Shape Segmentation Using Local Slippage Analysis

Conference Paper · January 2004

DOI: 10.1145/1057432.1057461 · Source: DBLP

2 authors:

N. Gelfand
Fermi National Accelerator Laboratory (Fermilab)
59 PUBLICATIONS 2,670 CITATIONS
SEE PROFILE

Leonidas J. Guibas
Stanford University
554 PUBLICATIONS 32,363 CITATIONS
SEE PROFILE

Eurographics Symposium on Geometry Processing (2004)
R. Scopigno, D. Zorin, (Editors)

Shape Segmentation Using Local Slippage Analysis

Natasha Gelfand and Leonidas J. Guibas

Computer Graphics Laboratory, Stanford University

Abstract

We propose a method for segmentation of 3D scanned shapes into simple geometric parts. Given an input point cloud, our method computes a set of components which possess one or more slippable motions: rigid motions which, when applied to a shape, slide the transformed version against the stationary version without forming any gaps. Slippable shapes include rotationally and translationally symmetrical shapes such as planes, spheres, and cylinders, which are often found as components of scanned mechanical parts. We show how to determine the slippable motions of a given shape by computing eigenvalues of a certain symmetric matrix derived from the points and normals of the shape. Our algorithm then discovers slippable components in the input data by computing local slippage signatures at a set of points of the input and iteratively aggregating regions with matching slippable motions. We demonstrate the performance of our algorithm for reverse engineering surfaces of mechanical parts.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational geometry and object modeling

1. Introduction

Reverse engineering applications deal with reconstructing a CAD model from an unstructured input dataset such as one that may come from a laser scanner. A significant problem in reverse engineering is the segmentation of the input dataset into a set of regions, such that each region can be approximated by a single simple surface. Segmentation is usually followed by surface fitting, where each component is approximated by best fitting parametric surface. The problems of segmentation and surface fitting are closely related: if we know the surfaces, we can segment the input pointset by grouping together points that lie within a threshold of the same surface. On the other hand, given a segmentation of the pointset into components, we can find the approximating surfaces by finding the best fitting surface for each component. For general objects, segmentation and surface fitting problems often require user input. However, in many CAD applications, the underlying model is composed of simple surfaces such as spheres, planes, cylinders and surfaces of revolution. In such cases, automatic segmentation and surface fitting are often possible.

There is a large body of research dealing with shape segmentation. A general survey of segmentation and surface fitting in reverse engineering of CAD objects can be found in [VMC97]. Most automatic segmentation methods fall into two general categories. Bottom-up, or region growing techniques, start with a set of seed points for which some local surface characteristics are computed. New points are then added to the seed regions as long as the computed surface characteristic does not change. The other approach to automatic segmentation is to proceed top-down. The original pointset is recursively subdivided until each subset belongs to a single component. This approach is common in image segmentation [SM00], however in model segmentation most techniques tend to use the bottom-up method, e.g. [SB95, BMV01]. For surface fitting of general shapes, the segmentation problem is generally difficult, and the user is often asked to indicate rough component boundaries [KL96], which are then refined and approximated with parametric patches. In Computer Graphics, automatic methods for segmentation of arbitrary shapes are often used for generating base mesh domains for multiresolution analysis and texture mapping. Such methods are generally based on generating regions that satisfy specific distance and planarity constraints [KT03, GWH01, BM03, SSGH02, LPRM02].

In this paper we develop a bottom-up segmentation algorithm that can be applied to engineering-type objects. The algorithm recognizes in the input simple surfaces such as planes, cylinders, spheres, surfaces of revolution and surfaces of linear extrusion (also known as kinematic surfaces). The algorithm is based on novel differential surface characteristic called *slippage signatures*. The slippage signature of a shape consists of the set of rigid motions that, when applied to the shape, slide the transformed copy along the original without forming any gaps. We show how to compute this surface characteristic by analyzing the Hessian matrix of a certain minimization problem. We then develop a surface comparison method that uses the slippage signatures to decide when two pointsets belong to the same surface. We demonstrate the performance of our segmentation algorithm on a variety of pointsets representing mechanical parts.

2. Shape classification through slippable motions

2.1. Definition of slippable motions

Our surface descriptor is based on how the surface behaves under different kinds of rigid motions. A rigid motion $M(t)$ consists of two time-varying components: $R(t) \in SO(3)$, which determines the rotational part of the motion and $T(t)$ which is the translational component. At time t, position of a point $\mathbf{x}_0$ moving according to M is given by

$$\mathbf{x}(t) = R(t) \cdot \mathbf{x}_0 + T(t). \quad (1)$$

At a given time instance, the motion at a point $\mathbf{x}$ is linear, and the instantaneous velocity vector of $\mathbf{x}$, obtained by differentiating Equation 1, is given by

$$\mathbf{v}(\mathbf{x}) = \mathbf{r} \times \mathbf{x} + \mathbf{t}, \quad (2)$$

where $\mathbf{r} = (r_x, r_y, r_z)$ is a 3×1 vector of rotations around x, y, and z axes and $\mathbf{t} = (t_x, t_y, t_z)$ is a translation vector.

