Eurographics Symposium on Geometry Processing (2006)
Konrad Polthier, Alla Sheffer (Editors)

Partial Matching of 3D Shapes with Priority-Driven Search

T. Funkhouser and P. Shilane

Princeton University, Princeton, NJ

Abstract

Priority-driven search is an algorithm for retrieving similar shapes from a large database of 3D objects. Given a query object and a database of target objects, all represented by sets of local 3D shape features, the algorithm produces a ranked list of the c best target objects sorted by how well any subset of k features on the query match features on the target object. To achieve this goal, the system maintains a priority queue of potential sets of feature correspondences (partial matches) sorted by a cost function accounting for both feature dissimilarity and the geometric deformation. Only partial matches that can possibly lead to the best full match are popped off the queue, and thus the system is able to find a provably optimal match while investigating only a small subset of potential matches. New methods based on feature distinction, feature correspondences at multiple scales, and feature difference ranking further improve search time and retrieval performance. In experiments with the Princeton Shape Benchmark, the algorithm provides significantly better classification rates than previously tested shape matching methods while returning the best matches in a few seconds per query.

1. Introduction

Large databases of 3D models are becoming available in a number of disciplines, including computer graphics, mechanical CAD, molecular biology, and medicine. As these databases grow, shape-based similarity search is emerging as a valuable tool for analysis and discovery.

The goal of our work is to develop effective methods to retrieve from a database a ranked list of 3D models most similar in shape to a 3D model provided as a query. This problem is difficult because objects of the same type may not have exactly the same sets of parts (e.g., some chairs have arms and others don't), and some parts that distinguish object types may be relatively small (e.g., the ears of a bunny). Shape representations that account only for global shape properties do not perform well at recognizing shapes in these situations.

A common method for addressing this problem is to represent every object by a set of local shape features centered on points sampled from the object's surface and then to compute a similarity metric for every pair of objects based on a cost function measuring the quality of matches in the optimally aligned set of feature correspondences (e.g., [BMP01, CJ96, JH99]). This approach is attractive because it is robust to shape variations within a class – as long as a few key shape features match, then the objects will match. The challenge is that the number of possible feature correspondence sets grows exponentially with the set size – naively checking all possible sets of k feature correspondences among n features on two objects takes $O(n^k)$ operations. In practice, searching the space of potential feature correspondences for a single pair of surfaces can take several seconds or minutes, and using these methods to find the best matches in a large database is impractical.

In this paper, we introduce a priority-driven algorithm for searching all objects in a database at once. The algorithm is given a query object and a database of target objects, all represented by sets of local shape features, and its goal is to produce a ranked list of the best target objects sorted by how well any subset of k features on the query match features on the target object. To achieve this goal, the system maintains a priority queue of potential sets of feature correspondences (partial matches) sorted by a cost function accounting for both feature dissimilarity and geometric deformation. Initially, all pairwise correspondences between the features of the query and features of target objects are loaded onto the priority queue. Then, at every step, the best partial match m is popped off the priority queue, new partial matches are created by extending m to include compatible feature correspondences, and those new partial matches are added to

the priority queue. This process is iterated until the desired number of full matches with k feature correspondences have been popped off the priority queue.

The advantage of this approach is that the algorithm provably finds the optimal set of matches over the entire database while investigating only a small subset of the potential matches. Like any priority-driven backtracking search (e.g., Dijkstra's shortest path algorithm), the algorithm considers only the partial matches that can possibly lead to the lowest cost match (Figure 1). Although some poor partial matches are generated, they never rise to the top of the priority queue, and thus they incur little computational overhead. By using a single priority queue to store partial matches for all objects in the database at once, we achieve great speedups when retrieving only the top matches – if a small set of target objects match the query well, their feature correspondences will be discovered quickly and the details of other potential matches will be left unexplored. This approach largely avoids the combinatorial explosion of searching for multi-feature matches in dissimilar objects.

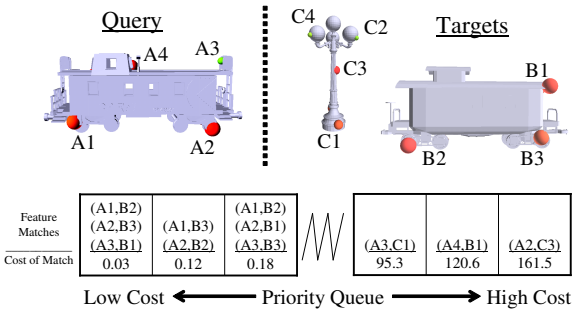


Figure 1: *Priority driven search: a priority queue (bottom) stores potential matches of features (labeled dots) on a query to features of all target objects at once. Matches are extended only when they reach the top of the priority queue (the left-most entry), and thus high cost feature correspondences sit deep in the priority queue and incur little computational expense.*

This paper makes several research contributions. In addition to the idea of priority-driven search, we explore ways of improving computational efficiency and retrieval performance of multi-feature matching algorithms: 1) we use ranks rather than L^2 differences to measure feature similarity; 2) we use a measure of class distinction to select features; and, 3) we match features at multiple scales. Finally, we provide a working shape-based retrieval system and analyze its performance over a wide range of options and parameter settings. We find that our system provides significantly better retrieval performance than previous shape matching approaches on the Princeton Shape Benchmark [SMKF04] while using increased, but reasonable, processing and storage costs.

The organization of the paper is as follows. The next section contains a summary of related work on matching of 3D surfaces. Section 3 contains an overview of the priority-driven search algorithm followed by a detailed description for every algorithmic step. Section 4 compares the performance of the priority-driven search approach to other state-of-the-art shape matching methods and investigates how modifying several aspects of the algorithm impacts its performance. Finally, Section 5 provides a brief discussion of limitations and topics for future work.

2. Background and Related Work

There has been a large amount of research on algorithms for shape-based retrieval of 3D surface models. In this section, we focus on the previous work most closely related to ours and refer the reader to survey articles for broad overviews of prior work in related areas [BKS*05, IJL*05, TV04].

