Point-based computing on scanned terrain with LidarViewer

Oliver Kreylos1,2,* , Michael Oskin1,3, Eric Cowgill1,3, Peter Gold3, Austin Elliott3, and Louise Kellogg1,3
1W.M. Keck Center for Active Visualization in the Earth Sciences, University of California, Davis, California 95616, USA
2Institute for Data Analysis and Visualization, Department of Computer Science, University of California, Davis, California 95616, USA
3Department of Geology, University of California, Davis, California 95616, USA

ABSTRACT

As an alternative to grid-based approaches, point-based computing offers access to the full information stored in unstructured point clouds derived from lidar scans of terrain. By employing appropriate hierarchical data structures and algorithms for out-of-core processing and view-dependent rendering, it is feasible to visualize and analyze three-dimensional (3D) lidar point-cloud data sets of arbitrary sizes in real time. Here, we describe LidarViewer, an implementation of point-based computing developed at the University of California (UC), Davis, W.M. Keck Center for Active Visualization in the Earth Sciences (KeckCAVES). Specifically, we show how point-based techniques can be used to simulate hillshading of a continuous terrain surface by computing local, point-centered tangent plane directions in a pre-processing step. Lidar scans can be analyzed interactively by extracting features using a selection brush. We present examples including measurement of bedding and fault surfaces and manual extraction of 3D features such as vegetation. Point-based computing approaches can offer significant advantages over grids, including analysis of arbitrarily large data sets, scale- and direction-independent analysis and feature extraction, point-based feature- and time-series comparison, and opportunities to develop semi-automated point filtering algorithms. Because LidarViewer is open-source, and its key computational framework is exposed via a Python interface, it provides ample opportunities to develop novel point-based computation methods for lidar data.

INTRODUCTION

Raw lidar data are a dense cloud of distributed point measurements rich in information on the nature of imaged terrain and its cover of vegetation or anthropogenic structures. The complex, irregular structure of these large data sets can hinder analysis and visualization. Preprocessing tools such as TerraScan (http://www.terrasolid.fi/en/products/terrascan), and commercial point-cloud rendering software such as Quick Terrain Modeler (http://www.appliedimagery.com/qtmmain.htm) can draw and perform specialized analysis of point clouds. However, commonly employed, generalized analysis tools, such as MatLab and ArcGIS, have not been designed to operate interactively on very large (>> memory capacity) unstructured point data sets. Thus, it is standard practice to model point data as structured (usually square-celled) topographic grids and apply computationally efficient grid-based analysis techniques. Such digital elevation models (DEMs) may be derived via spatial averaging of many surrounding points, or by interpolation through an intermediate mesh representation, such as a triangulated irregular network (TIN). Note that the unstructured nature of TINs is very similar to point clouds, and thus the same computational issues apply to very large TIN data sets as to point clouds. Compared to TINs, point-based approaches are simpler and more storage-efficient because they do not require an explicit representation of triangle connectivity. Established grid-based surface analysis tools are widely applied to compute local derivatives of the terrain, such as slope, aspect, and curvature. Visualization of hillshaded terrain data for mapping applications; fitting of geometric primitives such as lines, planes, or curves; and calculating higher-order geomorphic parameters such as flow accumulation all depend critically on these derivatives, which, unfortunately, may be significantly degraded by the process of gridding unstructured point data. For example, flow directions, and thus local channel slope, must fall along eight cardinal directions on a rectangular grid. Steep features, such as fault scarps, will form jagged steps in a grid. While griddimming and other aggregation techniques can overcome some of these issues, the root of the problem has been a lack of computationally efficient mechanisms for visualizing and analyzing very large point-cloud data sets directly.

Point-based computing offers an alternative to the grid paradigm, one that can work directly with raw lidar point-cloud data, circumventing grids or other data models, and is well suited to visualization and interactive analysis. The power of the point-based approach is that it exposes the raw data to analysis without filtering through a model to produce a grid or other continuous representation. Point-based visualization is closely related to TIN visualization in that the raw points are displayed to the user. The point-based approach differs in that at sufficient point density, no connections between vertices are required to give the illusion of a continuous surface. Visualizing just points also preserves the three-dimensional structure of the point cloud, such as that which results from scanning of vegetated surfaces. This software employs three key technologies, described in detail in Section 3: (1) pre-processing of the lidar point cloud into an efficient, octree-based hierarchical file format, (2) view-dependent rendering of the point cloud directly from the octree file, and (3) a two-level caching strategy keeping currently used parts of octree files in the computer’s main memory and graphics memory as the user navigates through the data. Through these approaches, LidarViewer is capable of visualizing and of user interaction with arbitrarily large data sets on commodity computer hardware (desktop or laptop with Intel Core-i5/7 or equivalent AMD CPU, at least 2 GB of RAM, and a discrete Nvidia GeForce GPU, G80 or newer, or equivalent hardware).
AMD/ATI GPU). A powerful advantage of the octree structure is data locality, which enables efficient computation of local properties such as derivatives. We show the power of this technique through real-time hillshade visualization of the point cloud, which also illustrates how point-based rendering can simulate a continuous terrain surface while simultaneously providing access to individual points for measurement and feature extraction (Fig. 1; Animation 1). We also show several geologic applications where such direct interaction with the raw point cloud yields insights that otherwise could not be obtained easily from gridded data.