There are three kinds of motions where $\mathbf{r}$ and $\mathbf{t}$ are constant over time:

- If $\mathbf{r} = 0$, the motion M is a translation with constant velocity along the direction $\mathbf{t}$.
- If $\mathbf{r} \cdot \mathbf{t} = 0$, M is a rotation with constant angular velocity.
- If $\mathbf{r} \cdot \mathbf{t} \neq 0$, M is a uniform helical motion.

Given a surface S we call a rigid motion M a *slippable motion of* S if the velocity vector of each point $\mathbf{x} \in S$ is tangent to S at $\mathbf{x}$. If the instantaneous motion of each point is tangential, it means that the distance between the transformed surface and the original does not change, to first order. The surface can be thought of as sliding against itself, without forming any gaps between the moving surface and the original copy. That is, the surface S is invariant under its slippable motions. Surfaces which are invariant under one of the types of rigid motions described above are known as kinematic surfaces [PW01]. One can differentiate between kinematic surfaces that are generated by rotational, translational or helical motions [PR98, Sri03].

A kinematic surface can be slippable in more than one way. The simplest example is a plane. Any translational motion along the plane is slippable, as is rotational motion around the plane's normal. A more interesting kinematic shape is a cylinder, for which slippable motions include rotations around the cylinder's axis and translations along the axis. Other kinematic shapes include spheres, helical surfaces, surfaces of revolution and surfaces of linear extrusion (translationally slippable surfaces).

2.2. Computing slippable motions

The goal of our segmentation is to break up the input pointset into component pointsets such that each component can be well approximated by a connected piece of a single kinematic surface. To differentiate between surfaces and pointsets, we will call a set of points P *slippable* if it can be approximated by some kinematic surface. Our segmentation algorithm is based on breaking up the input data into slippable components. First, we show how to determine if a given surface S is a kinematic surface.

Let $\mathbf{x}$ be a point belonging to the surface S, and let $\mathbf{n}$ be the vector normal to S at $\mathbf{x}$. We will examine how $\mathbf{x}$ is affected by an instantaneous motion whose parameters are given by the 6-vector $[\mathbf{r}\ \mathbf{t}]$. The amount of non-tangential motion is given by the dot product of the velocity vector with the normal at the point $\mathbf{x}$:

$$\mathbf{v}_{perp} = (\mathbf{r} \times \mathbf{x} + \mathbf{t}) \cdot \mathbf{n}. \quad (3)$$

The required condition for instantaneous motion $[\mathbf{r}\ \mathbf{t}]$ to be slippable is that the motion of each point is tangential to the surface at that point. This can be written as:

$$\int_S ((\mathbf{r} \times \mathbf{x} + \mathbf{t}) \cdot \mathbf{n})^2 dS = 0. \quad (4)$$

We assume that the input data is given by a pointset P of n points that have been sampled from some underlying surface S. Our goal is to determine if S is a kinematic surface and find its slippable motions. Each point $\mathbf{p}_i \in P$ is given by a 3- vector of its coordinates $\mathbf{p}_i = (p_{ix}, p_{iy}, p_{iz})$. We also assume that at each point $\mathbf{p}_i$ we have a corresponding normal $\mathbf{n}_i = (n_{ix}, n_{iy}, n_{iz})$ that approximates the normal vector to the surface S. If the input pointset is given as a mesh, we can use the triangles around $\mathbf{p}_i$ to estimate the normal. If no connectivity information is given with the input, the normals can be estimated by plane fitting using the technique described in [MNG04].

We can find the slippable motions of P (and correspondingly S) by posing Equation 4 as a minimization problem. We want to find an instantaneous motion vector $[\mathbf{r}\ \mathbf{t}]$ that, when applied to P minimizes the motion along the normal direction at each point.

$$\min_{[\mathbf{r}\ \mathbf{t}]} \sum_{i=1}^{n} ((\mathbf{r} \times \mathbf{p}_i + \mathbf{t}) \cdot \mathbf{n}_i)^2. \quad (5)$$

Not surprisingly, the same minimization problem arises in the context of pointset registration [CM91, RL01]. If we think of the pointset P as having two copies, a moving version P_T and a stationary version P_O, Equation 5 minimizes the point-to-plane error metric of Chen and Medioni [CM91] between the transformed and stationary pointset. A slippable motion is the one where the point-to-plane distance between P_T and P_O is zero [GIRL03]. Pottman [PR98, PHOW04] determines if a given pointset is sampled from a kinematic surface by analyzing the normals of P in line-space, which leads to a similar minimization problem.

