Surface Extraction from Point-Sampled Data through Region Growing

Miguel Vieira* and Kenji Shimada
Mechanical Engineering Department, Carnegie Mellon University

Abstract

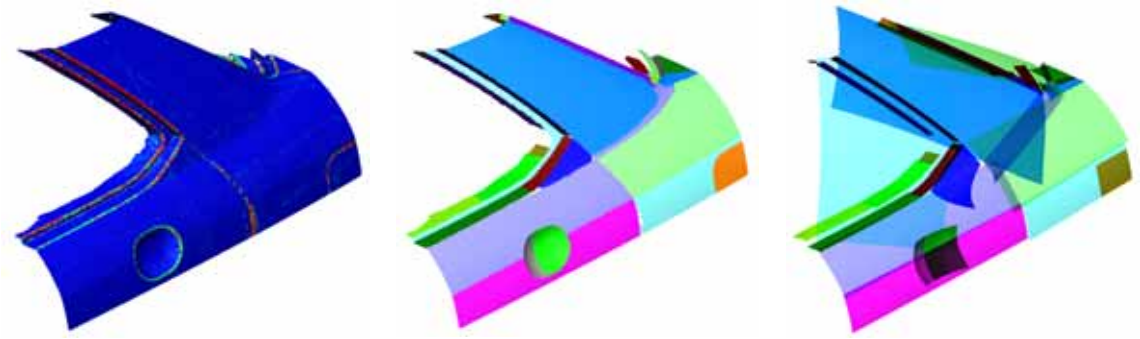
As three-dimensional range scanners make large point clouds a more common initial representation of real world objects, a need arises for algorithms that can efficiently process point sets. In this paper, we present a method for extracting smooth surfaces from dense point clouds. Given an unorganized set of points in space as input, our algorithm first uses principal component analysis to estimate the surface variation at each point. After defining conditions for determining the geometric compatibility of a point and a surface, we examine the points in order of increasing surface variation to find points whose neighborhoods can be closely approximated by a single surface. These neighborhoods become seed regions for region growing. The region growing step clusters points that are geometrically compatible with the approximating surface and refines the surface as the region grows to obtain the best approximation of the largest number of points. When no more points can be added to a region, the algorithm stores the extracted surface. Our algorithm works quickly with little user interaction and requires a fraction of the memory needed for a standard mesh data structure. To demonstrate its usefulness, we show results on large point clouds acquired from real-world objects.

Key Words: point-sampled geometry, surface reconstruction, surface extraction, segmentation, region growing

1. Introduction

Point clouds are becoming an increasingly common initial digital representation of real-world objects. This is due to the popularity of affordable and accurate scanning equipment that can quickly digitize the geometry of a real-world object. However, the resulting point cloud representing an object's surface can often contain millions of three-dimensional points. This large size and the noise associated with measurement can make processing the data difficult. Nevertheless, there are many applications where it is necessary to create a computer model consisting of just a few simple surfaces from the point-cloud data. Accurately reconstructing the object's geometry in this way is often a difficult and time-consuming task. In this work, we present a method for automatically extracting surfaces from point-cloud data that works quickly and minimizes the memory overhead of handling large data sets. Our surface extraction technique segments the point cloud and has the potential to be used in surface reconstruction, reverse engineering, industrial design, and rapid prototyping.

* Corresponding author:
Email: mcv@andrew.cmu.edu



(a) Surface variation (100,000 points) (b) Points assigned to surfaces (c) Extracted surfaces

Fig. 1. Different surface extraction steps for an automobile C-pillar.

Surface extraction through region growing is founded on the assumption of surface coherence [11, 38]. This is the observation that, despite the presence of noise, almost every point sampled from an object's surface will be geometrically related to its nearby points in that they will all lie near a single, smooth surface. Furthermore, it implies that the connectivity information of a triangle mesh can be replaced by spatial proximity of the sampled points if the point cloud is sufficiently dense. Our algorithm exploits this property to automatically organize the point cloud into distinct regions approximated by simple surfaces.

Our method iterates between region growing and surface fitting to automatically find sets of points that can be closely approximated by single surfaces. Given a set of N points $\mathbf{X} = \{\mathbf{x}_0, \ldots, \mathbf{x}_{N-1}\}$ sampled from a surface in $\mathbf{R}^3$ and a seed point $\mathbf{x}_i$, our algorithm attempts to find a parametric surface $\mathbf{b}(u,v)$ and a maximal set of points $\mathbf{R}_\mathbf{b}$, such that every point in $\mathbf{R}_\mathbf{b}$ is geometrically compatible with $\mathbf{b}(u,v)$. We define a point to be geometrically compatible with a surface if its position and normal differ from the nearby surface by small amounts. In this presentation, we assume all points have consistently oriented normals $\mathbf{N} = \{\mathbf{n}_0, \ldots, \mathbf{n}_{N-1}\}$. If the normal data is not available, then it can be calculated from a mesh representation, from the laser scanner acquisition process, or by estimating the tangent plane of each point and then propagating normals along a minimum spanning tree as described in [19].

