Sparse Surface Modeling with Curved Patches

Dimitrios Kanoulas and Marsette Vona

Abstract—Traditional segmentation algorithms for range images create partitions of connected and non-overlapping—but potentially irregularly shaped—regions corresponding to world surfaces. This paper presents an alternative paradigm based on regularly shaped curved patches (paraboloids) that model local contact regions potentially compatible with e.g. a robot’s toe, heel, or fingertip. These patches randomly sample the environment surface but are not required to strictly partition it. They are fit to neighborhoods of the range data and then validated for fit quality and fidelity to the actual data—extrapolations (like hole-filling) which are not directly supported by data are avoided. Two different neighborhood formation methods based on k-d tree and triangle mesh data structures are compared, and results are presented for 10 datasets taken in natural rocky terrain.

I. INTRODUCTION

Some of the most challenging open problems in robotics are those which will require reliable contact with unstructured world surfaces, for example, walking on natural rocky terrain or climbing in a pile of rubble [1]. Recent advancements often either assume significant structure in the environment (like humanoid walking on flat ground [2]) or that irregularities can be tolerated by low-level feedback control during contact (as in BigDog [3]). 3D perception is sometimes used for obstacle detection or coarse traversability prediction (as in the Mars Exploration Rovers (MER) [4]). However, to enable rough-terrain walking and climbing, a perception system that can spatially model and finely quantify potential 3D contact surfaces may be needed.

Paraboloid surface patches with regular boundaries and one or two curvatures can be the basis for such a system: they can be applied to unstructured scenes, they are potentially easier to reason about than polygon mesh, continuous height-field models (e.g. [5]), or multiply-curved NURBS\footnote{Non-Uniform Rational B-Spline.} patches (e.g. [6]), and they can model rough environments more compactly than planar patches (e.g. [7]).

Instead of building patches with widely varying boundary size, we propose to use one or several patch sizes driven by the task and robot structure. For example, in rough terrain walking the patch size(s) could be approximately the same as robot’s foot pads. Patches may overlap to provide a sampling of possible contacts in smooth regions, and they may also only partially cover the environment.

Figure 5 shows the set of bounded paraboloid patches from our prior work [8]. We use the models and the fitting algorithm from that work, but now fit many patches randomly over an environment surface that has been sampled by a range sensor. We are also working on detecting salient features such as peaks, valleys, and flats, and to integrate task-specific strategies to only sample relevant parts of the environment. These will be important refinements, but they will not obviate the need to (a) find good neighborhoods in the range data and (b) validate the fitted patches, which are the main topics of this paper.

Our algorithm (Figure 1) is divided into four main steps for each patch; patches may be fit either iteratively or in parallel. The first step is the selection of a seed point. Next is the neighborhood search around that point. We compare two different approaches for finding local neighborhoods, one using a triangle mesh structure and the other using k-dimensional (k-d) trees, which have different properties near surface discontinuities. The third step is to fit the pose, curvatures, and boundary of a paraboloid patch to the neighborhood. Finally the patch is evaluated using two different metrics, and possibly discarded. We use both a Euclidean residual to quantify fit quality and a grid-based coverage algorithm to ensure that the patch is sufficiently representative of the actual data. The system has been implemented in the Surface Patch Library (SPL), with code available on our website [9].

Next we cover related work and review the key data structures. We then give the details of the algorithm including neighborhood searching, a new refinement to the fitting algorithm to avoid side-wall fits (Fig. 6), and the residual and coverage evaluation algorithms. Finally we present experimental results for 10 datasets acquired outdoors in natural rocky terrain plus one artificial dataset.

Regarding the choice of k-d tree vs triangle mesh for neighborhood searching, a new refinement to the fitting algorithm to avoid side-wall fits (Fig. 6), and the residual and coverage evaluation algorithms. Finally we present experimental results for 10 datasets acquired outdoors in natural rocky terrain plus one artificial dataset.

A. Related Work

The approach explored here contrasts with the traditional study of range image segmentation [10]–[12] where a dense partition of connected and non-overlapping but potentially irregularly bounded—and usually planar [13]–[15]—regions is generated. Irregularly bounded regions, which may be very large or non-convex, can present a challenge for higher-level contact planning algorithms which still need to search for specific contact areas within each region. One aim of using regularly bounded regions of approximately the same size as relevant contact features on the robot is to trade such
potentially complex continuous searches within patches to a discrete search across patches.