Equation 5 is a least-squares problem whose minimum is the solution of a linear system $C\mathbf{x} = 0$, where C is a (covariance) matrix of second partial derivatives of the objective function with respect to the motion parameters.

$$C = \sum_{i=1}^{n} \begin{bmatrix} c_{ix}c_{ix} & c_{ix}c_{iy} & c_{ix}c_{iz} & c_{ix}n_{ix} & c_{ix}n_{iy} & c_{ix}n_{iz} \\ c_{iy}c_{ix} & c_{iy}c_{iy} & c_{iy}c_{iz} & c_{iy}n_{ix} & c_{iy}n_{iy} & c_{iy}n_{iz} \\ c_{iz}c_{ix} & c_{iz}c_{iy} & c_{iz}c_{iz} & c_{iz}n_{ix} & c_{iz}n_{iy} & c_{iz}n_{iz} \\ n_{ix}c_{ix} & n_{ix}c_{iy} & n_{ix}c_{iz} & n_{ix}n_{ix} & n_{ix}n_{iy} & n_{ix}n_{iz} \\ n_{iy}c_{ix} & n_{iy}c_{iy} & n_{iy}c_{iz} & n_{iy}n_{ix} & n_{iy}n_{iy} & n_{iy}n_{iz} \\ n_{iz}c_{ix} & n_{iz}c_{iy} & n_{iz}c_{iz} & n_{iz}n_{ix} & n_{iz}n_{iy} & n_{iz}n_{iz} \end{bmatrix} \tag{6}$$

where $c_{ik} = (p_i \times n_i)_k$. Therefore, the slippable motions of P are those that belong to the null space of C. To compute the actual motion vectors, we compute the eigenvalue decomposition $C = X\Lambda X^T$. Eigenvectors of C whose corresponding eigenvalues are 0 correspond to the slippable motions of the pointset P. In practice, due to noise C is likely to be full rank. In this case, the slippable motions are those eigenvectors of C whose eigenvalues are sufficiently small. Figure 1 shows examples of slippable shapes and their corresponding slippable motions.

3. Segmentation into slippable components

In Section 2.2 we showed how to determine slippable motions of a pointset P. Based on the number and type of slippable motions, we can classify the pointset as being sampled from a surface that is spherical, planar, cylindrical, helical, surface of revolution or surface of linear extrusion. In this section, we develop an algorithm that segments pointsets into slippable components.

3.1. Point classification

We cannot apply the method of Section 2.2 to the input pointset P directly, since P as a whole may not be slippable. Our goal is to discover a decomposition of P into $P_1, P_2, \ldots, P_k$ such that each P_i is large, connected, and slippable.

Our approach falls into the class of bottom-up segmentation algorithms. We start by computing for each point in the input a guess at what kind of kinematic surface it was sampled from. For each point $\mathbf{p}_i \in P$ we form a neighborhood P_i of m points around $\mathbf{p}_i$. This forms our original set of components. If the input data is given with the connectivity information, e.g. as a mesh, we can build each P_i by crawling the mesh structure outward from $\mathbf{p}_i$ until m points are encountered. If the input is given as a point cloud, we just take the m nearest neighbors of $\mathbf{p}_i$.

Next, we compute the covariance matrix C_i of points in P_i according to Equation 6. We make two modifications to the basic equation to make the computation more numerically stable. First, we shift all points in P_i so that the center of mass lies at the origin of the coordinate system. Second, we uniformly scale the points so that the average radius of the patch is 1. These steps do not change the slippable motions of the pointset, but ensure that the magnitude of the $\mathbf{p}_i \times \mathbf{n}_i$ term is comparable with the $\mathbf{n}_i$ term in the computation, making the computation more numerically robust.

The next step is to decide how many slippable motions the neighborhood around $\mathbf{p}_i$ has. Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_6$ be the eigenvalues of C_i and $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_6$ be the corresponding eigenvectors. We call the eigenvalue λ_j "small" if the ratio $\frac{\lambda_6}{\lambda_j}$ is greater than a given threshold g (we use a value between 100 and 300 in our implementation). If k is the number of small eigenvalues of P_i, we call the eigenvectors $\mathbf{x}_1, \ldots, \mathbf{x}_k$ the *slippage signature* of $\mathbf{p}_i$. We write the slippage signature in matrix form as $X_{1\ldots k}$, with the eigenvectors corresponding to slippable motions arranged in columns.

The actual segmentation proceeds by aggregating neighboring points into slippable components. Originally, the neighborhood around each point $\mathbf{p}_i$ is treated as a separate patch P_i. Each patch has a covariance matrix C_i and a slippage signature $X^i_{1\ldots k}$. Notice that a patch may not have any slippable motions (if all eigenvalues of C_i are large), in which case its slippage signature is empty. The algorithm proceeds as follows:

1. *Initialization:* Compute a similarity score between each pair of adjacent patches. In the case of mesh input, adjacency is easy to determine. In the case of point cloud input, two patches are adjacent if they share any vertices. The similarity score is based on both the number and the compatibility of the slippable motions of the two patches. We use a priority queue to store the patch pairs, with the pair that has the best similarity score at the top of the queue.
2. *Patch growing:* At each step, we select a pair of adjacent patches that are the most similar and collapse them into a single patch. The new covariance matrix is computed from the covariance matrixes of the two patches to obtain the slippage signature for the merged patch. We then update the similarity score between the new patch and its neighbors.
3. *Termination:* Stop aggregating when the similarity score of the pair of patches at the top of the queue drops below a threshold. We apply a cleaning step to remove any small patches that may have remained. The resulting set of patches is the segmentation of the pointset. Each slip-

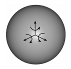
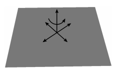
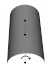
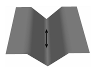
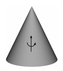
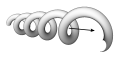
Num. small eigenvalues	Type of eigenvectors	Type of Surface	
3	3 rotations	sphere	
3	2 translation, 1 rotation	plane	
2	1 translation, 1 rotation	cylinder	
1	translation	linear extrusion	
1	rotation	surface of revolution	
1	helical motion	helix	

Figure 1: *Kinematic surfaces. For each surface, we indicate the number of small eigenvalues of the covariance matrix in Equation 6 and the type of the corresponding eigenvectors. Notice, that the eigenvectors given are only one possible set of slippable motions for that shape. Any motion that is a combination of the slippable eigenvectors is also slippable.*

pable patch can be approximated by a single kinematic surface.