The algorithm presented here, summarized in Algorithm 1, first partitions the bounding box of $\mathbf{X}$ into a cubical grid for efficient nearest-neighbor searching. Then it attempts to grow a region from each point $\mathbf{x} \in \mathbf{X}$ in order of increasing surface variation $\sigma_k(\mathbf{x})$, determined by principal component analysis. Region growing first checks if a small neighborhood, the seed region $\mathbf{S}(\mathbf{x})$, around a point can be approximated with a single surface $\mathbf{b}_{init}(u,v)$. If so, the compatible points in the seed region define a new region $\mathbf{R}_{\mathbf{b},new}$ and the algorithm adds to this new region all nearby points geometrically compatible with the surface. Once all compatible points have been added, the algorithm fits a new surface $\mathbf{b}_{new}(u,v)$ to the entire region and repeats the region growing. A final surface $\mathbf{b}(u,v)$ is extracted when the number of points in the region stops increasing. The different steps of the algorithm

EXTRACT-SURFACES($\mathbf{X}$)

Find x, y, z extents of $\mathbf{X}$ *and partition domain into a cubical grid*

for each $\mathbf{x} \in \mathbf{X}$, *calculate and store distance to k-nearest neighbors*

for each $\mathbf{x} \in \mathbf{X}$, *calculate and store* $\sigma_k(\mathbf{x})$

for each $\mathbf{x} \in \mathbf{X}$ *in order of increasing surface variation*

if LABELED($\mathbf{x}$), **continue** *(skip to next point)*

construct $\mathbf{S}(\mathbf{x})$ *and find* $\mathbf{b}_{init}(u,v)$

$\mathbf{b}_{new}(u,v) = \mathbf{b}_{init}(u,v)$

CLEAR($\mathbf{R}_{\mathbf{b},new}$)

do

$\mathbf{R}_{\mathbf{b},old} = \mathbf{R}_{\mathbf{b},new}$

$\mathbf{b}_{old}(u,v) = \mathbf{b}_{new}(u,v)$

REGION-GROWING($\mathbf{S}(\mathbf{x})$, $\mathbf{b}_{new}(u,v)$, $\mathbf{R}_{\mathbf{b},new}$)

while ($\left|\mathbf{R}_{\mathbf{b},new}\right| > \left|\mathbf{R}_{\mathbf{b},old}\right|$)

if $\left|\mathbf{R}_{\mathbf{b},new}\right| < \left|\mathbf{R}_{\mathbf{b},old}\right|$

$\mathbf{R}_{\mathbf{b}} = \mathbf{R}_{\mathbf{b},old}$

$\mathbf{b}(u,v) = \mathbf{b}_{old}(u,v)$

else

$\mathbf{R}_{\mathbf{b}} = \mathbf{R}_{\mathbf{b},new}$

$\mathbf{b}(u,v) = \mathbf{b}_{new}(u,v)$

create new region from $\mathbf{R}_{\mathbf{b}}$ *and* $\mathbf{b}(u,v)$

Algorithm 1. Pseudocode for the surface extraction algorithm

are illustrated in Figure 1. Figure 1(a) shows the point colored by surface variation, Figure 1(b) shows the points colored according to different underlying surfaces, and Figure 1(c) shows the extracted surfaces painted with the same colors as the points they represent. Note that in all of our examples the point clouds appear continuous because they are dense enough that there is more than one point per pixel.

Our region growing approach for surface extraction is intuitive, efficient, and straightforward to implement. By reducing the data representation to only what is essential, we eliminate the need for a mesh data structure and reduce the algorithmic complexity and memory requirements for processing the data. This allows users to process larger data sets, including not just point clouds, but also parametric and polygonal data, which one can easily convert into dense point clouds. Our algorithm can extract parametric surfaces from sets of millions of points in minutes with guaranteed approximation errors. Furthermore, one can speed up the surface extraction by simplifying the point cloud as in [29] without

creating a mesh. Finally, the region-growing algorithm can be easily adapted so that a user can simply click on a point cloud to automatically extract surfaces from it.

2. Related Work

Surface reconstruction from point clouds has received considerable study and continues to do so. There are several existing methods for generating a piecewise-linear mesh from a point cloud [13, 19, 5, 6]. There are also methods for reconstructing a smooth or piecewise-smooth surface from a point cloud [16, 7, 27, 26, 12, 20]. However, for our target applications, creating an approximating mesh from the point cloud is unnecessary. Furthermore, a fully automatic smooth or piecewise-smooth surface reconstruction that results in an implicit surface or a tiling of many triangular or rectangular patches is difficult for a designer or modeler to modify and cannot produce large surfaces of acceptable quality for consumer product styling. More recently, point clouds have also received attention as a potential rendering primitive because of their compactness and the increasing commonness of very large data sets [23, 31, 34, 4, 1, 2, 3].

Our region growing method is similar in principle to existing methods for automatically partitioning meshes into regions that can be easily parameterized. For example, [24, 15] use an *s*-source adaptation of Dijkstra's shortest-path algorithm to segment a mesh. On point clouds, [28] grows regions by enforcing a normal cone condition, requiring that the largest angle spanned by the point normals in a region is below a certain threshold. The present region growing method is more specific than these, requiring that both the positions and the normals of points are close to that of an underlying surface.

Our surface extraction algorithm can also be viewed as a complement to existing algorithms for extracting features from point clouds [18, 30]. These papers apply principal component analysis to a point cloud to detect features then use a minimum spanning graph of the feature points to extract sharp edges. Pauly, et al, [30] further model the extracted edges with adaptive contour models. In our approach, we use principal component analysis to find the areas of smallest variation, then use region growing to find clusters of points lying near smooth surfaces. While feature extraction is most suitable for mesh generation and non-photorealistic rendering, surface extraction is useful for reverse engineering, surface reconstruction, and industrial design.