The most common approach to shape-based retrieval of 3D objects is to represent every object by a single global *shape descriptor* representing its overall shape. Shape Histograms [AKKS99], the Light Field Descriptor [COTS03], and the Depth Buffer Descriptor [HKSV02] are a few examples. These descriptors can be searched efficiently, and thus they are suitable for queries into large databases of 3D shapes. However, retrieval precision is generally poor when objects within the same class have different overall shapes – e.g., due to articulated motions, missing parts, or extra parts.

Recently, several researchers have investigated approaches to partial shape matching based on feature correspondences (e.g., [BMP01, CJ96, GCO06, JH99, NDK05]). The general strategy is to compute multiple local shape descriptors (shape features) for every object, each representing the shape for a region centered at a point on the surface of the object. Then, the similarity of any pair of objects is determined by a cost function determined by the optimal set of feature correspondences at the optimal relative transformation, where the optimal match minimizes the differences between corresponding shape features and the geometric distortion implied by the feature correspondences. This approach has been used for recognizing objects in 2D images [BMP01, BBM05], recognizing range scans [JH99], registering medical images [AFP00], aligning point sets [CR00], aligning 3D range scans [GMGP05, LG05], and matching 3D surfaces [NDK05, SMS*04].

The challenge is to find an optimal set of feature correspondences efficiently. One approach is to consider an association graph containing a node for every possible feature correspondence and an edge for every compatible pair of correspondences [BB76]. If each node is weighted by the dissimilarity of its associated features and each edge is weighted by the cost of the geometric deformation implied by its associated pair of correspondences, then finding the optimal set of k feature correspondences reduces to finding a minimum weight k-clique in the association graph. Researchers have approached this problem with algorithms

based on branch-and-bound [GMGP05], integer quadratic programming [BBM05], etc. However, previous work in this area has been aimed at pairwise alignment of objects, and current solution methods are generally too slow for search of large databases.

Another approach is based on the RANSAC algorithm [FB81, SMS*04]. Sets of k feature correspondences are generated, where k is large enough to determine an aligning transformation, and the remaining features are used to score how well the objects match after the implied alignment. For example, [JH99] finds small sets of compatible feature correspondences, computes the alignment providing a least-squared best fit of corresponding features, and then "verifies" the alignment with an iterative closest point algorithm [BM92]. [SMS*04] proposed a "Batch RANSAC" version of this algorithm that considers matches to all target objects in a database all at once, generating candidate matches preferentially for the target objects with features compatible with ones in the query. However, their evaluation focused on recognition of vehicles from a small set of range scans, and studies have not been done to show how well it works for large databases of surface models.

Several researchers have considered methods for accelerating database searches using discrete approximations. For example, geometric hashing [LW88] uses a grid-based hash table to store every feature of every target object in n choose k hash cells. For every query, k features are used to determine a mapping into the hash, and then other features vote for object transformations wherever there are hash collisions. This approach is quite popular in computer vision, molecular biology, and partial surface matching (e.g., [GCO06]). However, it requires a lot of memory to store hash tables and produces approximate matches, since it discretizes both the set of possible transformations (it only considers transformations induced by combinations of features) and Euclidean space (features match only if they fall in the same grid cell).

Alternatively, "bag of words" approaches can be used to discretize feature space. For example, [MBM01] clusters features into "shapemes," builds a histogram of shapemes for every object, and then approximates the similarity of two objects by the similarity of their histograms, and [GD05] extends this approach to consider pyramids of clusters. However, these methods make little or no use of the geometric arrangements of features, and thus they do not provide as distinguishing matches as possible.

Our approach is to use priority-driven search to find the objects in a database that have sets of local feature correspondences minimizing a continuous cost function. The key idea is to use a priority queue to focus a backtracking search on sets of feature correspondences with lowest matching cost among all objects in the database all at once. This approach provides a significant efficiency improvement over more expensive algorithms that compute pairwise matches between the query and all objects in the database independently (e.g., [BBM05]) – i.e., it avoids computing the optimal set of correspondences for the target objects that do not appear at the top of the retrieval list. It can also provide an accuracy improvement over discrete or greedy approximate algorithms, since it guarantees optimal matches with respect to a continuous cost function.

3. System Execution

Execution of our system proceeds in two phases: a preprocessing phase and a query phase.

During the preprocessing phase, we build a multi-feature representation of every object in the database. First, we generate for each object a set of spherical regions covering its surface at different scales. Second, for every region, we compute a descriptor of the shape within that region. Third, we compute differences between all pairs of descriptors at the same scale and associate with every descriptor a mapping from rank to difference. Finally, we select a subset of features to represent each object based on how distinctive they are of their object class. The result of this preprocessing is a set of "shape features" (or "features," for short) for every object, each with an associated position (p), normal ($\vec{n}$), radius (r), and shape descriptor (a feature vector of numbers representing a local region of shape), and a description of how discriminating its shape descriptor is with respect to others in the database.

For every query, our matching procedure proceeds as shown in Figure 2. The inputs are: 1) a query object, *query*, 2) a database of target objects, *db*, each represented by a set of shape features, 3) a cost function, *cost*, measuring the quality of a proposed set of feature correspondences, 4) a constant, k, indicating the number of feature correspondences that should be found for a complete match, and 5) a constant, c, indicating the number of objects for which to retrieve optimal matches. The output is a list of the best matching target objects, M, along with a description of the feature correspondences and cost for each one.