**BACKGROUND**

In general, the vast majority of work in the geosciences that uses lidar data for quantitative topographic analysis relies upon a gridded representation (DEM) of the data, rather than the original scattered point-cloud measurements. For example, DEMs are widely used to study active faulting (e.g., Cowgill et al., 2012; Cunningham et al., 2006; Engelkemeir and Khan, 2008; Frankel et al., 2007a, 2007b, 2011; Gold et al., 2009; Haugerud et al., 2003; Hudnut et al., 2002; Johnson et al., 2004; Oskin et al., 2007). They have also been used to quantify the surface roughness of faults (Jones et al., 2009; Pearce et al., 2011; Renard et al., 2006), patterns on alluvial fans of debris-flow deposition (Staley et al., 2006) and surface roughness (Frankel and Dolan, 2007), stream (Snyder, 2009) and landslide morphology (McKean and Roering, 2004), and topographic characteristics thought to reflect off-fault damage (Wechsler et al., 2009).

In contrast, only a few geoscience studies have employed point-based computing approaches. Kovač and Žalik (2010) describe an approach to point-based rendering of lidar data that is similar to that described here and in Kreylos et al. (2008). Their method works directly off raw LAS data files, and therefore has significantly reduced preprocessing times. On the other hand, their approach does not offer support for efficient point-based computing besides rendering, nor does it provide measurement or analysis capabilities. Rychkov et al. (2012) presented a C++/Python library to support terrain surface analysis through handling of multi-gigabyte point clouds in operations such as decimation, compression, DEM differencing, and roughness analysis. Gold et al. (2012) used the point-based tools in LidarViewer to analyze terrestrial lidar scans at two sites along the 1954 Dixie Valley earthquake rupture in central Nevada, leading to the first constraints on the full 3D slip vector across this structurally complex fault zone. Glenn et al. (2006) computed slope and roughness from point cloud data to characterize landslides, while Jones et al. (2009) and Pearce et al. (2011) computed topographic and curvature properties to characterize natural faults. Wawrzyniec et al. (2007) developed a spot comparison method to quantify episodic, subcentimeter change due to erosion and deposition in a semi-arid landscape. To compute landslide displacement fields, Teza et al. (2007) developed a piecewise application of the iterative closest point algorithm of Besl and

![Figure 1. View of an airborne lidar survey of the surface rupture of the 4 April 2010 Mw 7.2 El Mayor-Cucapah earthquake on the Borrego fault, drawn from the full survey data set containing 3.7 billion points. View is along the fault zone, cutting through the northern Sierra Cucapah viewed from the north (32.4971°N, 115.6306°W), looking south. The white arrows indicate the newly exposed fault surface. This figure shows part of the complex array of fault scarps that were the subject of intensive mapping efforts immediately after the earthquake. (A) Points are colored by return intensity alone. While vegetation can be identified immediately, the fault scarp is hardly visible. (B) Same view as above, but rendered using simulated hillshading from a dynamic, arbitrary light source. (i.e., from the northeast here). The fault scarp can be easily identified, as can the topographic details of features it cuts.](image-url)
Immediate neighbors of a point hi,j in the interior reduces the amount of stored data by two-thirds. For example, the four positions of sample points are not implied by their order due to differences in terrain elevation, aircraft altitude and attitude, changes in scanning velocity as the laser turns around at either edge of a swath, and other factors. Worse, any semblance of regularity is lost when two or more partially overlapping swaths are combined to cover a larger ground area. Therefore, raw lidar data require explicit storage of the horizontal position of each point, and thus three times more space than storing a regular grid of the same overall point density.