Modeling the environment and detecting potential contact surface areas for locomotion using exteroceptive sensing is a common task, but still very challenging. Grid-based systems like those on Ambler [16], Dante II [17], and MER [4] use laser scanners or stereo cameras to build an elevation map, find obstacles, and quantify traversability, but usually don’t model detailed 3D contact features.

Visual odometry and stereo cameras or range sensors have also been used on several current walking robots including BigDog [18], [19] and DLR Crawler [20], though again mainly for obstacle avoidance and traversability analysis, not detailed 3D foot placement or contact planning. Some steps have been made in that direction in [21], where terrain is modeled using a Gaussian Process, and in [5] where a continuous surface model is used for Little Dog locomotion. Also, several works use stereo vision or laser scanners to map planar environment segments for humanoids operating indoors in controlled conditions [15], [22].

In [23] texture synthesis was presented to deal with the problem of occluded terrain by filling in the missing portions. Our approach avoids representing such missing areas where uncertainty is high. We instead plan to integrate multiple range scans taken from different perspectives as the robot moves to fill in missing areas with new observations using a volumetric fusion approach [24], [25].

II. DATA STRUCTURES

Range sensors are now commonly available. Stereo and structured light sensors like the Kinect, time-of-flight cameras, and laser scanners produce clouds of 3D sample points of environment surfaces. Here we focus on point cloud data in the form of an image acquired from a single point of view, with \( N \) 3D sample points organized in an \( N_r \times N_c \) grid, \( N_r N_c = N \). Due to occlusions, limited sensor range, and other factors, there may be grid cells map where depth information is invalid. (Grid-organized data can also be synthesized from other representations, including from the volumetric fusion mentioned above, by raycasting.)

Our algorithm requires finding local neighborhoods of 3D points, for which spatial decompositions like k-dimensional (k-d) trees [26] are commonly used. Another common structure for representing 3D sample points of surfaces is the triangle mesh. We next present details of these two structures.
B. K-D Tree

One of the most common data structures for spatial points is the k-d tree [26]. Whereas the triangle mesh approach depends on the grid organization of the data, k-d trees can be constructed from any point cloud. However, k-d trees do not directly encode information about depth discontinuities.

The cost for building a k-d tree is \( O(N \log^2 N) \) when using an \( O(N \log N) \) sorting algorithm for computing medians, or \( O(N \log N) \) with a linear median-finding algorithm [26]. The cost for finding \( k \) nearest neighbors is \( \tilde{O}(k \log N) \).

III. SPARSE SURFACE MODELING

Using one of the above data structures for neighborhood finding, patches are fit either iteratively or in parallel to the environment surface samples using the following algorithm:

1) Pick a seed point on the surface (Sec. III-A).
2) Find its neighbors within distance \( r \) (Sec. III-B).
3) Fit a bounded patch to the neighborhood (Sec. III-C).
4) Evaluate the patch with residual and coverage metrics and discard it unless both checks pass (Sec. III-D).
5) Continue until termination (Sec. III-E).

The neighborhood size \( r \) is set to a fixed value (e.g. 5–10cm) derived from the size of the intended contact surface on the robot\(^2\). Pre-processing, like background removal and decimation, can be applied first depending on the application. Several termination checks may be applied, for example, checking if the sum of the patch areas has reached a given ratio of the total surface area.

Next we describe each step of the algorithm in detail.

A. Seed Selection

Here we use random seed selection as a baseline method and general task-independent approach. Figure 1 illustrates 50 randomly seeded patches. We are also considering some types of (possibly task-specific) salient points [28], including peaks and valleys, and the possibility to direct more samples in task-specific areas of interest.

B. Patch Neighborhoods

Finding nearest neighbors in a point cloud is a well-studied problem. Many methods have been introduced, including approximations. Two concepts of a neighborhood are: (a) \( k \) nearest neighbors, i.e. the \( k \) closest points to a seed; (b) all neighbors within distance \( r \) from the seed, for some distance metric. For fitting uniformly bounded patches we use the latter; the number of points \( k \) in the recovered neighborhood thus varies depending on \( r \) and the specifics of the distance metric and the search algorithm (Fig. 4).