Initially, a priority queue, Q, is created to store partial matches, and an array, M, is created to store the best match to every target object. Then, all pairwise correspondences between the features of the query and features of the target objects are created, stored in lists associated with the target objects, and loaded onto the priority queue. The priority queue then holds all possible matches of size 1. Then, until c complete matches have been found, the best partial match, m, is popped off the priority queue. If it is a complete match (i.e., the number of feature correspondences satisfies k), then the search of that target object is complete, and the priority queue is cleared of partial matches to that object. Otherwise, for every feature correspondence between the query and the target of m, the match is extended by one feature correspondence to form a new match, m'. The best match for every target object is retained in an array, M, when it is added to

```
PriorityDrivenSearch(Object query, Database db,
    Function cost, int k, int c)
  # Create correspondences
  foreach Object target in db
    foreach Feature q in query
      foreach Feature t in target
        p = CreatePairwiseCorrespondence(q, t, cost)
        if (IsPlausible(p))
          AddToPriorityQueue(Q, p)
          AddToList(C[target], p)
          if (cost(p) < cost(M[target]))
            M[target] = p

  # Expand matches until find complete ones
  complete_match_count = 0
  while (complete_match_count < c)
    # Pop match off priority queue
    m = PopBestMatch(Q)
    target = GetTargetObject(m)

    # Check for complete match
    if (IsMatchComplete(m, k))
      RemoveMatchesFromPriorityQueue(Q, target)
      complete_match_count++
      continue;

    # Extend match
    foreach PairwiseCorrespondence p in C[target]
      m' = ExtendMatch(m, p, cost)
      if (IsPlausible(m'))
        AddToPriorityQueue(Q, m')
        if (cost(m') < cost(M[target]))
          M[target] = m'

  # Return result
  return M
```

Figure 2: *Pseudo-code for priority-driven search.*

the priority queue. This process is iterated until at least c full matches with k feature correspondences have been popped off the priority queue for c distinct target objects, and the array of the best matches to every target object, M, is returned as the result.

The computational savings of this procedure come from two sources. First, matches are considered from best to worst, and thus, poor pairwise correspondences are never considered for extension and add little to the execution time of the algorithm. Second, after complete matches for at least c target objects have been added to the priority queue, it is possible to determine an upper-bound on the cost of matches that can plausibly lead to one of the best matches. If the score computed for an extended match, m', is higher than that upper bound, then there is no reason to add it to the queue, and it can be ignored. Similarly, if a match, m, is popped off the queue, then it is provably the best remaining match – i.e., no future match can be considered with a lower cost. Thus, the algorithm can terminate early (immediately after c best matches have been popped off the priority queue) while still guaranteeing an optimal solution.

Of course, there are many design decisions that impact the efficacy of this search procedure, including how regions are constructed, how shape descriptors are computed, what cost function is used, how implausible matches are culled, and so on. The following subsections describe our design decisions in detail, and Section 4 provides the results of experiments aimed at evaluating the impact of each one on search speed and retrieval performance.

3.1. Constructing Regions

The first step of the process is to define a set of local regions covering the surface of every object. In theory, the regions could be volumetric or surface patches; they could be disjoint or overlap; and, they could be defined at any scale.

In our system, we construct overlapping regions defined by spherical volumes centered on points sampled from the surface of an object [KPNK03, NDK05]. We have experimented with two different point sampling methods, one that selects points randomly with uniform distribution with respect to surface area, and another that selects points at vertices of the mesh with probability equal to the surface area of the vertices' adjacent faces. However, they do not give significantly different performance, and so we consider only random sampling with respect to surface area for the remainder of this paper. Of course, other sampling methods that sample according to curvature, saliency, or other surface properties would be possible as well.

We have experimented with regions at four different scales. The smallest scale has radius 0.25 times the radius of the entire object and the other scales are 0.5, 1.0, and 2.0 times, respectively. These scales are chosen because the smallest scale is approximately the size of most "distinguishing features of an object" and the largest scale is just big enough to cover the entire object for spheres centered at the most extreme positions on the surface (Figure 3).

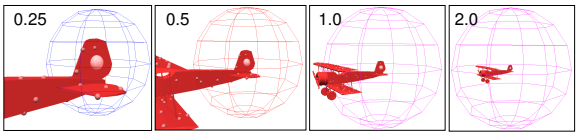


Figure 3: *Shape regions at four different scales.*

3.2. Computing Shape Descriptors

The second step of the process is to generate and store a representation of the shape for each spherical region (a shape descriptor). There will be many such regions for every surface, so the shape descriptors must be quick to compute, concise to store, and fast to compare, in addition to being as discriminating as possible. There are many shape descriptors that meet some or all of these criteria (see surveys in [BKS*05, IJL*05, TV04]). Examples include shape contexts [BMP01, MBM01, FHK*04],

spin images [JH99, SMS*04], harmonic shape descriptors [FHK*04, NDK05], curvature profiles [CJ96, GCO06], and volume integrals [GMGP05].

In our system, we have experimented with three different shape descriptors based on spherical harmonics. All three decompose a sphere into concentric shells of different radii and then describe the distributions of shape within those shells using properties of spherical harmonics. The first ("SD") simply stores the amplitude of all shape within each shell (the zero-th order component of spherical harmonics) – it is a one-dimensional descriptor equivalent to the "Shells" shape histogram of [AKKS99]. The second ("HSD") stores the amplitude of spherical harmonic coefficients within each frequency – it is equivalent to the Harmonic Shape Descriptor of [FMK*03, KFR03]. The last ("FSD") descriptor stores the amplitude of every spherical harmonic coefficient separately – it is similar to the Harmonic Shape Contexts of [FHK*04]. In all of our experiments, we utilize 32 spherical shells and 16 harmonic frequencies for each descriptor.

We chose these shape representations for several reasons. First, they are well-known descriptors that have been shown to provide good performance in previous studies [FHK*04, SMKF04]. Second, they are reasonably robust, concise, and fast to search. Finally, they provide a nested continuum with which to investigate the trade-offs between verbosity and discrimination – SD is very concise (32 values), but not that discriminating; HSD is more verbose (512 values) and more discriminating; and FSD is the most verbose (4352 values) and the most discriminating. The three descriptors are related in that each of the more concise descriptors is simply a subset or aggregation of terms in the more verbose ones (e.g., the SD descriptor stores the amplitude of only the zero-th order spherical harmonic frequencies). Thus, the L^2 difference of each descriptor provides a lower bound on the L^2 difference between the more verbose ones, which enables progressive refinement of descriptor differences, as proposed in Section 5.