Scattered data in general, and raw lidar data in particular, do not have implicit incidence relationships between points. For example, to find the data point hi closest to a query position q, one has to iterate through all h_i, calculate their distances to q, and then set c to the value i for which the distance is smallest. This naïve procedure is fundamentally inefficient, as it requires computation time proportional to the number of sample points, N. Therefore, the ability to directly use scattered data such as raw lidar data for processing, visualization, and analysis requires data structures and algorithms that minimize the impact of scattered data’s larger storage footprint, and most importantly, allow efficient calculation of incidence relationships (such as k-nearest neighbors) between points.

A HIERARCHICAL APPROACH TO SCATTERED POINT DATA

One consequence of the nature of scattered data is that the order of points in a data set does not matter—because ordinal position does not imply spatial position, permutation of the points in a data set yields the same data set. Most efficient data structures exploit this liberty by reordering a given data set in such a way that points with close spatial positions have close ordinal positions. This results in better support for local processing because all points required to compute a local filter are in the same portion of the data set. However, observational data used to construct terrain models typically does not arrive in the form of regular grids, but in the form of scattered data, i.e., as a set of 3D sample points of the form \( \{ \mathbf{h}_i \mid 0 \leq i < N \} \). This is particularly true for terrestrial lidar data, but even in airborne lidar data, which scans a swath of terrain in a regular zigzag pattern perpendicular to the aircraft flight direction, up to ~20° from nadir, the horizontal positions of sample points are not implied by their order due to differences in terrain elevation, aircraft altitude and attitude, changes in scanning velocity as the laser turns around at either edge of a swath, and other factors. Worse, any semblance of regularity is lost when two or more partially overlapping swaths are combined to cover a larger ground area. Therefore, raw lidar data require explicit storage of the horizontal position of each point, and thus three times more space than storing a regular grid of the same overall point density.

Scattered data in general, and raw lidar data in particular, do not have implicit incidence relationships between points. For example, to find the data point hi closest to a query position q, one has to iterate through all hi, calculate their distances to q, and then set c to the value i for which the distance is smallest. This naïve procedure is fundamentally inefficient, as it requires computation time proportional to the number of sample points, N. Therefore, the ability to directly use scattered data such as raw lidar data for processing, visualization, and analysis requires data structures and algorithms that minimize the impact of scattered data’s larger storage footprint, and most importantly, allow efficient calculation of incidence relationships (such as k-nearest neighbors) between points.

A HIERARCHICAL APPROACH TO SCATTERED POINT DATA

One consequence of the nature of scattered data is that the order of points in a data set does not matter—because ordinal position does not imply spatial position, permutation of the points in a data set yields the same data set. Most efficient data structures exploit this liberty by reordering a given data set in such a way that points with close spatial positions have close ordinal positions. This results in better support for local processing because all points required to compute a local filter are in the same portion of the data set, and can lead to highly efficient ways to find nearest neighbors. One such data structure is a point two-dimensional (2D) tree, better known as quadtree, for 2D data, and octree, for 3D data (Samet, 1990). LidarViewer is based on an octree data structure, but, for simplicity, we first explain the structure of a quadtree, and then its straightforward extension to an octree.

Preprocessing

A quadtree organizes 2D points in a hierarchy of boxes, where each box can only contain up to a fixed number k of points. The algorithm to create this hierarchy is straightforward (Fig. 2). One starts with a single candidate box containing all N points in the data set. This box is typically chosen as the smallest square containing all points, and following computer science nomenclature, is called the root. The rest of the algorithm is recursive: for every candidate box B, calculate N_B, the number of points it contains. If N_B ≤ k, call B a leaf node, and go to the next candidate box. If N_B > k, call B an interior node, and split it into four identically sized new candidate boxes, B’s children B_0 to B_3. If B is a square of side length d, then its children will have identical side lengths d/2, and together exactly cover the area of B. Then, iterate through the set of points contained in B, and store each point with the single child candidate box that contains it. Afterward, apply the same algorithm to the four new candidate boxes.

After this procedure is finished, all boxes, starting with the root, are of two types: interior nodes, which do not contain points themselves, but instead contain four child nodes (hence the “quad” in “quadtree”), or leaf nodes, which contain no more than k data points each. Since the children of any interior node cover exactly the same area as B itself, and the same applies to the children of B and so forth, all descendants of any interior node are nested, like Russian dolls. Particularly, the union of the areas of all leaf nodes is exactly the area of the root node. Hence, the set of leaf nodes forms a data-dependent tiling of the domain of the data set, with larger leaves in areas of low data density and smaller leaves in areas of high density.