\(^2\)Of course the robot may also make contact in other ways. The patch model could help plan intentional contacts while other data structures are simultaneously used for general collision prediction.

\(^3\)\( L \) is also the Darboux frame of the paraboloid.

\( r \) thus varies depending on the latter; the number of points within distance \( r \) \( k \) approximations. Two concepts of a neighborhood are: (a) all closest points to a seed; (b) all neighbors within distance \( r \) from the seed, pruning it when the chain distance exceeds \( r \). In that way we reduce the chances that the extracted neighbors cross discontinuities in the point cloud, even if they are spatially close (Fig. 2, right).

\( r \) is set to a fixed value (e.g. 5–10cm) derived from the size of the intended contact surface on the robot\(^2\). Pre-processing, like background removal and decimation, can be applied first depending on the application. Several termination checks may be applied, for example, checking if the sum of the patch areas has reached a given ratio of the total surface area.

Next we describe each step of the algorithm in detail.

A. Seed Selection

Here we use random seed selection as a baseline method and general task-independent approach. Figure 1 illustrates 50 randomly seeded patches. We are also considering some types of (possibly task-specific) salient points [28], including peaks and valleys, and the possibility to direct more samples in task-specific areas of interest.

B. Patch Neighborhoods

Finding nearest neighbors in a point cloud is a well-studied problem. Many methods have been introduced, including approximations. Two concepts of a neighborhood are: (a) \( k \) nearest neighbors, i.e. the \( k \) closest points to a seed; (b) all neighbors within distance \( r \) from the seed, for some distance metric. For fitting uniformly bounded patches we use the latter; the number of points \( k \) in the recovered neighborhood thus varies depending on \( r \) and the specifics of the distance metric and the search algorithm (Fig. 4).

\( \hat{T} \) is a point on the patch in frame \( 3 \) is defined by \( (r, t) \) is

\[ p_t^T K p_t - 2 p_t^T z = 0 \]

\[ K \triangleq \text{diag}(\kappa_x, \kappa_y, 0), \quad z \triangleq [0 \ 0 \ 1]^T \]

where \( p_t \in \mathbb{R}^3 \) is a point on the patch in frame \( L \).
**Patch Fitting:** A nonlinear fitting algorithm is used based on a variation of Levenberg-Marquardt iteration that fits bounded curved patches to \( N \) sample points \( q_i \in \mathbb{R}^3 \). The algorithm minimizes a sum-of-squares residual by optimizing the patch implicit and explicit parameters. The residual for an individual sample point \( q_i \) scales the value of the implicit form (1) by the inverse of a first-order estimate of its standard deviation, which is derived in turn from a covariance matrix modeling the sensor uncertainty for \( q_i \).

Type selection for the boundary curve is only partially automatic—rectangles and convex quads are only used if requested. We cover them here for completeness; only circle and ellipse boundaries are used in the experiments.

![Fig. 6](image)

**Fig. 6.** The reparameterization in Eq. (2) keeps the fitted paraboloid centered on the data (right). This prevents the “side-wall” effect (left) and helps ensure good coverage, but can compromise the Euclidean residual.

**Side-wall effect problem:** When the neighborhood points don’t have a central symmetry then they may be unevenly distributed in the patch when using the original fitting algorithm in [8]. We call this the side-wall effect (Fig. 6). To handle this we apply a constrained fitting where the center of the patch \( c \in \mathbb{R}^3 \) must lie on the line through the centroid \( c_p \) of the neighborhood parallel to the normal \( n_p \) to an initial fit plane. This is implemented as a reparameterization

\[
   c = c_p + \alpha n_p
\]

where \( \alpha \in \mathbb{R} \) is the new patch parameter replacing \( c \).

**D. Patch Evaluation**

The last part of the algorithm is the evaluation of the patch and the decision whether to keep it. We use two measures based on the residual and coverage. The first one evaluates the surface and the second one the boundary of the patch.

