expo, Vol. 1, 2002, pp. 705–708.

[7] R. Baraniuk, S. Kulkarni, A. Kurdila, S. Osher, G. Petrova, R. Sharpley, R. Tsai, and H. Zhao, *Model Classes, Approximation, and Metrics for Dynamic Processing of Urban Terrain Data*, MURI annual report, Award W911NF-07-1-0185, U.S. Army Research Office, Research Triangle Park, NC, 2010.

[8] K. Biggers, C. Wells, and G. Williams, *Breaking the Fog: Defining and Orienting Surfaces in Complex Point Cloud Datasets*, Technical report, Texas A&M University, College Station, TX, 2009.

[9] P. Binev, A. Cohen, W. Dahmen, and R. DeVore, *Universal piecewise polynomial estimators for machine learning*, in Curves and Surface Design, Avignon 2006, A. Cohen, J. L. Merrien, and L. Shumaker, eds., Nashboro Press, Brentwood, TN, 2007, pp. 48–78.

[10] P. Binev, A. Cohen, W. Dahmen, V. Temlyakov, and R. DeVore, *Universal algorithms for learning theory.* I. *Piecewise constant functions*, J. Mach. Learn., 6 (2005), pp. 1297–1321.

[11] A. Cohen, W. Dahmen, I. Daubechies, and R. DeVore, *Tree approximation and encoding*, Appl. Comput. Harmon. Anal., 11 (2001), pp. 192–226.

[12] I. Daubechies, *Ten Lectures on Wavelets*, CBMS-NSF Regional Conf. Ser. in Appl. Math. 61, SIAM, Philadelphia, 1992.
[13] T. DeMarse, R. DeVore, W. Ditto, M. Furman, P. Ifju, T. Kanade, P. Khargonekar, A. Kurdila, R. Lind, M. Nechyba, and R. Sharpley, *Vision-based Control of Agile, Autonomous Micro Air Vehicles and Small UAVs in Urban Environments*, MURI final report, Award F49620-03-1-0170, U.S. Air Force Research Laboratory, Wright-Patterson Air Force Base, OH, 2007.
[14] M. Guthe, P. Borodin, and R. Klein, *Fast and accurate Hausdorff distance calculation between meshes*, J. WSCG, 13 (2005), pp. 41–48.
[15] H. Hoppe, T. DeRose, T. DuChamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle, *Piecewise smooth surface reconstruction*, in SIGGRAPH '94, Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, 1994, pp. 295–302.
[16] H. Hoppe, T. Derose, T. Duchamp, J. Mcdonald, and W. Stuetzle, *Surface reconstruction from unorganized points*, SIGGRAPH Comput. Graph., 26 (1992), pp. 71–78.
[17] B. Jawerth, B. Lucier, and R. DeVore, *Surface compression*, Comput. Aided Geom. Design, 9 (1992), pp. 219–239.
[18] L. Johnson, C. Pan, R. Sharpley, and R. DeVore, *Optimal entropy encoders for mining multiply resolved data*, in Data Mining II, N. Ebecken and C. A. Brebbia, eds., WIT Press, Boston, 2000, pp. 73–82.
[19] H. Lee, M. Desbrun, and P. Schröder, *Progressive encoding of complex isosurfaces*, ACM Trans. Graph., 22 (2003), pp. 471–476.
[20] J. Liang, F. Park, and H. Zhao, *Robust and efficient implicit surface reconstruction of point clouds based on convexified image segmentation*, submitted.
[21] Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer, *Parameterization-free projection for geometry reconstruction*, ACM Trans. Graph., 26 (2007), 22.
[22] J. MacQueen, *Some methods for classification and analysis of multivariate observations*, in Proceedings of the Fifth Berkley Symposium on Mathematical Statistics and Probability, Vol. 1, 1967, pp. 281–297.
[23] J. Manson, G. Petrova, and S. Schaefer, *Streaming surface reconstruction using wavelets*, Computer Graphics Forum, 27 (2008), pp. 1411–1420.
[24] J. Manson and S. Schaefer, *Wavelet rasterization*, Computer Graphics Forum, 30 (2011), pp. 395–404.
[25] S. Park and S. Lee, *Multiscale representation and compression of* 3*-D point data*, IEEE Trans. Multimedia, 11 (2009), pp. 177–182.
[26] Z. Qing, Z. Yeting, and L. Fengchun, *Three-dimensional TIN algorithm for digital terrain modeling*, Geo-spatial Information Science, 11 (2008), pp. 79–85.
[27] J. Smith, G. Petrova, and S. Schaefer, *Progressive encoding and compression of surfaces generated from point cloud data*, Comput. Graph., 36 (2012), pp. 341–348.
[28] A. Solé, V. Caselles, G. Sapiro, and F. Aràndiga, *Morse description and geometric encoding of digital elevation maps*, IEEE Trans. Image Process., 13 (2004), pp. 1245–1262.
[29] M. Tang, M. Lee, and Y. J. Kim, *Interactive Hausdorff distance computation for general polygonal models*, ACM Trans. Graph., 28 (2009), pp. 1–9.
[30] A. Thies, B. Philips, P. Binev, R. DeVore, M. Hielsberg, L. Johnson, B. Karaivanov, B. Lane, and R. Sharpley, *Smooth, Piecewise-Polynomial Terrain Representation Using Nontraditional Metrics*, STTR final report, Schafer Corporation Contract W911NF-04-C-0060, U.S. Army Research Office, Research Triangle Park, NC, 2005.
[31] A. Waters and R. Baraniuk, *Multiscale Point Cloud Representation and Compression Hausdorff Distortion*, preprint.
[32] T. Weyrich, M. Pauly, R. Keiser, S. Heinzle, S. Scandella, and M. Gross, *Post-processing of scanned* 3*D surface data*, in Proceedings of the Eurographics Symposium on Point-Based Graphics, Zurich, Switzerland, 2004, pp. 85–94.