For 3D data, the creation algorithm for an octree works exactly the same for a quadtree, but instead of splitting each square interior node into four identical children, each cubic interior node is split into eight identical children (Fig. 3). All other properties of the tree structure still apply. One might wonder why quadrees are not a sufficient data structure for lidar data. After all, terrain models are mostly two-dimensional, with large horizontal extents and small vertical ranges. While this is somewhat the case for airborne lidar data, it does not hold for terrestrial lidar data, which often include vertical features and potentially even overhangs, resulting in multiple z values for a given x, y position in a grid. It is definitely not true in an engineering
context, where complete anthropogenic structures are scanned from multiple points on the ground or in the air (Fig. 3). Using octrees over quadtrees significantly improves generality, but does not have a major impact on performance.

**Point-based Computation**

A central benefit of point tree-based storage is data locality. While the above algorithm does not mention such details, actual implementations will store nodes such that any interior node’s children are adjacent in the resulting data file(s). Because adjacency of children carries over to their children and descendants, the result is that data points which are spatially close are typically close in file order. This has significant implications on processing large scattered data sets. When applying local operators such as computing derivatives, one does not have to load the entire data set into main memory. Because derivative computation only requires knowledge of a small neighborhood around any given point, only a small part of the tree needs to be loaded at any given time. When applying a local operator to all points in sequence, in the order in which they are stored in the octree, one has to find all points near a small neighborhood around any given point, inside a node’s children are assessed recursively (called “level-of-detail rendering”).

![Figure 2. Recursive creation of a point quadtree for \( k = 2 \), i.e., no node contains more than two points. The tree is created starting at the root (leftmost panel), which encloses the entire point set. In each level (three right panels), any node containing \( k \) or fewer points is marked as a leaf (indicated by shaded background), and all other nodes are split into four child nodes to be processed on the next level. The full tree (rightmost panel) shows a data-dependent tiling of the point set’s domain.](image)

As a result, this algorithm is competitive with parts of a data set need to be loaded at any time. Otherwise, if more detail is required, the node’s children are assessed recursively (called “level-of-detail rendering”).

**Point-based Rendering**

A central usability requirement for point-based computing applications is real-time visualization of point clouds at full density, i.e., the ability to render views of the point cloud above 30 frames per second (Kreylos et al., 2003) without visibly degrading the resolution of the point cloud. The fundamental property that enables such rendering for very large data sets is that computer displays have a finite number of pixels. It does not make sense to draw more data points than there are pixels on the screen, since each pixel can show at most one—the closest—data point. The hierarchical structure of octrees provides a very good basis for implementing view-dependent rendering algorithms that can quickly extract the subset of data points that most closely maps to pixels on the display. LidarViewer extends the preprocessing algorithm described above by computing successively reduced-density point sets for the interior nodes of the octree. In other words, octrees for rendering contain reduced-resolution versions of their full point sets in higher levels of the tree. Storing these additional points, which are only used during rendering, leads to a small (approx. \( 1/8 + 1/64 + 1/512 + ... = 1/7 \)) increase in total data size, as each interior node replicates on average 1/8 of the lidar points contained in its children.

At rendering, nodes are processed recursively starting at the root node. If a node is entirely off-screen, it and its children are skipped entirely (called “view frustum culling”) (Clark, 1976). If an interior node is at least partially visible, its reduced point set is rendered immediately if the level of detail is appropriate, e.g., if the node is small or far away from the viewer (see Fig. 3A). Otherwise, if more detail is required, the node’s children are assessed recursively (called “level-of-detail rendering”).

*Geosphere, June 2013* 549
The combination of view frustum culling and level-of-detail rendering ensures that only a small subset of the original point cloud is rendered for any frame, without impacting visual quality. If a view is from far away and shows an entire data set, a reduced-resolution version is rendered, but this reduction is invisible due to the fixed pixel size of the screen. If a view is from up close, the full-resolution point cloud will be rendered, but almost all of it will be culled because it lies off-screen. This ensures that, by adjusting the so-called “level-of-detail cut-off value,” real-time frame rates may be sustained independently of the performance of the computer or the size of the data set.