**Residual Evaluation:** The patch residual measures the deviation between the sample points and the (unbounded) paraboloid patch surface. We use the root-mean-square error (RMSE) Euclidean residual \( \varrho \) between the sample points \( q_i \) and their corresponding closest points \( p_i \) on the patch:

\[
   \varrho = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \|q_i - p_i\|^2}. \tag{3}
\]

(Here we consider both \( q_i \) and \( p_i \) to be expressed in the patch local frame \( L \).)

Whereas the patch fitting algorithm uses an algebraic residual for speed, \( \varrho \) is an Euclidean residual and gives its result in the same physical units as the input data (e.g. meters) [29], enabling it to be compared to a meaningful threshold. However, the required \( p_i \) must be calculated for each \( q_i \).

![Fig. 7](image)

**Fig. 7.** Left: Coverage evaluation for a bad (top) and a good (bottom) patch with bad cells in red. Right: Sorted residuals for 1000 random patches (see text), approximately 95% of which are below 0.01.

When \( \kappa_x \approx \kappa_y \approx 0 \) the paraboloid surface was fitted as a plane, so \( p_i = (I - \hat{z} \hat{z}^T)q_i \), i.e. \( p_i \) is the projection of \( q_i \) onto the \( xy \) plane of \( L \). Otherwise \( p_i \) is characterized as:

\[
   \min_{p_i \text{ satisfying } (1)} \|q_i - p_i\|. \tag{4}
\]

A solution can be found using Lagrange multipliers [30]. Define the Lagrange function \( \Lambda \) as:

\[
   \Lambda(p_i, \lambda) = (q_i - p_i)^T(q_i - p_i) + \lambda(p_i^TKp_i - 2p_i^T\hat{z}). \tag{5}
\]

with Lagrange gradient constraints

\[
   \nabla \Lambda(p_i, \lambda) = 0^T \Leftrightarrow \partial \Lambda / \partial p_i = [0 \ 0 \ 0] \text{ and } \partial \Lambda / \partial \lambda = 0. \tag{6}
\]

Expand the first gradient constraint from (6)

\[
   -2q_i^T + 2p_i^T + \lambda(2p_i^TK - 2\hat{z}^T) = [0 \ 0 \ 0]
\]

\[
   -q_i + p_i + \lambda(Kp_i - 2\hat{z}) = [0 \ 0 \ 0]^T
\]

\[
   (I + \lambda K)p_i = q_i + \lambda \hat{z}
\]

\[
   p_i = (I + \lambda K)^{-1}(q_i + \lambda \hat{z}) \tag{7}
\]

and substitute for \( p_i \) in the second gradient constraint, which is the same as (1). This leads to a fifth degree polynomial in \( \lambda \), for which there is at least one real solution because imaginary solutions come in pairs. Finally, backsubstitute\(^4\) the real solution(s) in (7) and find the minimum as in (4).

To determine the residual threshold \( T_r \) such that any patch with \( \varrho > T_r \) will be dropped, we sorted all residuals (Fig. 7, right) for a sampling of 1000 random patches \( (r = 0.1 \text{m, k-d tree neighborhoods}) \), 100 on each of our 10 rock datasets. The value \( T_r = 0.01 \text{m} \) was selected to include approximately 95% of the patches. In general \( T_r \) can be set in an application dependent way. Furthermore, the choice of RMSE residual is not essential. For example, an alternate residual

\[
   \varrho_\text{alt} = \max \|q_i - p_i\| \tag{8}
\]

could be used to check if any small surface bumps protrude more than a desired amount from the surface.

\(^4\)Division by zero can occur during this backsubstitution when \( q_i \) is on a symmetry plane or axis, but alternate forms can be used in those cases.
coverage evaluation: a different evaluation is needed to take into account the patch boundary. a patch may fit the data but still not faithfully represent the neighborhood, either because too many sample points are outside the patch boundary or there is too much area inside the boundary that is not supported by data points. (we opt not to speculatively fill holes in the data.)

to detect these cases we generate an axis-aligned grid of fixed pitch \( w \) on the \( xy \) plane the patch local frame \( l \). we generate only the required number of rows and columns in this grid to fit the projection of the patch boundary.