We now examine the steps of the above algorithm in more detail.

3.2. Similarity score

Given two patches P_i and P_j (these can be either single points, whose slippage signature is computed from an initial neighborhood, or merged patches during the running of the segmentation algorithm), we say that they belong to the same component if:

- Their corresponding covariance matrixes C_i and C_j have the same number of small eigenvalues.
- The corresponding slippage signatures are the same, that is we can express the slippable motions of P_i as a combination of slippable motions of P_j and vice versa.

As described in Section 2, we call λ_k a small eigenvalue if $\frac{\lambda_6}{\lambda_k} > g$ for some minimum "condition number" g. The number of slippable motions for a patch P_i is given by the largest k for which the above condition holds:

$$s(P_i) = \text{argmax}_k\{\frac{\lambda_6}{\lambda_k} > g\} \qquad (7)$$

This means that the distance between the moving and the stationary copy of the patch P_i changes as least g times slower in the direction of slippable motions than any other motions. For a non-degenerate patch (i.e. not a curve), the maximum number of slippable motions is 3. We call a pointset with k small eigenvalues *k-slippable*. Figure 2 shows a simple object whose points are colored according to the number of small eigenvalues in a region around each point. Notice that the planar and the spherical regions are colored the same. This means that we cannot use just the number of small eigenvalues as the surface descriptor for segmentation, we need to look at the corresponding slippable motions as well.

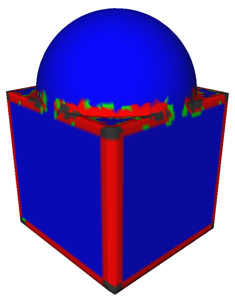


Figure 2: *Coloring points of a shape consisting of kinematic surfaces based on the number of small eigenvalues of the region around the point. Points whose neighborhoods are one-slippable are colored red, and whose neighborhoods are three-slippable are colored blue. Gray regions correspond to points in neighborhoods with no slippable motions, while green points are incorrectly classified as being two-slippable (See Section 4). The width of the one-slippable regions depends on the size of the initial neighborhood around each point (set to 10 points in this example).*

The first test for similarity between patches P_i and P_j rejects the patch pairs for which the number of slippable motions computed according to Equation 7 is different.

For the second similarity test, let $X_{1...k}$ and $Y_{1...k}$ be the slippage signatures of patches P_i and P_j respectively and let k be the number of slippable components of each patch. Since the first test was successful, k is the same for both patches. Each component of the slippage signature is a 6×1 vector corresponding to a rigid motion. Two slippage signatures are compatible if the rigid motions of one can be expressed as combination of rigid motions of the other.

In general, deciding if a given rigid motion M can be expressed as a combination of other rigid motions $M_1 \dots M_k$ is a difficult problem. The space of all rigid motions of $\Re^3$, $SE(3)$, is a curved manifold, which means simple interpolation techniques cannot be applied to rigid motions [Ale02, PR97]. In our case, however, we are dealing with instantaneous rigid motions, since the eigenvectors of the matrix C correspond to velocities. This means that the eigenvectors of C lie in the tangent space of $SE(3)$, which is flat. As a result we can treat the components of the slippage signatures as vectors in $\Re^6$. The columns of each slippage signature form an orthogonal basis for the space of all instantaneous slippable motions of the corresponding pointset. To answer if two slippage signatures $X_{1...k}$ and $Y_{1...k}$ are compatible, we just need to test if each column $X_{1...k}$ can be expressed as a linear combination of columns of $Y_{1...k}$.

Because of noise in the data we will never be able to perfectly express the slippable motions of P_i in terms of the slippable motions of P_j. Therefore, we need to look at the residual after the approximation. The amount by which two slippage signatures are dissimilar is given by the $(k+1)$st singular value of the combined matrix $[X_{1...k}Y_{1...k}]$. In the actual implementation we need to transform the rigid motions of one patch into the coordinate system of the other since we applied a shift and scaling in the computation of the covariance matrix.

We combine the two similarity tests into one similarity score as follows:

$$Sim(P_i,P_j) = \begin{cases} 0 & \text{if } s(P_i) \neq s(P_j) \\ F(\sigma_{k+1}) & \text{where } k = s(P_i) \text{ otherwise.} \end{cases} \tag{8}$$

where s is computed according to Equation 7, σ_{k+1} is the $(k+1)$st singular value of the combined matrix $[X_{1...k}Y_{1...k}]$, and F is described below.

We would like the similarity score to increase as the patches become more similar, so the function F maps small singular values into high similarity scores. We also use F to map the similarity scores into the range of values between 0 and 1. F is a Gaussian centered around 0, whose width determines how different slippage signatures of two patches that are considered similar can be.