The concept of growing regions to find groups of points that can be approximated by surfaces was introduced for images by [11, 10] and adapted for gridded height data in [36]. In [39, 38], the present authors applied the region growing approach to segment dense, unstructured meshes. In this paper, we further extend the region growing approach into a method for automatic surface extraction from point clouds. As widespread use of modern measurement equipment makes very large data sets more ubiquitous, operating directly on the point cloud becomes more desirable. By utilizing only the necessary data,

the amount of computer memory required for analysis becomes a fraction of what is required for maintaining a mesh data structure.

The work done by Huang and Menq and Benkő and Várady is most similar to ours. The method proposed by Huang and Menq [21, 22] first constructs a mesh from the point cloud, then segments it and fits B-spline surfaces to the segments. However, the assumptions and complexity of the approach make it unfeasible for very large data sets. The method presented by Benkő and Várady [8, 9] is more closely related to ours. It applies a hierarchy of tests to recursively subdivide a point cloud into small regions that can be approximated by a single analytic surface such as a plane, cylinder, cone, sphere, or torus. However, it is not clear that such an approach can be used on point clouds representing free-form surfaces.

3. Surface extraction algorithm

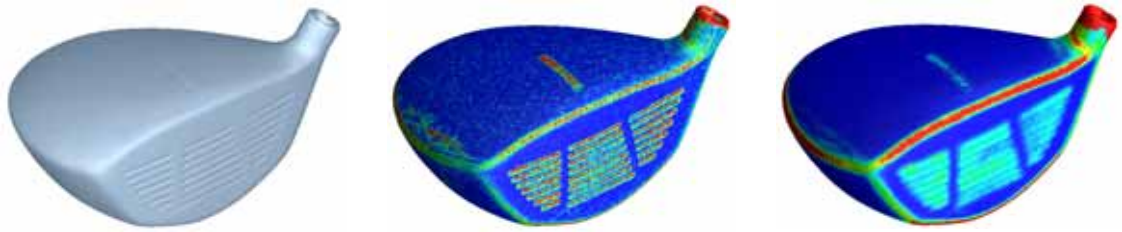
3.1. Seed point selection

Our algorithm begins with a single point, called a seed point, for the extraction of each surface. A small neighborhood around the seed point is then used to find an initial surface approximation for region growing. Intuitively then, region growing will be most effective when the seed point is interior to a large group of points lying near a single surface. We choose a simple, fast heuristic method to select seed points, first estimating the surface variation at each point in the point cloud, then examining the points in order of increasing surface variation. This makes the reasonable assumption that region growing will be more successful in areas with low variation than in areas with high variation. In addition, as surfaces are extracted from the point cloud, potential seed points can simply be skipped if they are already assigned to regions.

The surface variation at each point is calculated using principal component analysis, a technique commonly used to estimate local surface properties of point clouds [29, 18, 30]. It is illustrated in Figure 2. The technique can be understood as analogous to finding the mean and variance of a one-dimensional distribution. Let $\mathbf{N}(\mathbf{x}_i)$ be the *k*-nearest neighbors of a point $\mathbf{x}_i \in \mathbf{X}$. Denoting by $\bar{\mathbf{x}}$ the centroid of $\mathbf{N}(\mathbf{x}_i)$, we can define the 3×3 covariance matrix $\mathbf{C}$ by

$$\mathbf{C} = \sum_{\mathbf{y} \in \mathbf{N}(\mathbf{x}_i)} (\mathbf{y} - \bar{\mathbf{x}}) \cdot (\mathbf{y} - \bar{\mathbf{x}})^T . \tag{1}$$

This matrix is symmetric and positive semi-definite with real eigenvalues $\lambda_0 \le \lambda_1 \le \lambda_2$ and corresponding eigenvectors $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_2$ forming an orthogonal basis of $\mathbf{R}^3$. The eigenvalues λ_l measure the variance of $\mathbf{N}(\mathbf{x}_i)$ in the directions $\mathbf{v}_l$. In particular, $\mathbf{v}_0$ estimates the surface normal of $\mathbf{x}_i$ (up to sign) and the plane through $\bar{\mathbf{x}}$ spanned by $\mathbf{v}_1$ and $\mathbf{v}_2$ approximates the tangent plane at $\mathbf{x}_i$ [29, 19].



(a) Input point cloud (200,000 points)

(b) Surface variation using few nearest neighbors

(c) Surface variation using many nearest neighbors

Fig. 2. The effects of varying neighborhood size on surface variation estimation for a golf club head.

Therefore, λ_0 quantifies the deviation of $\mathbf{N}(\mathbf{x}_i)$ from the tangent plane and we can define the scale-invariant surface variation of the *k*-nearest neighbors as

$$\sigma_k(\mathbf{x}_i) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}. \tag{2}$$

As shown in Figure 2, this measure of variation distinguishes clearly between curved and flat regions.

3.2. Region growing

Region growing first finds a small set of points near the seed point and approximates this neighborhood with an initial surface. The region then grows by clustering points that are geometrically compatible with the approximating surface. Once all compatible points have been added, the algorithm fits a new surface to the region to improve the approximation. The algorithm then uses this new surface to re-grow the region. This process is repeated until the region size stops increasing.