define \( i_0 \) and \( o_0 \) to be the number of data points whose \( xy \) projections are both inside a given cell and respectively inside or outside the projected patch boundary. define \( a_i \) to be the area of the geometric intersection of the cell and the projected patch boundary, which will be detailed below. the cell is considered bad if

\[
i_c < \frac{a_i}{w_c^2} t_i \text{ or } o_c > (1 - \frac{a_i}{w_c^2}) t_o.
\]

for thresholds \( t_i \) and \( t_o \). here we fix these thresholds relative to the expected number of samples \( n_c \) in a given cell if all samples were in-bounds and evenly distributed:

\[
t_i = \zeta_i n_c, \quad t_o = \zeta_o n_c, \quad n_c = \frac{k}{n_p}, \quad n_p = \frac{a_p}{w_c^2},
\]

where \( k \) is the number of sample points in the neighborhood and \( a_p \) is the area of the patch approximated as the area inside the projected boundary.

the patch fails coverage evaluation iff there are more than \( t_p \) bad cells. after some experiments in fitting paraboloid patches with \( r = 0.1m \), we set \( w_c = 0.01m, \zeta_i = 0.8, \zeta_o = 0.2, \) and \( t_p = 0.3n_p \). figure 7 (left) illustrates patches that pass and fail coverage evaluation.

intersection area for ellipse and circle boundaries: for an ellipse boundary with radii \( a, b \), or for the degenerate case of a circle boundary with radius \( r = a = b \), we compute the intersection area with a secant approximation since the exact computation involves a relatively expensive inverse trig function. wlog we describe only the top right quadrant (fig. 8, left); the other three are symmetric. let \( p_0...3 \) be the four corners of a grid cell in counter-clockwise order starting from the lower left. the algorithm for computing the intersection area is:

1) if \( p_2 \) is inside the ellipse then \( a_i = w_c^2 \)
2) else if \( p_0 \) is not inside the ellipse then \( a_i = 0 \)
3) else if \( p_1 \) is inside the ellipse then
   if \( p_3 \) is inside the ellipse then \( a_i = a_1 \) else \( a_i = a_2 \)
4) else if \( p_3 \) is inside the ellipse then \( a_i = a_3 \)
5) else \( a_i = a_4 \).

\[
a_1 = (x_b - x_0)w_c + (x_c - x_b)(y_b - y_0) + ((x_c - x_b)(y_0 + w_e - y(x_b))) / 2
\]
\[
a_2 = (x_c - x_0)(y(x_c) - y_0) + (x_c - x_0)(y(x_0) - y(x_c)) / 2
\]
\[
a_3 = (y_c - y_0)(x(y_c) - x_0) + (y_c - y_0)(x(y_0) - x(y_c)) / 2
\]
\[
a_4 = (x(y_0) - x_0)(y(x_0) - y(y_0)) / 2
\]

\[
x(y) \equiv a \sqrt{1 - y^2/b^2}, \quad y(x) \equiv b \sqrt{1 - x^2/a^2}
\]

\[
[p_0, y_0]^T \equiv [x_0, y_0] + [x_0 - x_0] \equiv [x_0, y_0] + [a - b]
\]

intersection area for axis-aligned rectangle boundary: as above we consider only the top right quadrant (fig 8, middle). let the rectangle half-lengths be \( a, b \) and define \( [x_0, y_0]^T \equiv p_0 \). the exact intersection area can be computed as follows:

1) if \( p_2 \) is inside the rect then \( a_i = w_c^2 \)
2) else if \( p_0 \) is not inside the rect then \( a_i = 0 \)
3) else if \( p_1 \) is inside the rect then \( a_i = a_5 = w_e(b - y_0) \)
4) else if \( p_3 \) is inside the rect then \( a_i = a_6 = w_c(a - x_0) \)
5) else \( a_i = a_7 = (a - x_0)(b - y_0) \).

intersection area for convex quadrilateral boundary: to handle the case of a general convex quadrilateral (fig. 8, right), we use the fact that the intersection between a convex quad and a rectangle is always convex:

1) find the set of grid cell corner points that are inside the quad and vice-versa.
2) find the intersection points between the grid cell boundaries and the convex quad boundaries.
3) discard all points from steps 1 and 2 except those that lie in or on both figures.
4) sort all the points computed in the previous steps in counterclockwise order and connect the first point with each of the others in order, forming a triangle fan. \( a_i \) is the sum of the triangle areas.

e. termination criteria

various termination criteria can be used, for example: wall-clock time, total number of patches, or task-specific criteria. another approach is to specify a desired fraction \( \nu \) of the total sampled surface area \( s \) that should probabilistically be covered by patches. note that \( \nu \) can be both less than 1, to sample sparsely, or more than 1, to oversample. with \( r \)-ball neighborhood search (using either k-d tree or triangle mesh) and ellipse-bounded paraboloid patch fitting we can estimate the expected number of patches for this criteria as

\[
\nu \frac{s}{\pi r^2}.
\]

or, as we do in the experiments below, we can fit patches until the sum of their areas reaches or exceeds \( \nu s \).
In practice it is nontrivial both to calculate the total sampled surface area $S$ and the area of any individual patch. To approximate $S$ we compute the triangle mesh (Sec. II-A) and sum the triangle patches. We approximate the area of an individual patch as the area of the projection of its boundary on the $xy$ plane of the patch local frame $L$.

### IV. EXPERIMENTAL RESULTS

We performed experiments to test the overall approach and to compare the triangle mesh and k-d tree data structures for neighborhood finding. 11 datasets were collected with the Kinect at $640 \times 480$ resolution and then decimated by 2. The first 10 are scenes of natural rocky terrain (Fig. 11) acquired with a hand-held Kinect outdoors on an overcast day (the Kinect does not work in direct sunlight). The last is taken in the lab with synthetic rocks (Fig. 1). The first 10 are scenes of natural rocky terrain (Fig. 11) acquired with a hand-held Kinect outdoors on an overcast day (the Kinect does not work in direct sunlight). The last is taken in the lab with synthetic rocks (Fig. 1).

The parameters were: neighborhood radius $r = 0.1 m$, residual threshold $T_r = 0.01 m$, coverage cell size $w_c = 0.01 m$, coverage threshold factors $\xi = 0.8$, $\zeta = 0.2$, and $T_p = 0.3 A_p / w_c^2$ (all motivated above). We let the algorithm run for each dataset until the sum of the patch areas equaled or exceeded 90% of the sampled surface area, both approximated as described in Sec. III-E. Our current Matlab implementation is not optimized and takes about 70ms per patch. We aim to improve this for real-time use by optimization and parallelization.

Qualitatively, as depicted in Figure 9 and 10, the algorithm appears to give a reasonable representation of non-smooth environment surfaces. Quantitatively, we measured the following statistics (Table I): the total number of patches before evaluation, the number of valid patches passing both residual and coverage evaluation, the number of dropped patches for each test, the average Euclidean residual (of the valid patches), and the total surface area for each dataset.