Rapid Data Assessment

Due to the high cost of mobilizing an airborne lidar survey, quality control during a survey is critical to generating the best possible terrain product. Returning later to fill in gaps or re-fly problematic lines may be prohibitively expensive. Even in comparatively cheaper terrestrial surveys, being able to judge data coverage during collection can optimize survey workflows. LidarViewer is ideally suited to the task of manual data assessment because the octree structure is quick to generate and amend with additional data as a survey proceeds. Because the original swaths of data points are rendered by LidarViewer in 3D, gaps between surveys, holes in coverage, or gross registration errors can be identified immediately after preprocessing, omitting the time-consuming step of gridding the data first. Likewise, rapid real-time rendering of the point cloud enables monitoring for problems with the lidar instrument, differential GPS system, or inertial measurement unit that could manifest as missing points or apparently distorted lidar images. Such rendering also enables surveyors to determine if the point density is sufficient to resolve features of particular interest, evaluate the extent of vegetation penetration, or to check for excessive point loss due to dust or moisture.

Figure 3. Two screen shots from LidarViewer, showing the structure of the rendered octree at two different rendering quality settings. (A) Reduced-quality view for illustration purposes. (B) Full-quality view, rendered above 30 frames per second on current laptop or desktop computers. In both panels, red boxes outline all interior or leaf nodes selected for rendering based on the level-of-detail cutoff. It is noteworthy that even the full-quality view on the right does not use the full resolution of the data set. The shown data set is a terrestrial lidar scan of the UC Davis water tower and Mondavi Center for the Performing Arts, viewed from the south (38.5334°N, 121.7495°W). These engineered structures show an advantage of point-based rendering over grids, which could not represent the structures at all, and the benefits of octrees over quadtrees, which could not represent them efficiently.

LidarViewer (Kreylos et al., 2008) is developed at the UC Davis W.M. Keck Center for Active Visualization in the Earth Sciences (KeckCAVES). The software package is built on top of the Virtual Reality User Interface (Vrui) toolkit (Kreylos, 2008), also developed at KeckCAVES. The Vrui toolkit provides a uniform interface to 3D visualizations across a large range of hardware, from a laptop computer to a CAVE Automatic Virtual Environment (CAVE). The LidarViewer package also includes a suite of auxiliary software programs for processing and subsetting lidar data. LidarViewer is released under the GNU General Public License v2 and works on Linux, Mac OS X, and other UNIX-like operating systems. Download links and operating instructions are online at http://wiki.cse.ucdavis.edu/keckcaves/home. The following subsections show examples of earth science applications that highlight the current capabilities of LidarViewer and its associated software programs. We conclude with a discussion of some future directions in the analysis of lidar data in a point-based computing framework such as LidarViewer.
Rapid data assessment plays a larger role in the realm of disaster response, where it is desirable to interpret lidar data concurrent with, or immediately after, its collection. For example, the airborne lidar data set collected over the surface rupture from the 4 April 2010 Mw 7.2 El Mayor–Cucapah earthquake in northern Baja California, a small subset of which is shown in Figure 1A, contains 3.7 billion points, was collected over ~2.5 days, and preprocessed afterwards into a 100Gb octree file in 3.5 hours using a quad-core 2.8GHz Linux PC (Table 1). Individual swaths of LAS files could have been preprocessed in a few minutes each as they became available, for concurrent analysis. Figure 1B shows how direct hillshading of the point cloud can quickly highlight subtle features of interest, such as the fault scarp of the Borrego fault illustrated here.

Whether viewing the raw intensity values (Fig. 1A) or a hillshade rendering (Fig. 1B), both representations consist solely of a cloud of points. Except in the nearest foreground, the density of this data set is sufficient for points to overlap, producing the illusion of a continuous surface. This quality of imagery can be produced dynamically by LidarViewer at 30 frames per second, such that the user can always navigate through the full data set at what appears to be its full resolution.

### Real-time Hillshading

By default, LidarViewer renders points with fixed color assignments, which may be either the intensity of the laser returns or a color assigned from co-located imagery. Intensity data, which are commonly discarded in gridding, can be useful in identifying features such as stratification or vegetation (Figs. 1A and 4A; Animation 1). With local tangent planes calculated in an optional illumination preprocessing step (Table 1), hillshading can be simulated from one or more dynamic and arbitrary light sources (Figs. 1B and 4B), and may be combined with color information to produce a hybrid image (Fig. 4C).

For the Raplee Ridge area, point-return intensity shown in Figures 4A and 4C and Animation 1 carries useful information on the lithology of deformed beds. In this case, brightly colored gray limestone reflected more laser energy than dark red sandstone or shale units.