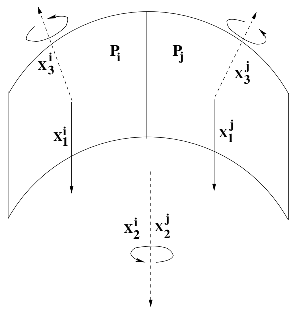


Figure 3: *Two patches that are part of a shallow cylinder incorrectly classified as having three slippable motions. The first two eigenvectors, which correspond to translation along the cylinder (direction of translation indicated as solid arrow) and rotation around the cylinder's axis (indicated with dashed arrow) match for both patches. Since the cylinder is shallow, rotation around the plane of the patches (indicated with dashed arrow) also has a small eigenvalue, but the corresponding eigenvectors do not match. Such classification errors are the motivation for our multi-pass algorithm.*

4. Robust implementation

Using Equation 8, we can now assign a similarity score to each pair of patches and run our clustering algorithm. Since the width of the Gaussian in F controls how fast the similarity score drops when the slippage signatures become different, we can terminate the algorithm when the similarity score of the next candidate pair of patches drops too low. However, using Equation 8 in its present form will result in poor segmentation due to a number of robustness issues, largely due to incorrectly determining the number of slippable motions of a patch. We now present an algorithm that is tolerant to such mistakes.

4.1. Multi-pass segmentation algorithm

Incorrectly determining the number of small eigenvalues of a patch can affect the similarity score in two ways. First, if we set g in Equation 7 too high, we can miss some slippable motions for a patch and decide that two patches are incompatible because they have a different number of small eigenvalues. If we set g too low, we can pick eigenvectors that are not slippable as part of the slippage signature. This can make patches incompatible because the non-slippable motions of the patches are not likely to match. Therefore, the performance of our algorithm depends on how well we identify the number of slippable components of each patch.

In practice, it turns out that it is difficult to correctly determine the number of slippable motions of a patch, especially in the early iterations of the algorithm when the patches are still small, and in the presence of noise in the data. The reason for this is that a small neighborhood around a point generally looks planar, e.g. looking around a point we cannot tell

Algorithm 1 Multi-pass segmentation of a point set P into slippable components.

```
1:  for each point $\mathbf{p}_i \in P$ do
2:      Form patch $P_i$ of $m$ vertices (using mesh crawling or nearest neighbors).
3:      Form the covariance matrix $C$ according to Equation 6 and compute its eigenvector decomposition $C = X\Lambda X^T$.
4:  end for
5:  $k \leftarrow 3$
6:  while $k \neq 0$ do
7:      for each pair of adjacent patches $(P_i, P_j)$ do
8:          Form combined matrix of slippage signatures, $[X_{1...k} Y_{1...k}]$, as described in Section 3.2.
9:          Compute $(k+1)$st singular value $\sigma_{k+1}$ of $[X_{1...k} Y_{1...k}]$.
10:         Compute $Sim(P_i, P_j)$ according to Equation 9.
11:     end for
12:     Initialize priority queue to empty.
13:     Insert all pairs of adjacent patches into the priority queue in order of decreasing similarity score.
14:     $(P_i, P_j) =$ EXTRACTMAX$(pqueue)$
15:     while $Sim(P_i, P_j) >$ MINSIMILARITY do
16:         $P_{ij} =$ MERGE$(P_i, P_j)$
17:         for $P_k \in neighbors(P_i) \cup neighbors(P_j)$ do
18:             Remove pair $(P_k, P_i)$ (correspondingly $P_j$) from the priority queue.
19:             Compute $Sim(P_k, P_{ij})$ according to Equation 9 and insert pair $(P_k, P_{ij})$ into the priority queue.
20:         end for
21:         $(P_i, P_j) =$ EXTRACTMAX$(pqueue)$
22:     end while
23:     $k \leftarrow k - 1$
24: end while
```

if the point should belong to a plane or a cylinder of large radius. In general we cannot reliably classify which shape the neighborhood belongs to until it grows sufficiently large. But growing the patch depends on merging it with its neighbors, which is in turn based on comparing the slippage signatures. For example, if we try to treat two patches that are part of the cylinder as planar, we are likely to get a bad similarity score from their slippage signatures, since only two of the three eigenvectors in each slippage signature are going to match well (See Figure 3). This is the general "chicken and egg problem" of segmentation [VMC97]: we cannot make a decision about a pointset until we know what shape it belongs to, that is until we have segmented the input. Therefore, we will need to make our algorithm robust against misclassifying shapes at the early stages of segmentation.

Our solution is based on the observation that any k-slippable shape is also $(k-1)$-slippable. Therefore, instead of comparing just the k-column slippage signatures of the two patches, we should also compare the slippage signatures made from the first $(k-1)$ eigenvectors etc. The best of the (at most three) similarity scores is assigned as the merge score for the two patches. In the case of misclassifying the cylindrical patches as planar, the first two eigenvectors of each patch will form the basis for the slippable motions of the cylinder, while the third will be the one that belongs to the plane (rotation around the normal at the center of the patch). In this case, comparing the slippage signatures with $k = 2$ will give a high similarity score.