Our method for computing the descriptors for all regions of a single surface starts by computing a 3D grid containing the Euclidian Distance Transform of the surface. The grid resolution is chosen to match the finest sampling rate required by the HSD for regions at the smallest scale; the triangles of the surface are rasterized into the grid; and the squared distance transform is computed and stored. Then, for every spherical region centered on a point sampled from the surface, a spherical grid is constructed by aligning a sphere with the normal to the surface; a Gaussian function of the distance transform (GEDT) [KFR03] is sampled at regular intervals of radius and polar angles; the Spharmonickit software is used to compute the spherical harmonic decomposition for each radius; the amplitudes of the harmonic coefficients (or frequencies, depending on the type of shape descriptor) are computed; the shape descriptors are compressed using principal component analysis (PCA); and, the dimensions associated with the top C eigenvalues (C $\sim$ 10%) are stored as a shape descriptor.

For each 3D object, computing the three types of shape descriptors centered at 128 points for 4 scales (0.25, 0.5, 1.0, and 2.0) takes approximately four minutes overall and generates around 1MB of data per object. One minute is spent rasterizing the triangles and computing the squared distance transform at resolution sufficient for the smallest scale descriptors, almost two minutes are spent computing the spherical grids, and a few seconds are spent decomposing the grids into spherical harmonics for each object. Compression amortizes to approximately one minute per object for SFDs and approximately 1 second per object for SHDs.

3.3. Selecting Distinctive Features

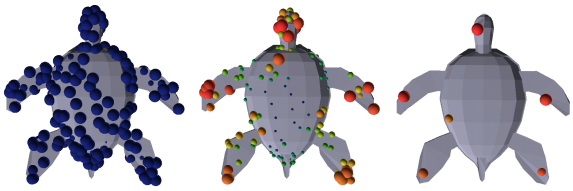
The third step of our process is to characterize the differences between shape descriptors and to select a subset of the shape features to be used for matching for each target object. Our goal is to augment the features with information about how discriminating they are and to select a subset of the most distinctive features in order to improve processing speed and retrieval precision.

Selecting a subset of local shape descriptors is a well known technique for speeding up retrieval, and several researchers have proposed different methods for this task, The simplest technique is to select features randomly [JH99, FHK*04, MBM01]. Other methods have considered selecting features based on surface curvature [YF02], saliency [GCO06], likelihood within the same shape [GMGP05, JH99], persistence across scales [GMGP05], number of matches to another shape [SMS*04], likelihood within the database [SF06], and distinction of its object's class [SF06].

In our system, we follow the ideas of [SF06]. For every feature, we compute the L^2 difference of its shape descriptor to the best match of every other object in the database, sort the differences from best to worst, and save them in a *rank-to-difference mapping* (RTD). To save space, we store an approximation to the RTD containing log(N) values by sampling distances at exponentially larger ranks. We then use the RTD to estimate the distinction of every shape feature. Distinctive features are ones that are both similar to features in few other objects (ones in the same class) and different from the rest (objects in other classes). When given a classification for the target objects, we quantify this notion by using a measure of retrieval performance as our model for feature distinction [SF06].

Once the distinction of every feature has been computed, we employ a greedy algorithm to select a small set of features to represent every target object during the query phase (Figure 4). The selection algorithm iteratively chooses the feature with highest DCG whose position is not closer than a Euclidean distance threshold, *minlength*, to the position of

any previously selected feature. This process avoids selecting features nearby each other on the mesh and provides an easy way to vary the subset size by adjusting the distance threshold.



(a) All Features (b) Feature Distinction (c) Selected Features

Figure 4: *Feature selection: (a) positions sampled randomly on surface, (b) computed DCG values used to represent feature distinction (red is highest, blue is lowest), and (c) features selected to represent object during matching.*

The net result of this process is a small set of features for every target object, each with an associated position (p), normal ($\vec{n}$), radius (r), a set of shape descriptors (SD, HSD, and FSD), a rank-to-difference-mapping (RTD), and a retrieval performance score (DCG). In our implementation, computing the RTD and the distinction for each feature takes a little less than 1 second, and selecting the most distinctive features takes less than a second for each object. The storage required for the resulting data required at query time is approximately 100KB per object.

3.4. Creating Pairwise Feature Correspondences

When given a query object to match to a database of target objects, the first step is to compute the cost of pairwise correspondences between features of the query to features of the target. The key to this step is to develop a cost function that provides low values only when two features are compatible and gradually penalizes pairs that are less similar. The simplest and most common approach is to use the L^2 difference between their associated shape descriptors. This approach forms the basis for our implementation, but we augment it in three ways.

First, given features F_1 and F_2, we compute the L^2 difference, D, between their shape descriptors. Then, we use the rank-to-difference mappings (RTD) of each feature to convert D into a rank (i.e., where that distance falls in the ranked list associated with each feature). The new difference measure (C_{rank}) is the sum of the ranks computed for F_1 with respect to the RTD of F_2, and vice versa:

$$C_{rank} = Rank(RTD_1, D) + Rank(RTD_2, D)$$

This feature rank cost (which we believe is novel) avoids the problem that very common features (e.g., flat planar regions) can provide indistinguishing matches (false positives) when L^2 differences are small. Our approach considers not the absolute difference between two features, but rather their difference relative to the best matching features of other objects in the database. Thus, a pair of features will only be considered similar if both rank highly in the retrieval list of the other.

Second, we augment the cost function with geometric terms. For part-in-whole object matching, we can take advantage of the fact that features are more likely to be in correspondence if they appear at the same relative position and orientation with respect to the rest of their objects. Thus, for each feature, we compute the distance between its position and the center of mass of its object (R), scaled by the average of R for all features in the object ($RAVG$), and we add a distance term C_{radius} to the cost function accounting for the difference between these distances:

$$C_{radius} = |\frac{R_1}{RAVG_1} - \frac{R_2}{RAVG_2}|$$

We also compute a normalized vector $\vec{r}$ from the object's center of mass to the position of each feature and store the dot product of that vector with the surface normal ($\vec{n}$) associated with the feature. The absolute value of the dot product is taken to account for the possibility of backfacing surface normals. Then, the difference between dot products for any pair of features is used to form a normal consistency term to the cost function:

$$C_{normal} = ||\vec{r}_1 \cdot \vec{n}_1| - |\vec{r}_2 \cdot \vec{n}_2||$$

Overall, the cost of a feature correspondence is a simple function of these three terms:

$$C_{correspondence} = \alpha_{rank} C_{rank}^{\gamma_{rank}} + \alpha_{radius} C_{radius}^{\gamma_{radius}} + \alpha_{normal} C_{normal}^{\gamma_{normal}}$$

where the α coefficients and γ exponents are used to normalize and weight the terms with respect to each other.