As we progress through the point cloud, we attempt to approximate the neighborhood of each point $\mathbf{x}_i$ with a surface. We first check if $\mathbf{x}_i$ has been labeled as belonging to a region. If so, we simply skip it. If not, we construct a seed region $\mathbf{S}(\mathbf{x}_i)$ of all the unlabeled points within a radius $\rho_{\mathbf{S}}$ of $\mathbf{x}_i$:

$$\mathbf{S}(\mathbf{x}_i) = \left\{ \mathbf{x} \in \mathbf{X} \;\middle|\; \| \mathbf{x} - \mathbf{x}_i \| < \rho_{\mathbf{S}} \;\; \textit{and} \;\; \mathbf{x} \;\textit{is unlabeled} \right\}.$$

A seed region is illustrated in 2D in Figure 3. If $\mathbf{S}(\mathbf{x}_i)$ does not contain enough points for surface fitting, we move on to the next seed point, leaving $\mathbf{x}_i$ unlabeled. Otherwise, we fit a surface to the seed region $\mathbf{S}(\mathbf{x}_i)$ as described in Section 3.3. The radius $\rho_{\mathbf{S}}$ should be chosen to allow enough points for surface fitting but not so many that it includes points that may correspond to other surfaces. We discuss its selection in Section 4.

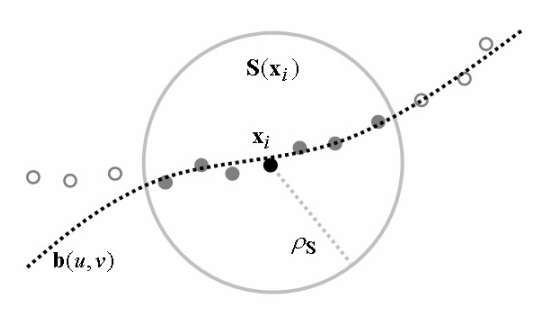


Fig. 3. Seed region selection. The points in the seed region are filled with gray. The initial surface is fit only to the points in the seed region.

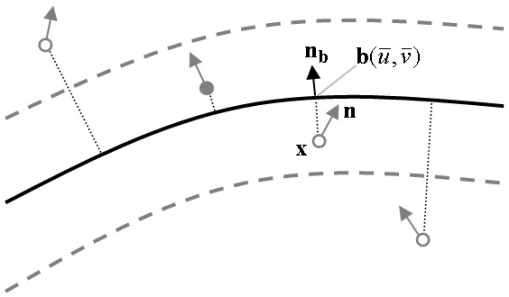


Fig. 4. Geometric compatibility. Only one point, filled with gray, is geometrically compatible with the surface. The dotted curves represent the G^0 compatibility threshold.

Each region formed by the algorithm is a set of points geometrically compatible with the region's underlying surface. We define geometric compatibility in a G^0 and G^1 sense. For a point $\mathbf{x}$, a parametric surface $\mathbf{b}(u,v)$, and parameters $(\bar{u},\bar{v})$ that minimize the distance from $\mathbf{x}$ to $\mathbf{b}(u,v)$, we say the point is G^0 compatible with the surface if

$$\left\| \mathbf{x} - \mathbf{b}(\bar{u},\bar{v}) \right\| < \varepsilon^0 \tag{3}$$

and G^1 compatible if

$$\cos^{-1}(\mathbf{n},\mathbf{n_b}) < \varepsilon^1 , \tag{4}$$

where $\mathbf{n}$ is the point normal, $\mathbf{n_b}$ is the surface normal at $(\bar{u},\bar{v})$,

$$\mathbf{n_b} = \frac{\mathbf{b}_u(\bar{u},\bar{v}) \times \mathbf{b}_v(\bar{u},\bar{v})}{\left\| \mathbf{b}_u(\bar{u},\bar{v}) \times \mathbf{b}_v(\bar{u},\bar{v}) \right\|}, \tag{5}$$

and ε^1 is in radians. All the different possible cases of compatibility and non-compatibility are shown in Figure 4. The dotted lines show the shortest paths to the surface and thus imply the direction of the surface normal $\mathbf{n_b}$ at the position on the surface nearest to each point. When point normals deviate too much, the points are incompatible with the surface. The compatibility thresholds ε^0 and ε^1 are the only user-defined parameters. They can be determined automatically as described in [38], but to increase interactivity and to decrease computation, we leave them to the user in this presentation. The parameters $(\bar{u},\bar{v})$ of the point on the surface closest to $\mathbf{x}$ are determined by first coarsely sampling points on the surface and then using the closest point to $\mathbf{x}$ as the initial value for a Newton iteration that minimizes the distance from the surface to $\mathbf{x}$. A detailed description of this process is given in [32].

Now that we have defined geometric compatibility, we can present the algorithm for region growing. Given a seed region $\mathbf{S}(\mathbf{x}_i)$ and a parametric surface $\mathbf{b}(u,v)$, the first region growing step

```
REGION-GROWING( S(x_i) , b(u,v) , R_b )
    CLEAR( R_b )
    CLEAR( Q )
    for each  x ∈ S(x_i)
        if  x  compatible with  b(u,v)
            LABEL( x )
            INSERT( R_b , x )
        ENQUEUE( Q , x )
    GROW( R_b , Q ,  b(u,v) )
```

Algorithm 2. This function grows a region given a seed region and an initial approximating surface.

```
GROW( R_b , Q ,  b(u,v) )
    while NOT-EMPTY( Q )
        x  ←  DEQUEUE( Q )
        N(x)  ←  k-nearest neighbors of  x
        for each  y ∈ N(x)
            if  y  unlabeled and compatible with
                b(u,v)
                LABEL( y )
                INSERT( R_b , y )
                ENQUEUE( Q , y )
```

Algorithm 3. The GROW function, called in Algorithm 2, uses a queue and nearest-neighbor searching to grow the region.

labels all the compatible points in $\mathbf{S}(\mathbf{x}_i)$ and adds them to the region $\mathbf{R_b}$. These points are also inserted into a queue Q used for region growing. The algorithm dequeues points and checks their *k*-nearest neighbors for any unlabeled points. If any of these unlabeled points are compatible with $\mathbf{b}(u,v)$, the algorithm inserts them into the queue. This continues until the queue is empty. This can be written in pseudocode as an Algorithm 2, where the function GROW that grows each region can be written as in Algorithm 3.