However, picking too few components as slippable motions of a patch can result in undesirable segmentation. To illustrate, suppose we have a shape consisting of a cone and a cylinder, which share an axis of revolution. There are two valid segmentations into slippable components: both the cone and the cylinder are in one component, which is one-slippable, with the rotational motion; or there are two components, a two-slippable cylinder and a one-slippable cone. The input clearly consists of two different geometric shapes, so we would like the segmentation determined by our algorithm to reflect that.

We will do the segmentation in several passes. First, we try to merge together all three-slippable components. There may be many patches that are classified as three-slippable because the patch size is not large enough to correctly determine the number of slippable motions. However, the only patches that will be merged are those whose slippable motions are compatible, which are the patches that belong to planar and spherical components of the input. In the second pass, we merge all patches that are classified by the size of their eigenvalues as *at least two-slippable*, i.e. we allow a patch to "drop" a slippable motion. In comparing the slippage signatures of such patches, we only use the first two eigenvectors of each covariance matrix as the slippage signature (i.e. the largest slippable eigenvector is dropped). This handles the case of two-slippable patches being classified as planar. In the final pass, we merge all patches that are at least one-slippable, using the first eigenvector as the slippage signature for each patch. In practice, we repeat this process several times, every time making the width of the Gaussian in F larger to accommodate more noise as patches become larger. Finally, since we tend to misclassify patches more often at the early stage of the segmentation, we do not allow patches that consist of a large number of points to drop slippable motions (this prevents the case of merging the cone with the cylinder as described above). Our new similarity score is as follows:

$$Sim(P_i, P_j) = F(\sigma_{k+1}) \cdot G(P_i, k) \cdot G(P_j, k). \qquad (9)$$

Here, the function G acts as a confidence multiplier for the similarity score. The simplest form of G is just a cutoff. At each iteration of the algorithm, let k be the minimum number of slippable components that we are considering. Then

$$G(P_i,k) = \begin{cases} 1 & \text{if } \frac{\lambda_6}{\lambda_k} \geq g \text{ and } \frac{\lambda_6}{\lambda_{k+1}} < g \\ 1 & \text{if } \frac{\lambda_6}{\lambda_{k+1}} \geq g \text{ and } |P_j| < N \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

Here N is the desired size of individual segmented components. Notice that G also prevents the algorithms from merging patches which have different number of small eigenvalues, so we do not need to explicitly test for the number of small eigenvalues of a patch. Algorithm 1 contains the pseudocode for the multi-pass segmentation algorithm.

4.2. Initial patch size selection

The last important parameter that affects the robustness of our algorithm is the size of the initial neighborhood formed around each patch. As described above, we do not need to reliably determine the number of small eigenvalues of the patch, but if two patches belong to the same component, we would like the corresponding eigenvectors to match within the threshold set by F.

In practice, we noticed that the algorithm is most sensitive to the value of initial patch size m, as opposed to the value of the condition number g. Therefore, selecting the right patch size is important for good segmentation.

Unfortunately, this parameter is difficult to determine analytically. Therefore, in our implementation, we determine the correct initial patch size by the quality of the final segmentation. If the initial neighborhood size is too small, the eigenvectors in the slippage signature of each point will not match the point's neighbors, and the algorithm will not be able to aggregate large patches. The final output will be over segmented: there will be a large number of components, with only a few points in each. Therefore, if the final segmentation has too many components, we increase the value of m and try again.

4.3. Post-processing steps

After the completion of the algorithm, it is likely that a number of regions remains that are not part of any slippable component. One reason of this is that those parts corresponds to areas of the shape that are not slippable. However, often there are regions in the input that are actually part of a slippable shape, but due to noise did not get clustered correctly. We therefore allow large slippable regions to absorb their small neighbors, as long as the overall region remains slippable.

We also apply some simplification to the border between regions. The exact border between two components depends on the neighborhoods size, m, that was used in the initial point classification. For example, in Figure 2, a larger neighborhood size would increase the width of the one-slippable regions. While we may need larger size of m for the initial clustering, we often prefer the points to belong to a more slippable component at the end of the segmentation. Therefore, when a point can be classified as belonging to two different patches, as happens near the border between two regions, we assign the point to the more slippable region. In a way, we allow the more slippable regions to eat into the less slippable regions, but only as long as no regions become disconnected.

5. Results

In this section, we show the results obtained by our segmentation algorithm.

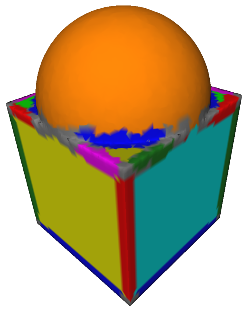


Figure 4: *Segmentation of a simple model into slippable components. Grey areas correspond to stable regions. Notice that even through the spherical part and planar parts were classified as having the same number of small eigenvalues in Figure 2, since their slippable motions are incompatible with each other, they are segmented into separate regions.*

Figure 4 shows the final segmentation of the shape in Figure 2. The shape consists of a cube with a hemisphere attached to one of the faces. The final segmentation consists of three-slippable and one-slippable regions, as well as a number of stable (not slippable) corners. The planar faces and the sphere are segmented into separate components, since even though they have the same number of slippable motions, the motions themselves are different. The segmentation also includes one-slippable components that correspond to the edges between the three-slippable regions. The width of these components depends on the size of the initial neighborhood that is built around each point. Our second cleaning step thins such edge pieces as long as they don't fall apart into disconnected regions. In our application, the input to the algorithm was given as a mesh. Therefore, we used the connectivity of the input to prevent the segmented regions from forming disconnected components during thinning. In the case when input data is given without any underlying connectivity, more sophisticated methods for preserving topology of the regions are required [ELZ02].