Of course, computing all potential pairwise feature correspondences between a query object and a database of targets is very costly. If the query has M_Q features and each of N targets has M_T selected features, then the total number of potential feature correspondences is $N \times M_Q \times M_T$. To accelerate this process, we utilize conservative thresholds on each of the three terms (*maxrank*, *maxradius*, *maxnormal*) to throw away obviously poor feature correspondences. The terms are computed and the thresholds are checked progressively in order of how expensive they are to compute (e.g., C_{rank} is last), and thus there is great opportunity for trivial rejection of poor matches with little computation. Indexing and progressive refinement could further reduce the compute time as described in Section 5.

3.5. Searching for the Optimal Multi-Feature Match

The second step of the query process is to search for the best multi-feature matches between the query object and the target objects. This is the main step of priority-driven search.

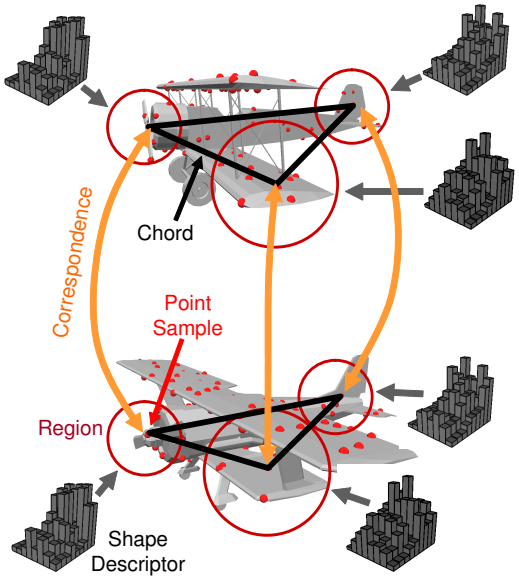


Figure 5: *A 3-feature match for two airplanes. Red points represent feature positions on the surface. For three features, red circles represent regions, gray histograms represent shape descriptors, orange lines represent feature correspondences, and black lines represent lengths between features of same object. Consistency of all shape descriptors, lengths, and angles is required for a good match.*

A priority queue is used to store incomplete sets of feature matches during a backtracking search. Initially, all pairwise correspondences (computed as described in the previous subsection) are loaded onto the priority queue. Then, the best partial match, m, is repeatedly popped off the priority queue, and then extended matches are created for every compatible feature correspondence and loaded onto the priority queue. This process is iterated until at least c full matches with k feature correspondences have been popped off the priority queue for distinct target objects.

As a partial match is extended to include one more feature correspondence, two extra terms are added to the cost function to account for geometric deformations implied by multiple pairwise feature correspondences (Figure 5). First, a chord length term C_{length} is added to penalize matches with inconsistent inter-feature lengths. Specifically, for every pair of feature correspondences in m, we compute the length of the chord between feature positions in the same object (L), scaled by the average of L over all features in the object ($LAVG$). Then, we compute the difference between these distances and normalize by the greater of the two to produce the length term of the cost function:

$$C_{length} = \frac{|\frac{L_1}{LAVG_1} - \frac{L_2}{LAVG_2}|}{max(\frac{L_1}{LAVG_1}, \frac{L_2}{LAVG_2})}$$

Second, a surface orientation term is added to penalize matches with pairs of feature correspondences whose surface normals are inconsistent. This term penalizes both mismatches in the relative orientations of the two pairs of normals with respect to one other and mismatches in the orientations of the normals with respect to the chord between the features. If $\vec{v_1}$ is the normalized vector between features 1a and 1b with normals $\vec{n_{1a}}$ and $\vec{n_{1b}}$ in object 1, and similar variables describe the relative orientations of features in object 2, then the orientation term of the cost function can be computed as follows:

$$\begin{aligned} C_{orient} = \quad & ||\vec{n_{1a}} \cdot \vec{n_{1b}}| - |\vec{n_{2a}} \cdot \vec{n_{2b}}||+ \\ & ||\vec{v_1} \cdot \vec{n_{1a}}| - |\vec{v_2} \cdot \vec{n_{2a}}||+ \\ & ||\vec{v_1} \cdot \vec{n_{1b}}| - |\vec{v_2} \cdot \vec{n_{2b}}|| \end{aligned}$$

These terms are also weighted and raised to exponents to provide normalization when added to the overall scoring function computed for a match with k feature correspondences:

$$C_{chord} = \alpha_{length} C_{length}^{\gamma_{length}} + \alpha_{orient} C_{orient}^{\gamma_{orient}}$$

As in the previous section, we utilize conservative thresholds on C_{length} and C_{orient} ($maxlength$ and $maxorientation$) to throw away obviously poor feature correspondences. We also utilize a threshold on the minimum distance between features within the same object ($minlength$) in order to avoid matches comprised of features in close proximity to one another.

The overall cost of a match is the sum of the terms representing differences in the k feature correspondences and the geometric differences between the k(k-1)/2 chords spanning pairs of features:

$$C_{match} = \sum_{i<k} C_{correspondence}(i) + \sum_{i,j<k, i<j} C_{chord}(i,j)$$

4. Results

In this section, we present results of experiments with priority-driven search. We investigate the performance of the method in relation to the state of the art in shape-based retrieval and investigate the impact of several design choices on the speed and quality of retrieval results.

All experiments were based on the 3D data provided in the Princeton Shape Benchmark [SMKF04]. It contains 907 polygonal models partitioned into 92 classes (sedans, race cars, commercial jets, fighter jets, dining room chairs, etc.). This database was chosen because it has been used in several previous 3D shape retrieval studies (e.g., [BKS*05, SMKF04]) and thus forms a basis for comparison with competing methods.