Once the region growing is complete, we fit a new surface to the points in $\mathbf{R_b}$ and repeat the region growing process with $\mathbf{S}(\mathbf{x}_i)$ and the new surface. As shown in Algorithm 1, we temporarily store the region points and underlying surface for each iteration until the region size does not increase from one iteration to the next. In this case, we stop region growing and use the larger of the current and previous regions and its corresponding underlying surface. This iterative procedure maximizes the size of each region while improving the quality of the approximating surface.

3.3. Surface fitting

The region growing framework is quite general and can be used with a variety of surfaces. The algorithm can be used with any class of surfaces that allows approximation (for surface fitting), differentiation, and point projection (for testing geometric compatibility). We implemented the region growing algorithm with non-rational bicubic Bézier surfaces because they are easy to manage yet representative of surfaces commonly used in computer-aided design. They can be expressed as

$$\mathbf{b}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{p}_{i,j} B_i^3(u) B_j^3(v), \quad 0 \le u,v \le 1, \tag{6}$$

where the $B_i^n(u)$ are the n[th]-degree Bernstein polynomials. Given parameters (u,v) for each point in the region $\mathbf{R_b}$, we can write an overdetermined system of linear equations that can be solved for the control points $\mathbf{p}_{i,j}$ using linear least squares [17, 33]. However, determining quality parameters for each point is often subtle [37, 25].

We adopt a fast and simple scheme for point parameterization that has given good results in all our test cases. Given the points $\mathbf{R_b}$ in a region, we calculate the plane perpendicular to a weighted average of the point normals. We first find the centroid $\bar{\mathbf{x}}$ of the points in the region and the maximum squared distance from any point to the centroid,

$$d_{\max}^2 = \max_{\mathbf{x}_i \in \mathbf{R_b}} \left\| \bar{\mathbf{x}} - \mathbf{x}_i \right\|^2. \tag{7}$$

The plane normal is the Gaussian-weighted average of the point normals in the region

$$\mathbf{n} = \frac{\sum_{\mathbf{x}_i \in \mathbf{R_b}} w_i \mathbf{n}_i}{\sum_{\mathbf{x}_i \in \mathbf{R_b}} w_i}, \quad w_i = \exp\left[-\frac{1}{2} \frac{\left\| \bar{\mathbf{x}} - \mathbf{x}_i \right\|^2}{\sigma^2} \right], \tag{8}$$

where σ is chosen such that $\exp[-1/2(d_{\max}^2/\sigma^2)] \approx 0.1$. We then project the points in $\mathbf{R_b}$ onto the plane defined by $\bar{\mathbf{x}}$ and $\mathbf{n}$ and rotate them in-plane so that the area of their axis-aligned bounding box is minimized. The in-plane coordinates are then linearly scaled so that each point in $\mathbf{R_b}$ is assigned parameters (u,v) such that $\alpha \le u,v \le 1-\alpha$. The constant α is necessary because the Bézier surfaces are only defined for $0 \le u,v \le 1$ and the region needs room to grow. In our implementation, we use $\alpha = 0.25$.

Once we have found parameters for each point, we can perform surface fitting with linear least squares. Although this initial surface approximation is often adequate, we can improve the fit accuracy by projecting the points back onto $\mathbf{b}(u,v)$ to find new parameters for them. These new parameters can then be used to fit a new surface to the region with linear least-squares. We use three iterations of this in our implementation.

4. Results and implementation

We implemented the algorithm as described in Section 3. Our current program can take in mesh or point-normal data and output control points for the parametric surfaces approximating the data.

For principal component analysis and region growing, we find the *k*-nearest neighbors with $k = 10$. To make the nearest neighbor searches faster, we precompute the radius ρ_i of the ball that contains the 10-nearest neighbors of each point $\mathbf{x}_i$. This eliminates the need for sorting or maintaining a priority queue when making nearest neighbor queries. For creating seed regions, we use the points in a ball of radius $\rho_{\mathbf{S}} = 3\rho$.

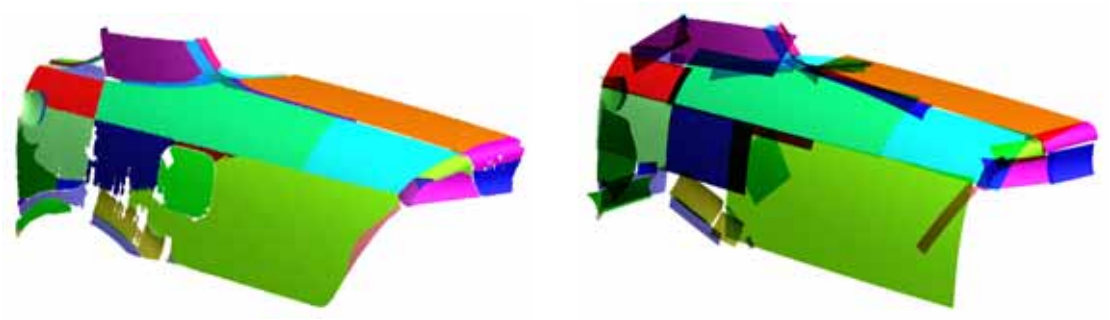
Table 1. Results with execution times and memory usage.