One of the powerful capabilities of modern graphics hardware and software is dynamic shading of rendered 3D objects. LidarViewer takes advantage of such technology to generate hillshaded terrain views in real-time. Users can adjust the azimuth and elevation of a “virtual sun” via a graphical user interface (GUI), and can create and move additional light sources either via a GUI, or directly via tracked 3D input devices, which then become “virtual flashlights.” Real-time illumination from multiple light sources combined with 3D rendering of full-resolution point clouds from arbitrary points of view provides a powerful way to interactively visualize lidar point clouds. Such interactivity, in turn, enables new approaches to locating features of interest in point-cloud data, such as bedding planes (Fig. 4B) or fault scarpers (Fig. 1B).

Currently, LidarViewer renders raw point color (e.g., intensity), hillshading, or a combination of these. Rendering of slope, aspect, and curvature is usually projected onto data points underneath it. The selection brush’s radius, aspect, and curvature are straightforward extensions of the hillshading algorithm and can be done by the user as an additional preprocessing step using a Python programming interface. The ability to display multiple landscape attributes from one data set is a goal for the next major release of LidarViewer.

### Selection and Measurement of Features

By providing the ability to directly interact with raw lidar points, LidarViewer makes it simple to select and measure features embedded within point clouds. As an intuitive example, we show in Figure 5 and Animation 1 how one can quickly and interactively fit a geometric plane to bedding to measure dip and dip-direction. Data points are selected via a spherical selection brush of adjustable radius. The brush’s radius is a physical (relative to the screen), rather than geographic (relative to the data) value, such that zooming in or out allows the user to quickly make accurate selections by working at multiple scales in rapid succession. In an immersive 3D environment (3D display and tracked 3D input devices) such as a CAVE, the selection brush may be moved arbitrarily through the data to select points. For typical 2D environments, e.g., desktop or laptop computers, the selection brush is usually projected onto data points underneath the mouse cursor, therefore locking the brush to the scanned terrain. In both cases, it is straightforward to select points from planar features in the landscape (Fig. 5A).

Once a feature has been selected, a user can immediately construct a model of the feature by fitting one of several geometric primitives to the selected points using a least-squares approach (Fig. 5B; Animation 1). There are four advantages to working with original point data for this purpose: first, the selected set reflects the available independent point measurements, which thus leads to a realistic evaluation of goodness-of-fit to the data (reported by LidarViewer as an RMS residual). Second, points may be included or excluded from the selected set based on objective criteria, such as exclusion of vegetation based upon its peculiar spatial structure and laser-return intensity, or inclusion of data identified as the target bed from its position and laser-return intensity (e.g., Fig. 5A). Third, areas of no data, such as shadows adjacent to a ledge of bed- 

### Interactive Analysis of Point Clouds

The native 3D data representation of LidarViewer lends itself to types of interactive analysis not possible with gridded data. Here, we show an example of analysis of point distance from a fitted plane using an example of terrestrial laser scanner data collected following the El Mayor-Cucapah earthquake (Gold et al., 2010). These data include very densely spaced (~1 cm spacing) measurements on an exposed fault free face (Fig. 6A). After fitting a plane to the fault surface, we use functionality built into LidarViewer to render points based on real-time calculation of orthogonal distance from this plane (Fig. 6B). The analysis shows subtle undulations in the fault plane that may be related to the near-field deformation, such as secondary faulting and folding of the hanging wall. Note that this
example also illustrates the importance of 3D representation of terrestrial laser-scanner point clouds: the detailed texture of the 85° dipping fault free face would be projected into a horizontal grid less than one tenth the area of the original fault-face, likely resulting in severe degradation from its original resolution. This problem could also be overcome with TIN-based techniques, but these do not offer any computational advantage over the raw point cloud.

Another example of interactive analysis that may be employed with LidarViewer is manual devegetation of vegetation within a point cloud (Animation 2). Though impractical over an expansive laser scan, manual devegetation can be effective at selected study sites where it is desirable to maximize the number of ground returns. Typically, devegetation is employed as a filtering step prior to production of a gridded digital elevation model. Several criteria may be employed as filters, though local point-to-point slope plus a distance threshold is often the most effective mechanism for classifying vegetation returns (e.g., Haugerud et al., 2003). By nature, this approach will also lead to undesirable effects, such as blunting of locally steep features (e.g., channel walls and fault scarps), and dilution of ground returns on steep slopes. By working with LidarViewer in a 3D environment, vegetation may be visually discriminated from the ground plane. By further employing 3D interactive selection with a tracked 3D input device, vegetation returns can be selected efficiently, and subsequently removed from the point cloud using LidarSubtractor, an additional auxiliary program in the LidarViewer package. The manual approach requires an immersive 3D visualization environment such as a CAVE or low-cost (~$7,000) desktop 3D display system (http://www.idav.ucdavis.edu/~okreylos/ResDev/LowCostVR/index.html).