Many segmentation algorithms perform edge detection as the first step of segmentation, using the assumption

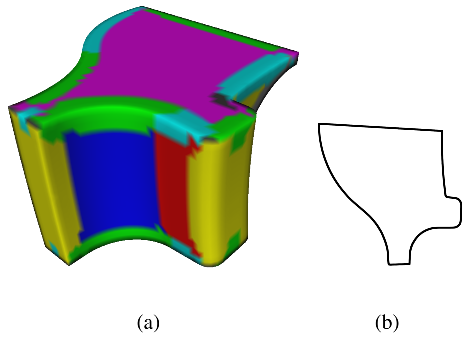


Figure 5: *Segmentation without sharp edges. (a) The back of the fandisk model contains a cylindrical region (blue) and a planar region (red). Even though there is no sharp edge between them, our segmentation algorithm is able to recognize the two regions. (b) Outline of the bottom of the model, showing the shape of the cylindrical and planar region.*

that different regions in the input are separated by sharp edges [BMV01]. Our algorithm does not need the edge detection step, and in fact can find boundaries between slippable regions even if no sharp edge is present in the data. Figure 5 shows a shape containing a cylindrical part that smoothly joins to a plane, which are correctly identified by our algorithm.

Figure 6 shows the segmentation of a mechanical part consisting mostly of cylinders and surfaces of revolution. The input pointset contains 20,000 vertices. The initial patch classification of a neighborhood of 30 vertices is shown in Figure 6(a). The size of the neighborhood was not large enough to correctly classify all vertices that belong to the cylindrical regions, as indicated by the blue coloring (corresponding to the three-slippable regions) at the lower part of the shape. The misclassified vertices were not aggregated in the first pass of the algorithm, since treated as planes, their similarity score was too low. However, in the second pass, they were correctly treated as two-slippable regions and clustered. The results are shows in Figure 6(b)-(d). Notice that the segmentation captures the edge between the planar bottom part and the cylindrical side part as a separate one-slippable component. The fact that the algorithm classifies sharp edges as separate components can be advantageous in reverse engineering. Since sharp edges are generally hard to capture accurately in laser scanning, this component is useful as an indicator of the area that is likely to contain a sharp edge. In the construction of a CAD model from this input pointset, this component can be replaced by a surface blend [BMV01].

Finally, Figure 7 shows segmentation of another mechanical part, consisting of a larger number of slippable components. The model consists of 40,000 points, and the input neighborhood size was set to 30 vertices. After the initial segmentation, the post-processing pass thinned the one-slippable components to the maximum width of 5 vertices.

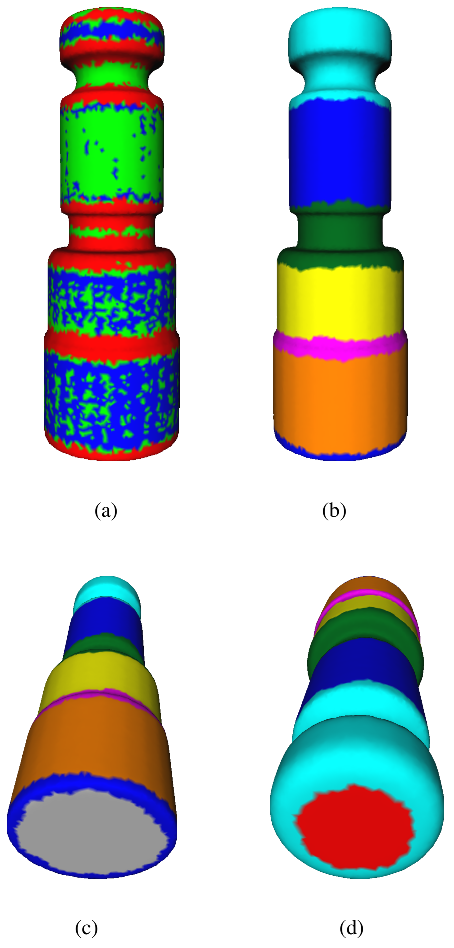
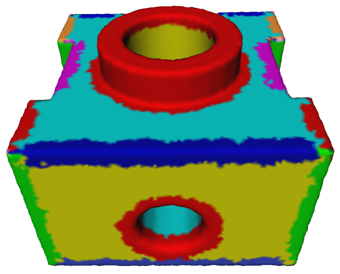