Model	Num. Points	Thresh. Angle	Peak RAM	Time (secs)
C-pillar	99,790	5°	25 MB	24
Rear Fender	1,065,886	6°	90 MB	255
Front Door	1,497,459	5°	125 MB	288

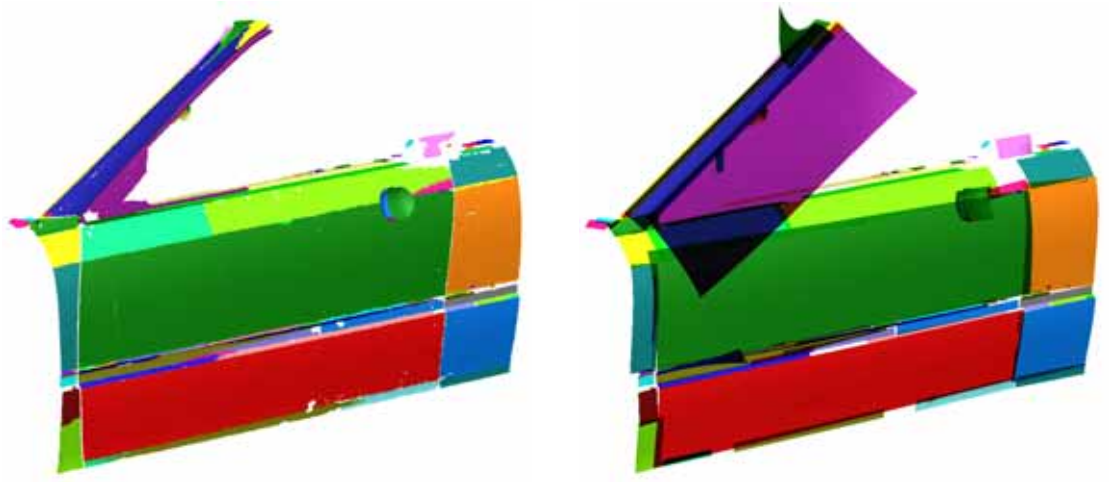
Table 1 shows the memory use and time for region growing. We do not include the time required for principal component analysis because it is a standard analysis tool and takes only a few seconds for even the largest models. A threshold distance of 0.3 mm was used for all the examples, so it is also elided. In practice, threshold distance can be easily determined based on the laser-scanner tolerances. The threshold angle is more application dependent. The algorithm extracts surfaces from relatively small models in seconds and takes just minutes for the largest models. Also, processing a point cloud takes considerably less RAM than processing a mesh. Our implementation approaches approximately 80 bytes per point for large models. In our experience, this is less than 25% of the RAM required to hold a mesh data structure with faces and edges representing the same data. Our tests were run on an AMD Athlon 64 2.2 GHz processor with 2 GB of RAM.

Figure 5 shows point clouds on the left and extracted surfaces on the right. The points in each region and their underlying surface have the same color. Notice that the algorithm can extract surfaces from large regions, often with a single surface representing hundreds of thousands of points. For visualization clarity we rendered the surfaces for $0.25 \le u, v \le 0.75$. The surfaces have otherwise not been trimmed to correspond to the point-set geometry.

Figure 6 demonstrates the quality of the extracted surfaces by simulating reflection lines on the point cloud. The figure was created by using the point normals to map a cylindrical light-strip texture onto the points. For the results, we projected labeled points onto the surfaces and assigned them the corresponding surface normals. Note that, in general, not all points were assigned to regions. In the figures, only labeled points are displayed.



(a) Rear Fender (1 million points)