Thus far, we have employed manual devegetation in both airborne and terrestrial laser-scanner data (Figs. 6 and 7). In a study of the Enriquillo fault in Haiti, Cowgill et al. (2012) found that automated vegetation classification removed up to 90% of laser returns from the airborne lidar data collected immediately after the 12 January 2010 earthquake. Though suitable for removal of cultural features in a low-relief urban setting, vegetation filtering of the Haiti data set proved too aggressive for mapping of subtle fault-zone features. By manually selecting and removing vegetation returns, Cowgill et al. (2012) found that only ~30% of the points were non-ground returns at selected study sites, and that even a fine-tuned version of the automated vegetation-removal algorithm could not retain more than 50% of the points. Likewise, comparing automatically and manually classified data from the northern San Andreas Fault imaged by NCALM yielded 83% and 51% of points as non-ground returns, respectively (Animation 2; Fig. 7).

For terrestrial laser-scanner data, we have found that the high level of detail and 3D structure of scans may be prohibitive to fully automatic filtering. For the example data from Baja California (Fig. 6), we employed a mean-height filter to remove vegetation points that were clearly above the ground surface, and then employed manual classification to discriminate
the remaining vegetation points from terrain features. In many cases, manual vegetation classification may be more practically applied to terrestrial lidar data sets where the higher return density and greater precision make it more straightforward to distinguish between vegetation and ground returns.

**DISCUSSION**

Our experience with the LidarViewer software package has shown that point-based computing and visualization are feasible approaches to working with lidar data, and expand the set of tools available to work with and gain knowledge from such data. Limitations on data set size, in-memory indexing of data, and visualization can be overcome through the application of appropriate algorithms. In current practice, point-based methods may be only applied as the first steps in lidar data processing, i.e., vegetation removal and gridding, after which the original

---

**Figure 5.** Point-based feature selection and extraction, here used to calculate bedding dip and dip direction in the Raplee Ridge lidar survey (selection centered at 37.1832°N, 109.8271°W). (A) Illustration showing how the user zoomed in to see and work with individual point returns. The white arrow indicates the selection brush (yellow sphere), which is locked to the terrain surface because this screenshot was taken in a desktop environment. Already selected points are highlighted in green. (B) View showing the geometric plane (in transparent yellow) extracted from the points selected in A. Arrows and labels on the plane indicate dip and dip direction. The thin blue line points straight up, and the thin green line points due north.
raw data are no longer used. One reason for this practice is that a substantial body of standard terrain processing routines have been designed to operate on grids (e.g., Band, 1986) and are available in commonly used software packages. A second reason for this is that current software has not been designed to overcome representation and processing overhead incurred by scattered data, although a number of x-y grid programs (e.g., AutoCAD, ArcGIS) are developing lidar tool sets that can handle 3D data more efficiently. As demonstrated here, appropriate data structures and algorithms can lead to processing performance on par with equivalent grid-based approaches (Table 1). When properly implemented, point-based computing is so efficient that it is not necessary to subset or tile a data set before applying operators to it.

Another benefit of point-based computing in the context of lidar data is that it works directly on the raw data and not on derived (gridded) products. Gridding is an implicit low-pass filter, and particularly affects linear features such as channels or edges that are not aligned with the primary grid directions. Furthermore, grid nodes are models of the true data and do not correspond to original data points, which makes it difficult to estimate errors when fitting geometric primitives to a surface. Grids also cannot represent discontinuous features, such as near-vertical fault-scarp surfaces, whereas point-based computing can explicitly detect and handle discontinuities. A fundamental problem with grid-based approaches is point clouds with non-uniform point densities, such as radially scanned terrestrial lidar scans, multi-swath airborne scans, or merged airborne/terrestrial scans. The spacing of a grid needs to be chosen carefully to minimize aliasing and loss of fine detail, and in non-uniform point sets, no single grid spacing works over an entire data set. Point-based computing, on the other hand, handles non-uniform densities inherently and without special care.