In a representative preprocessing phase, we generated features at 128 surface points with 4 different scales for every object. For every feature, we computed its shape descriptors,

RTDs, and DCGs, and then we selected the most distinctive set of descriptors using the methods described in Sections 3.1-3.3. The total preprocessing time for all 907 objects was 70 hours and the total size of all data generated was 1GB, of which 64MB represents the selected features that had to be stored in memory for target objects during the query phase.

During the query phase, we performed a series of "leave-one-out" classification tests. In each test, every object of the database was used as a *query object* to search databases containing the remaining $N-1$ target objects. Standard information retrieval metrics, such as precision, recall, nearest neighbor classification rate (1-NN), first-tier percentage (1-tier), second-tier percentage (2-tier), and discounted cumulative gain (DCG), were computed to measure how many objects in the query's class appear near the top of its ranked retrieval list, and those metrics were averaged for all queries.

Unless otherwise stated, experiments were run on a x86_64 processor with 12GB of memory running Linux. Parameters for the "base configuration" of the system were set as follows: $c = 1$, $k = 3$, number of features per object = 128, number of feature scales = 4 (0.25, 0.5, 1.0, and 2.0), shape descriptor type = HSD, compression ratio = 10X, $maxradius = maxnormal = maxlength = maxorientation = 0.25$, $minlength = 0.3 \cdot RAVG$, $\alpha_{rank} = 0.01$, $\alpha_{radius} = \alpha_{normal} = \alpha_{length} = \alpha_{orient} = 1$, and $\gamma_{rank} = 4$, $\gamma_{radius} = \gamma_{normal} = \gamma_{length} = \gamma_{orient} = 2$. These parameters were determined empirically and used for all experiments without adjustment, except in Section 4.1 where the FSD shape descriptor was used, and in Section 4.3 where the impact of specific parameter settings was studied.

4.1. Comparison to Previous Methods

The goal of the first experiment was to evaluate the retrieval performance of the proposed priority-driven search (PDS) approach with respect to previous state-of-the-art shape-based retrieval methods:

- **Depth Buffer Descriptor (DSR740B)**: this shape descriptor achieved the highest retrieval performance in the study of [BKS*06]. It describes an object by six depth buffer images captured from orthogonal parallel projections [HKSV02]. Images are stored as Fourier coefficients of the lowest frequencies, and differences between Fourier coefficients provide a measure of object dissimilarity. We use Dejan Vranic's implementation of this method [Vra06] without modification and ran it on a 2GHz Pentium4 running WindowsXP.
- **Light Field Descriptor (LFD)**: this shape descriptor achieved the highest retrieval performance in the study of [SMKF04]. It represents an object as a collection of images rendered from uniformly sampled positions on a view sphere [COTS03]. The dissimilarity of two objects is defined as the minimum L_1-difference between aligned images of the light field, taken over all rotations and all pairings of vertices on two dodecahedra. We use the original implementation provided by Chen et al. without modification and ran it on a 2GHz Pentium4 running WindowsXP.
- **Global Harmonic Shape Descriptor (GHSD)**: this is the shape descriptor currently used in the Princeton 3D Search Engine [FMK*03]. It describes an object by a single HSD feature positioned at the center of mass with radius $RAVG$. We include it in this study to provide an apples-to-apples comparison to a method that matches a single global shape descriptor of the same type used in our study.
- **Random**: This method provides a baseline for retrieval performance. It produces a random retrieval list for every query.

Figure 6 shows a precision-recall plot comparing the average retrieval performance for all queries for each of these shape matching methods. Briefly, precision and recall are metrics used to evaluate ranked retrieval lists. If one considers the top M matches for any query, *recall* measures the fraction of the query's class found, and *precision* measures the fraction of objects found from the query's class – higher curves represent better retrieval performance.

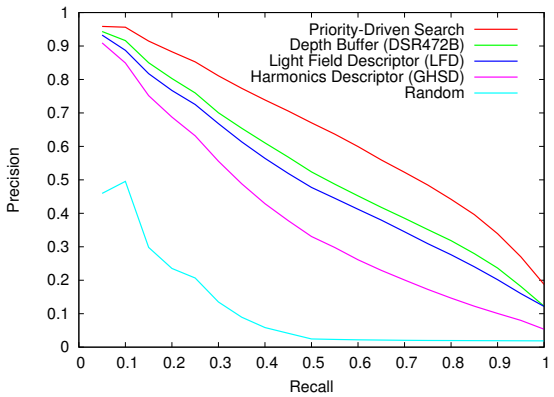


Figure 6: *Precision-recall plot comparing priority-driven search (PDS) to other state-of-the-art shape matching methods using the Princeton Shape Benchmark.*

Timing statistics and standard retrieval performance measures are also shown in Table 1. The leftmost column indicates the shape matching method (PDS is the one described in this paper). The remaining columns list the average time required for one query into the database (in seconds), the average classification rate achieved with a nearest neighbor classifier (1-NN), the average percentages of the query's class that appear in the first-tier (1-Tier) and second-tier (2-Tier), and the average discounted cumulative gain (DCG) computed from the ranked retrieval lists.

From these statistics, we see that the priority-driven search algorithm provides the best retrieval performance of

Method	Time	1-NN	1-Tier	2-Tier	DCG
PDS	2.4	83.4	51.7	63.4	75.9
DSR740B	0.005	66.5	40.3	51.2	66.3
LFD	-	65.0	37.2	47.4	63.6
GHSD	0.003	55.6	30.9	41.1	58.4
Random	0	1.7	1.6	3.4	26.1

Table 1: *Comparison of retrieval statistics between priority-driven search (PDS) and other methods on the Princeton Shape Benchmark (times are in seconds).*

the tested methods on this data set. The improvement in nearest neighbor classification rate over the Depth Buffer Descriptor is 25.4% (83.4% vs. 66.5%) and the improvement over the Light Field Descriptor is 28.3% (83.4% vs. 65.0%). These are remarkable improvements for this data set – typical differences between algorithms found in other studies are usually a couple of percentage points [SMKF04].