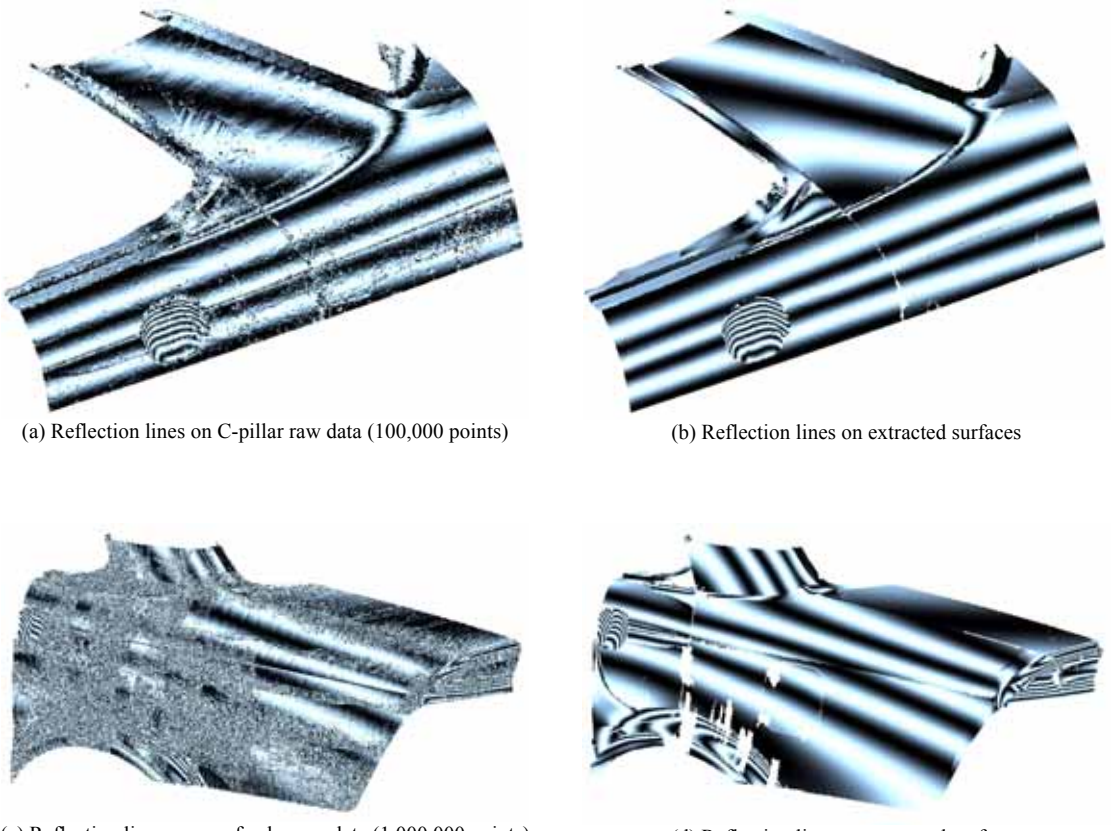


(b) Front Door (1.5 million points)

Fig. 5. The point clouds colored by region labels and the extracted surfaces.

5. Discussion

Our implementation processes point clouds in-core, requiring the entire point cloud in main memory. The most important requirement of the data structure for processing the point cloud is fast and efficient nearest-neighbors queries. For this, one can use hierarchical space-partitioning data structures such as octrees [35] or kd-trees [14], but finding nearest neighbors with these is logarithmic in N. For a large point cloud, region growing may make hundreds of millions of nearest-neighbors queries, making logarithmic time complexity unacceptable. Therefore, we exploit the fact that our data is somewhat uniformly distributed and use a simple cubical 3D grid data structure that allows nearest-neighbor queries in constant time with modest memory overhead. We first find the axis-aligned bounding box of the input data and calculate its volume. Then, we use the volume to partition it into K cubes where $K \approx N$. This partitioning induces a hash on the point coordinates, and each cube in the grid contains pointers to the points it contains. If the radius that contains the *k*-nearest neighbors is pre-computed, as in our implementation, then the nearest neighbors may be found without sorting, making *k*-nearest neighbors queries an $O(k)$ operation.



(a) Reflection lines on C-pillar raw data (100,000 points)

(b) Reflection lines on extracted surfaces

(c) Reflection lines on rear fender raw data (1,000,000 points)

(d) Reflection lines on extracted surfaces

Fig.6. Simulated reflection lines on point cloud data before and after surface extraction.

For each of the N points, we store the position and normal information along with the region number and the radius for nearest neighbor searching. The grid requires approximately $B(N+K)$ bytes, where $B=4$ on 32-bit architecture and $B=8$ on 64-bit architecture and K is the number of grid cubes. We note in passing that, if there is sufficient memory, pointers to nearest neighbors can be stored explicitly for each point, making nearest-neighbor searching an $O(1)$ operation during region growing. This can speed up the algorithm dramatically.

Also during region growing, when fitting a parametric surface to a large number of points (over, say, 10^5), the normal equations for linear least-squares can become ill-conditioned and numerically unstable [17]. Because some regions in our tests could contain over a million points, we use only a relatively small ($\approx 10^4$) random sample of the points in a region for parameterization and surface fitting. This adds to the numerical stability of the algorithm and decreases the processing time, yet makes no apparent difference in the results.

6. Conclusions and future work

We have presented a method for efficiently extracting surfaces from large point clouds with little user interaction. By taking advantage of the density of a point cloud, we can reliably find groups of points that can be approximated by single surfaces. The extracted surfaces can easily be output in an industry-standard format for designers and modelers to create a final design. Furthermore, the region growing algorithm can be implemented in an interactive setting where the user selects seed points for region growing. This would not require calculating the surface variation, of course.

We emphasize that the output of the algorithm is a disjoint set of surfaces to be used in downstream applications. In future work, we hope to adapt the surface extraction approach presented in this paper to create a full reconstruction of the scanned object. In particular, we would like to blend and intersect the extracted surfaces by using an implicit partition of unity approach to create a piecewise-smooth reconstruction of the scanned object [26]. After surface extraction, we could subdivide the domain into, say, cubical cells and then use the labeling of the points in each cell to determine whether to use one extracted surface in the cell, join two or more extracted surfaces in the cell, or fit a new surface to the points in the cell. Then the approximations in each cell would be blended with an implicit partition of unity. This approach should also handle sharp edges and holes in the segmentation.

7. Acknowledgements

This work is based in part on work supported by an NSF CAREER Award (No. 9985288). We would like to thank Cyberware for the Golf Club data set used in our testing.

References

1. A. Adamson and M. Alexa (2003), Ray Tracing Point Set Surfaces, Proceedings of Shape Modeling International 2003, 272-279.
2. A. Adamson and M. Alexa (2003), Approximating and Intersecting Surfaces from Points, Proceedings of EUROGRAPHICS '03, 245-254.
3. A. Adamson and M. Alexa (2004), Approximating Bounded, Non-Orientable Surfaces from Points, Proceedings of Shape Modeling International 2004, 243-252.
4. M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva (2001), Point Set Surfaces, IEEE Visualization 2001, 21-28.
5. N. Amenta, M. Bern and M. Kamvysselis (1998), A New Voronoi-Based Surface Reconstruction Algorithm, Proceedings of SIGGRAPH 98, 415-422.
6. M. Attene and M. Spagnuolo (2000), Automatic Surface Reconstruction from Point Sets in Space, Proceedings of EUROGRAPHICS 00, 457-465.