A downside of point-based computing is that it requires non-trivial data structures and algorithms to be efficient. While these can be exposed via easy-to-use programming interfaces (LidarViewer itself contains a Python interface to implement arbitrary point-based computations efficiently), the hierarchical structure of point trees is typically lost when exporting (parts of) point clouds to other processing software (e.g., Matlab), which can lead to performance reductions by several orders of magnitude.
Much potential exists to extend the point-based computing approach of LidarViewer to many operations that are typically performed on gridded data. The most straightforward are extensions of the existing local derivative calculator, which could also compute flow direction, divergence, and various forms of curvature, as well as surface texture and roughness. The hierarchical octree data structure may be leveraged to compute similarly hierarchical terrain attributes, such as flow accumulation and delineation of watersheds. Importantly, the lack of a set scale in the point-based approach lends itself to multi-scale analysis, which is useful in computation of derivatives and extraction of features such as stream channels. One of the key benefits of point-based computing over grids is that derivatives and feature extraction do not suffer from inherent bias toward the primary grid directions. This benefit extends to extraction of arbitrary topographic profiles, which is essentially itself an act of interpolation, and thus strongly subject to aliasing artifacts when derived from an already existing grid.

Another promising avenue of development is feature comparison and matching, such as quantifying landslide movement or for calculating fault slip vectors by correlating features. Because small and particularly linear features are not degraded, displacement vectors may be calculated from point clouds with higher accuracy than in purely grid-based approaches. Additionally, it is very straightforward to retro-deform a landscape using point clouds, because each 3D point may be precisely offset by its displacement vector. Retro-deformation on a grid, on the other hand, is challenging because shifting features underneath the grid further degrade the quality of the grid’s representation.

A frontier of point-based computing of lidar data will be to take full advantage of the rich 3D structure of landscapes in semi-automated processing and analysis. For example, vegetation classification and removal could be coupled to user training of the algorithm and on-the-fly adjustment of filtering parameters. Flow calculations within a channel or over a flooding landscape could be structured to include the vegetation information as a form of roughness, improving estimates of flow velocity and flood wavespeed. Ultimately, the free/open-source nature of LidarViewer, and the included Python programming interface, invite broader community participation in the development of novel point-based computation methods for terrain data.

CONCLUSION

Point-based computing offers a feasible alternative and improvement to grid-based analysis of terrain data that enables access to the full richness of information embedded within lidar point-clouds. Barriers to point-based computing are not computational—existing hardware is more than sufficient—but rather that currently used software was not designed to operate efficiently on very large unstructured data sets. Compared to computation based on triangular irregular networks (TINs), point-based computing is simpler and more storage-efficient, because it does not represent explicit connectivity between vertices.

Here, we have described the algorithms that enable efficient point-based computing on arbitrarily large data sets, and show an implementation of these algorithms in the software package, LidarViewer. The software, developed at the UC Davis W.M. Keck Center for Active Visualization in the Earth Sciences (KeckCAVES), enables real-time visualization (30 frames per second), as well as neighborhood- and selection-based computation on 3D lidar point clouds.

Four examples highlight the existing capabilities of the LidarViewer software. By avoiding gridding altogether, data may be rapidly assessed in LidarViewer for quality assurance. Preprocessing speeds match or exceed current rates of data collection. Through neighborhood-based calculation of normal-vectors, we show...
how hillshading can be extended to point cloud data. We further describe a technique for point-selection that may be used in 2D or 3D environments to subset lidar point clouds. Examples of applications of this technique include manual removal of vegetation points, fitting geometric primitives to selected data, measuring the plane orientation, and visualization of point distance with respect to a fitted plane.

The LidarViewer software package is open source and extensible through the source code or via a Python interface. Because grids are a subset of a point cloud, all operations currently tailored to gridded terrain data could in theory be implemented with the point-based approach. However, gridding cannot replicate the full richness of information preserved in unstructured point-cloud data. Point-based computing thus offers a new pathway to interrogate this information in scanned terrain data, while overcoming the fixed scaling and directional aliasing inherent in grid-based techniques.

ACKNOWLEDGMENTS

We thank the W.M. Keck Foundation for funding the establishment of the UC Davis KeckCAVES. This contribution is based upon work supported by the W.M. Keck Foundation and by the National Science Foundation (NSF) CI-TEAM program (OCI-0753407). Data collection from the El Mayor-Cucapah surface rupture was supported by an NSF RAPID grant (EAR-1039168), with additional support from the Southern California Earthquake Center (SCEC). SCEC is funded by NSF Cooperative Agreement EAR-0169024 and USGS Cooperative Agreement O2HG/AG0008. George Hilley and Dave Pollard kindly provided access to the Rapture Ridge lidar data set, acquired as part of a project funded by the NSF (EAR-0417521). Gerald Bawden provided the UC Davis water tower scan. We thank the members of the UC Davis KeckCAVES and the Institute for Data Analysis and Visualization for their support and feedback throughout development of the LidarViewer. Discussions with R. Haugenedt clarified ideas presented here.

REFERENCES CITED

Kreylos et al.