However, the PDS algorithm takes considerably more compute time to preprocess the database (4-5 minutes per object), more memory per object (100KB per target object), and more time to find matches (2.4 seconds per query) than the other tested shape descriptors. Almost all of the query processing time is spent establishing the cost of feature correspondences, and less than a tenth of a second is spent finding the optimal multi-feature match with priority driven search. Thus, we believe that simple improvements to the basic algorithm (e.g., compression, indexing, etc.) will significantly improve the processing speed and that query processing times less than a second are possible in this framework (Section 5).

In any case, it seems that priority-driven search is well-suited for batch applications where retrieval accuracy is premium. Often, query results can be computed off-line and cached for later interactive analysis – e.g., for discovery of relationships in mechanical CAD, molecular biology, etc. Even interactive search engines can benefit from off-line preprocessing with high-accuracy matching methods, for example, to preprocess queries that find a shape similar to another in the database (over 90% of the 3D queries to the Princeton 3D Search Engine are of this type [MHKF03]).

4.2. Evaluation of Algorithmic Contributions

The goal of the second experiment is to understand which algorithmic features of the priority-driven search algorithm contribute most to its timing and retrieval performance. To study this question, we started with the "base configuration" and ran the system multiple times on the Princeton Shape Benchmark with different aspects of the system enabled and disabled.

- **Rank (R):** If enabled, the cost of two corresponding shape descriptors (C_{rank}) was the sum of the two ranks in their respective retrieval lists, as described in Section 3.4. Otherwise, it was the direct L^2 distance between shape descriptors (the most common measure of descriptor difference in other systems).
- **Multi-Scale (S):** If enabled, the costs of the best matches found at all four scales were summed. Otherwise, the cost of the best match found among features at scale 0.5 was used (the scale that gave the best retrieval performance on its own).
- **Distinction (D):** If enabled, a small subset of features ($\sim$ 7) was selected for matching within every target object, as described in Section 3.3. Otherwise, all features were included within the target objects.

Results of this experiment are shown in Figure 7 and Table 2. The first three columns of Table 2 indicate whether each of the three algorithmic features (R, M, and D) are enabled (Y) or disabled (N), and the remaining columns provide retrieval performance statistics (note that the top row repeats the performance statistics for PDS with all its algorithmic features enabled: Y Y Y).

R	S	D	1-NN	1-Tier	2-Tier	DCG
Y	Y	Y	74.3	45.5	57.0	70.6
N	Y	Y	67.9	37.0	47.7	64.1
Y	N	Y	67.0	36.6	47.5	62.8
Y	Y	N	66.6	37.2	48.7	64.4
N	Y	N	63.4	32.7	42.6	60.6
Y	N	N	63.0	30.2	41.0	58.3
N	N	Y	54.0	28.2	38.3	56.0
N	N	N	57.0	26.7	36.2	54.7

Table 2: *Results of experiments to investigate the individual and combined value of three algorithmic features of priority-driven search (PDS). The top row represents the base PDS algorithm (Y Y Y). Other rows represent variants of the algorithms with three algorithmic features (R = rank, S = multi-scale, and D = distinctive feature selection) enabled (Y) or disabled (N). Differences in the results achieved with these variants provide insights into which aspects of the PDS algorithm contribute most to its results.*

From these results, we see that the retrieval performance of our system comes from several sources. That is, all three algorithmic features tested contribute a modest but significant improvement to the overall result. Specifically, if we consider the incremental improvements in nearest neighbor classification rates (1-NN) of the combinations shown in Figure 7, we find that multi-scale features provide 11% improvement over using the best single scale (63.4% vs. 57.0%); selecting distinctive features of target objects further boosts performance by another 7% (67.9% vs. 63.4%); and, using descriptor ranks rather than L^2 differences provides a further 9% improvement (74.3% vs. 67.9%). These three algorithmic features combine to contribute a cumulative 30% improvement in retrieval performance over the basic version of our multi-feature matching algorithm (74.3% vs. 57.0%).

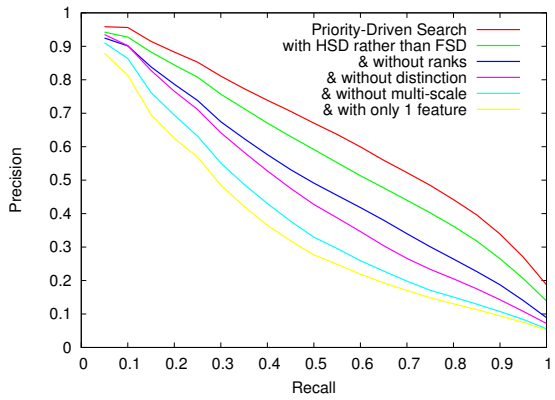


Figure 7: *Precision-recall plot showing the relative contributions of different algorithmic features of priority-driven search. The top curve (red) shows the retrieval performance of the best performing set of options for the PDS algorithm (it is the same as the red curve in Figure 6). The second curve (green) shows the result of using HSDs rather than FSDs as shape descriptors (it represents the "base configuration" for the study in Section 4.3). The third curve (blue) shows the results of using L^2 differences instead of ranks to measure feature correspondence costs; the fourth curve (magenta) shows the same, but without selecting a subset of distinctive features on target objects; the next-to-bottom curve (cyan) also disables multi-scale feature matching (all features are matched only at scale 0.5); and, the bottom curve (yellow) shows the results when finding only one point per match rather than 3. Note how the retrieval performance degrades significantly when each of these algorithmic features is disabled.*

With respect to timing, the main expense of the priority driven search implementation is establishing the initial set of pairwise feature correspondences (∼0.3 seconds per query per scale). By comparison, the time required to search for the best multi-feature match is negligible (<0.1 seconds). So, the timing results are currently dominated by the number of features considered for each target object and the number of scales considered for each feature.

Overall, we find that choosing distinctive features (D) improves both precision and speed significantly; using ranks rather than L^2 differences (R) improves precision with negligible extra compute time; and, using features at four scales (S) improves precision, but incurs four times the computational expense.