7. C. Bajaj, F. Bernardini and G. Xu (1995), Automatic Reconstruction of Surfaces and Scalar Fields from 3D Scans, Proceedings of SIGGRAPH 95, 109-118.
8. P. Benkõ and T. Várady (2002), Direct Segmentation of Smooth, Multiple Point Regions, Proceedings of Geometric Modeling and Processing, 169-178.
9. P. Benkõ and T. Várady (2004), Segmentation Methods for smooth point regions of conventional engineering objects, *Computer-Aided Design* 36(6), 511-523.
10. P. Besl (1988), *Surfaces in Range Image Understanding*, Springer -Verlag.
11. P. Besl and R. Jain (1988), Segmentation Through Variable-Order Surface Fitting, IEEE Transactions on Pattern Analysis and Machine Intelligence, 167-192.
12. J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans (2001), Reconstruction and Representation of 3D Objects with Radial Basis Functions, Proceedings of SIGGPRAH 2001, 67-76.
13. B. Curless and M. Levoy (1996), A Volumetric Method for Building Complex Models from Range Image, SIGGRAPH 96, 303-312.
14. M. de Berg, M. van Krevald, M. Overmars, and O. Schwarzkopf (2000), *Computational Geometry*.
15. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle (1995), Multiresolution Analysis of Arbitrary Meshes, Proceedings of SIGGRAPH 95, 173-182.
16. M. Eck and H. Hoppe (1996), Automatic Reconstruction of B-Spline Surface of Arbitrary Topological Type, Proceedings of SIGGRAPH 96, 325-334.
17. G. Farin (2002), *Curves and Surfaces for CAGD*, Academic Press, A Harcourt Science and Technology Company.
18. S. Gumhold, X. Wang, and R. MacLeod (2001), Feature Extraction from Point Clouds, Proceedings of the 10th Int. Meshing Roundtable.
19. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle (1992), Surface Reconstruction from Unorganized Points, SIGGRAPH 92, 71-78.
20. H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle (1994), Piecewise Smooth Surface Reconstruction, Proceedings of SIGGRAPH 94, 295-302.
21. J. Huang and C.-H. Menq (2001), Automatic Data Segmentation for Geometric Feature Extraction from Unorganized 3-D Coordinate Points, *IEEE Transactions on Robotics and Automation*, 268-279.
22. J. Huang and C.-H. Menq (2002), Automatic CAD Model Reconstruction from Multiple Point Clouds for Reverse Engineering, *Journal of Computing and Information Science in Engineering*, 160-170.
23. M. Levoy and T. Whitted (1985), The Use of Points as a Display Primitive, University of North Carolina at Chapel Hill Tech. Rept., TR 85-022.
24. B. Lévy, S. Petitjean, N. Ray, and J. Maillot (2002), Least Squares Conformal Maps for Automatic Texture Atlas Generation, Proceedings of SIGGRAPH 02, 362-371.

25. W. Ma and J. P. Kruth (1995), Parameterization of randomly measured points for least squares fitting of B-spline curves and surfaces, *Computer-Aided Design* 27(9), 663-675.
26. Y. Ohtake, A. Belyaev, M. Alexa, G. Turk and H.-P. Seidel (2003), Multi-Level Partition of Unity Implicits, Proceedings of SIGGRAPH 2003, 463-470.
27. Y. Ohtake, A. Belyaev and H.-P. Seidel (2004), 3D Scattered Data Approximation with Adaptive Compactly Supported Radial Basis Functions, Shape Modeling International 2004, 31-39.
28. M. Pauly and M. Gross (2001), Spectral Processing of Point-Sampled Geometry, Proceedings of SIGGRAPH 01, 379-386.
29. M. Pauly, M. Gross and L. Kobbelt (2002), Efficient Simplification of Point-Sampled Geometry, IEEE Visualization 02.
30. M. Pauly, R. Keiser and M. Gross (2003), Multi-scale Feature Extraction on Point-Sampled Surfaces, EUROGRAPHICS 2003.
31. H. Pfister, M. Zwicker, J. v. Baar and M. Gross (2000), Surfels: Surface Elements as Rendering Primitives, Proceedings of SIGGRAPH 2000, 335-342.
32. L. Piegl and W. Tiller (1997), *The NURBS Book*, Springer-Verlag.
33. W. Press, S. Teukolsky, W. Vetterling and B. Flannery (1997), *Numerical Recipes in C, Second Edition*, Cambridge University Press.
34. S. Rusinkiewicz and M. Levoy (2000), QSplat: A Multiresolution Point Rendering System for Large Meshes, Proceedings of SIGGRAPH 2000, 343-352.
35. H. Samet (1990), *Applications of Spatial Data Structures*, Addison-Wesley.
36. N. Sapidis and P. Besl (1995), Direct Construction of Polynomial Surfaces from Dense Range Images through Region Growing, *ACM Transactions on Graphics*, 171-200.
37. B. Sarkar and C.-H. Menq (1991), Parameter Optimization in Approximating Curves and Surfaces to Measurement Data, *Computer Aided Geometric Design*, 267-290.
38. M. Vieira and K. Shimada (2004), Surface Mesh Segmentation and Smooth Surface Extraction Through Region Growing, *Computer Aided Geometric Design,* (in print).
39. M. Vieira and K. Shimada (2004), Segmentation of Noisy Laser-Scanner Generated Meshes with Piecewise Polynomial Approximations, Proceedings of the ASME Design Automation Conference.