<table>
<thead>
<tr>
<th>Data</th>
<th>Structure</th>
<th>Patches total</th>
<th>Dropped patches due to residual total</th>
<th>Average residual ($mm$)</th>
<th>Total area ($m^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rock 1</td>
<td>k-d tree</td>
<td>167</td>
<td>10</td>
<td>4.6</td>
<td>4.57</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>164</td>
<td>7</td>
<td>4.3</td>
<td>4.57</td>
</tr>
<tr>
<td>rock 2</td>
<td>k-d tree</td>
<td>160</td>
<td>18</td>
<td>5.0</td>
<td>3.13</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>185</td>
<td>24</td>
<td>4.0</td>
<td>3.13</td>
</tr>
<tr>
<td>rock 3</td>
<td>k-d tree</td>
<td>231</td>
<td>8</td>
<td>5.2</td>
<td>5.53</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>227</td>
<td>7</td>
<td>4.7</td>
<td>5.53</td>
</tr>
<tr>
<td>rock 4</td>
<td>k-d tree</td>
<td>215</td>
<td>8</td>
<td>5.0</td>
<td>5.04</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>181</td>
<td>5</td>
<td>4.8</td>
<td>5.01</td>
</tr>
<tr>
<td>rock 5</td>
<td>k-d tree</td>
<td>195</td>
<td>17</td>
<td>5.4</td>
<td>4.86</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>188</td>
<td>5</td>
<td>5.1</td>
<td>4.86</td>
</tr>
<tr>
<td>rock 6</td>
<td>k-d tree</td>
<td>235</td>
<td>31</td>
<td>5.7</td>
<td>5.69</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>273</td>
<td>20</td>
<td>5.3</td>
<td>5.69</td>
</tr>
<tr>
<td>rock 7</td>
<td>k-d tree</td>
<td>267</td>
<td>30</td>
<td>4.4</td>
<td>6.54</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>226</td>
<td>21</td>
<td>4.2</td>
<td>6.54</td>
</tr>
<tr>
<td>rock 8</td>
<td>k-d tree</td>
<td>256</td>
<td>16</td>
<td>4.2</td>
<td>7.04</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>231</td>
<td>23</td>
<td>3.9</td>
<td>7.04</td>
</tr>
<tr>
<td>rock 9</td>
<td>k-d tree</td>
<td>187</td>
<td>13</td>
<td>4.8</td>
<td>4.95</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>198</td>
<td>12</td>
<td>4.6</td>
<td>4.95</td>
</tr>
<tr>
<td>rock 10</td>
<td>k-d tree</td>
<td>301</td>
<td>30</td>
<td>5.0</td>
<td>6.98</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>236</td>
<td>27</td>
<td>4.7</td>
<td>6.98</td>
</tr>
<tr>
<td>rock average</td>
<td>k-d tree</td>
<td>222</td>
<td>176</td>
<td>4.9</td>
<td>5.45</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>226</td>
<td>176</td>
<td>4.6</td>
<td>5.43</td>
</tr>
<tr>
<td>fake</td>
<td>k-d tree</td>
<td>65</td>
<td>18</td>
<td>3.8</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>tri mesh</td>
<td>75</td>
<td>21</td>
<td>3.8</td>
<td>0.50</td>
</tr>
</tbody>
</table>

**TABLE I**

There are generally more patches dropped due to residual for k-d tree neighborhoods, possibly because the triangle mesh neighborhoods avoid discontinuities which may not be fit well by a paraboloid. We see the opposite effect for patches dropped due to coverage: more patches are generally dropped due to insufficient coverage when using triangle mesh neighborhoods. The k-d tree neighborhoods may distribute samples more evenly, particularly near discontinuities.

Another interesting result is that more patches are required to reach 90% of the surface area when using triangle mesh neighborhoods. In Figure 10 we see that the distribution of patch areas created using mesh neighborhoods is skewed more to the low side than those created using k-d tree neighborhoods. This can again be explained by the fact that k-d tree neighborhoods will span discontinuities but remain roughly circular, whereas triangle mesh neighborhoods may be less circular when the seed point is near an edge.

### V. CONCLUSIONS AND FUTURE WORKS

Paraboloid patches with one or two curvatures and regularly shaped bounds, sized to match contact surfaces on a robot, are an alternative to traditional segmentations for representing 3D contact surfaces in unstructured environments. The set of bounded patch models and the fit-and-validate strategy described here could be the basis for such a system. We investigated several aspects of the system design, in particular, the choice of data structures for neighborhood searching and the type of validation tests to use.

With respect to the relative merits of k-d tree vs triangle mesh data structures, our experiments indicate that both approaches are competitive. K-D trees do not directly encode discontinuity information, resulting in more patches dropped...
due to large residual, but also produce more consistently sized neighborhoods than triangle mesh. Though we use grid-organized samples here, general point clouds can be easier to handle with k-d trees than with triangle mesh. Preliminary timing results also suggest that the larger theoretical time complexity for k-d trees is not a significant factor in practice.

We are now optimizing and exploring adding seed points at salient features to ensure that task-relevant areas are sampled. We are also planning to integrate sparse patch-based modeling with dense fusion of range data [25], with the goal of maintaining a dynamic “patch map” of relevant contact surfaces around a robot. We are intending to use these methods in a new perceptual system for bipedal locomotion on rough terrain (see [8], Figure 4) where individual foot contact surfaces are located visually before stepping.

This material is based upon work supported by the National Science Foundation under Grant No. 1149235.

REFERENCES