Figure 6: *Segmentation of a mechanical part consisting mostly of cylinders and surfaces of revolution. (a) Initial classification of neighborhoods into one-slippable (red), two-slippable (green), and three-slippable (blue). Notice that some cylindrical neighborhoods are incorrectly classified as planar. (b) Segmentation obtained by our algorithm. (c) View of the bottom of the part. Notice that the join between the planar bottom and the cylindrical side is captured as a separate rotationally symmetrical component. (d) View of the top of the part.*

6. Conclusions

We presented an algorithm for segmenting an input pointset into regions that can be well approximated by kinematic surfaces. A sample application for such algorithm is the seg-


(a) Top view

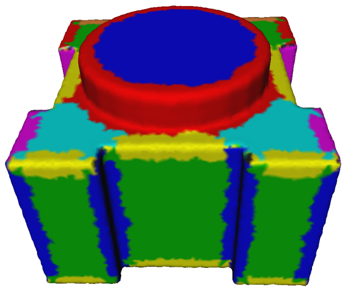

(b) Bottom view

Figure 7: *Segmentation of a part consisting of planar and cylindrical components. Edges between the planar components and between the planar and cylindrical components are captured as separate regions.*

mentation of mechanical parts in the area of reverse engineering. We see another use of our algorithm in segmentation of scans of architectural structures, which are often composed of planar surfaces (walls) and surfaces of linear extrusion (corners, door frames).

Several directions are possible for future work. We need an analytic way to choose the neighborhood size parameter used for the initial classification of points. As mentioned in Section 4.2, neighborhood size that is too small leads to slippage signatures that are essentially random, and results in a poor segmentation. We expect that we can characterize the optimum neighborhood size in terms of the local curvature and an estimate of noise in the data.

In this paper, we focused on developing the concept of slippable motions and the idea of using slippage signature of a neighborhood around a point as a local surface descriptor. Our segmentation algorithm is a simple greedy clustering method. We expect that we can improve the quality of the segmentation by using more sophisticated clustering techniques such as [KT03] with our surface descriptor.

Acknowledgments The authors wish to acknowledge support from NSF grant CARGO-0138456 and a Max-Planck-Institut Fellowship. We thank Daniel Russel and Marc Levoy for useful discussions.

References

[Ale02] ALEXA M.: Linear combination of transformations. In *Proc. SIGGRAPH* (2002).

[BM03] BOIER-MARTIN I.: Domain decomposition for multiresolution analysis. In *Proc. SGP* (2003).

[BMV01] BENKO P., MARTIN R., VARADY T.: Algorithms for reverse engineering boundary representation models. *Computer Aided Design 33*, 11 (2001).

[CM91] CHEN Y., MEDIONI G.: Object modeling by registration of multiple range images. In *Proc. IEEE Conf. on Robotics and Automation* (1991).

[ELZ02] EDELSBRUNNER H., LETSCHER D., ZOMORODIAN A.: Topological persistence and simplification. *Discrete Comput. Geom. 28* (2002), 511–533.

[GIRL03] GELFAND N., IKEMOTO L., RUSINKIEWICZ S., LEVOY M.: Geometrically stable sampling for the ICP algorithm. In *Proc. Intl. Conf. on 3D Imaging and Modeling* (2003).

[GWH01] GARLAND M., WILLMOTT A., HECKBERT P.: Hierarchical face clustering on polygonal surfaces. In *Proc. ACM Symp. Inter. 3D Graphics* (2001).

[KL96] KRISHNAMURTHY V., LEVOY M.: Fitting smooth surfaces to dense polygon meshes. In *Proc. SIGGRAPH* (1996).

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. In *Proc. SIGGRAPH* (2003).

[LPRM02] LEVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. In *Proc. SIGGRAPH* (2002).

[MNG04] MITRA N., NGUYEN A., GUIBAS L.: Estimating surface normals in noisy point cloud data. In *International Journal of Computational Geometry and Applications* (2004). To Appear.

[PHOW04] POTTMANN H., HOFER M., ODEHNAL B., WALLNER J.: Line geometry for 3D shape understanding and reconstruction. In *ECCV* (2004).

[PR97] PARK F., RAVANI B.: Smooth invariant interpolation of rotations. *ACM Transactions on Graphics 16*, 3 (1997).

[PR98] POTTMANN H., RANDRUP T.: Rotational and helical surface approximation for reverse engineering. *Computing 60* (1998).

[PW01] POTTMANN H., WALLNER J.: *Computational Line Geometry*. Springer Verlag, 2001.

[RL01] RUSINKIEWICZ S., LEVOY M.: Efficient variants of the ICP algorithm. In *Proc. Intl. Conf. on 3D Imaging and Modeling* (2001).

[SB95] SAPIDIS N., BESL P.: Direct construction of polynomial surfaces from dense range images through region growing. *ACM Transactions on Graphics 14*, 2 (1995).

[SM00] SHI J., MALIK J.: Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (2000).

[Sri03] SRINIVASAN V.: *Theory of Dimensioning*. Marcel Dekker, 2003.

[SSGH02] SANDER P., SNYDER J., GORTLER S. J., HOPPE H.: Texture mapping progressive meshes. In *Proc. SIGGRAPH* (2002).

[VMC97] VARADY T., MARTIN R., COX J.: Reverse engineering of geometric models - an introduction. *Computer-Aided Design 29*, 4 (1997).