4.3. Investigation of Parameter Settings

The goal of the third experiment is to investigate in detail how various options of the priority-driven search system affect the timing and retrieval performance. Of course, there is a large space of possible options, and thus we are forced to focus our discussion on small "slices" through this space. Our approach is to center our investigation on the "base configuration" set of options described in the beginning of this section and to study how timing and retrieval statistics are affected independently as one option is varied at a time.

The results of this study are shown in Table 3(a-d) – each table studies the impact of a different option, and different rows represent a different setting for that option. Please note that rows marked with an '*' represent the same data – they provide results for the base configuration through which slices of option space are being studied.

Descriptor	Time	1-NN	1-Tier	2-Tier	DCG
SD	1.1	75.5	44.2	56.1	71.1
HSD *	1.2	74.3	45.5	57.0	70.6
FSD	2.4	83.4	51.7	63.4	75.9

(a) Shape descriptor type

Radius	Time	1-NN	1-Tier	2-Tier	DCG
0.25	0.3	62.6	31.0	41.2	58.8
0.5	0.3	67.0	36.6	47.5	62.8
1.0	0.3	63.9	37.2	48.5	63.0
2.0	0.3	60.0	33.2	43.4	59.5
Multi-scale *	1.2	74.3	45.5	57.0	70.6
All	0.6	71.3	40.6	54.2	68.0

(b) Scales used for matching shape features

# Points	Time	1-NN	1-Tier	2-Tier	DCG
64	0.6	71.9	42.4	54.2	68.6
128 *	1.2	74.3	45.5	57.0	70.6
256	4.0	75.5	47.3	59.2	71.9
512	17.6	76.6	48.6	60.2	72.6

(c) Number of sample points per object

k	Time	1-NN	1-Tier	2-Tier	DCG
1	1.2	72.9	43.4	54.9	68.9
2	1.2	72.1	44.3	55.8	69.5
3 *	1.2	74.3	45.5	57.0	70.6
4	1.2	72.8	45.4	56.8	70.3
5	1.2	71.2	44.9	56.3	69.7

(d) Number of feature correspondences per match (k)

Table 3: *Results of experiments to investigate the impact of several options on the query time (in seconds) and retrieval performance of priority-driven search.*

Impact of shape descriptor type (Table 3(a)): more verbose descriptors generally provide better retrieval performance, albeit at higher storage and compute costs. For example, the Fourier shape descriptor (FSD) provides better nearest neighbor classification rates (83.4%) than the Harmonic shape descriptor (HSD) (74.3%). However, it is also eight times bigger, and thus eight times more expensive to compare. Interestingly, the Shells shape descriptor (SD) provides retrieval performance similar that of the HSD in this test. Further study is required to determine which descriptors provide the best "bang for the buck" for specific applications and how multiple descriptors can be combined to provide

the accuracy of the most verbose ones while incurring query times of the smaller ones (Section 5).

Impact of feature scale (Table 3(b)): medium scale features (radius = 0.5-1.0) provide better retrieval performance than small and large scales in this test, and multi-scale features perform the best of all (nearest neighbor classification rates are 74.3% with multi-scale versus 67.0% with the best single scale (0.5)). Interestingly, summing the cost functions computed for matches at all four scales separately ("Multi-scale") provides better retrieval performance than matching features at all scales simultaneously ("All"). The difference is that the same set of features must match at all 4 scales in "All," while different features can be selected independently for each scale in "Multi-scale." This result seems to suggest that features persistent across multiple scales are not necessarily as useful for classification as ones that are very distinctive at a particular scale.

Impact of the number of sample points per object (Table 3(c)): including more sample points for each object improves retrieval performance in this test, at least up to 512 points. The nearest neighbor classification rate is 76.6% for 512 points per object, while it is 75.5% for 256 points, 74.3% for 128 points, and 71.9% for 64 points. Although a small set of distinctive features are ultimately selected for every target object during a preprocess, features centered at all sample points of the query object are candidates for a match, and thus the compute time for each query should be proportional to the number of points (the quadratic growth observed in this experiment is an artifact of our implementation).

Impact of number of feature correspondences (Table 3(d)): matching large numbers of features does not improve retrieval performance in this study. In fact, matching more than 3 features seems to degrade performance. This result may be because features are quite large scale and spread apart, and thus 3 features may describe the shape as well as is possible with the HSD feature representation. Interestingly, matching larger numbers of features also does not increase query times – this is because the priority-driven search algorithm is able to find good matches in time that is largely independent of the number of possible matches – it investigates only the good matches and ignores the rest.

5. Conclusion and Future Work

This paper describes an algorithm for multi-feature matching of 3D shapes with priority-driven search. The main contribution is an algorithm for searching a database for the best multi-feature matches without computing complete matches for every object. Perhaps just as valuable is the investigation of factors that contribute to speed and retrieval performance improvements in a multi-feature matching system. We find that: 1) using ranks to measure the cost of a feature correspondence is more effective that using L^2 differences directly; 2) selecting target features based on how distinctive they are of their object's class can improve both search speed and retrieval performance significantly; and, 3) matching features at different scales independently and then adding the resulting costs is an effective way to combine shape information from multiple scales.

This work suggests several areas for improvement and future work. In particular, there are three main computational bottlenecks in the system: 1) constructing shape descriptors, 2) determining the distinction of shape descriptors, and 3) generating pairwise feature correspondences. There are many simple ways to speed up these steps, including random sampling, compression, and indexing. For example, the time required to establish the best pairwise correspondences between features could be improved with standard multi-dimensional indexing schemes. We have focused our efforts in this paper on the priority-driven search algorithm, and thus we have not yet investigated these options in detail.

Another interesting option is to compute the cost of feature correspondences progressively – i.e., initially compute a conservative lower bound on the difference between shape descriptors (e.g., using SD), and only refine it for the best matches. When the correspondence rises to the top of the priority queue, the lower bound on the descriptor difference can be refined a little further (e.g., using HSD) and loaded back onto the priority queue. After the feature correspondence has reached the top of the priority queue and been refined a number of times, the full correspondence cost will be computed (e.g., using FSD) and the PDS algorithm could proceed as usual with that correspondence. This approach would utilize the priority driven strategy not only for extending matches, but also for computing correspondences in the first place.

Perhaps the most interesting question for further study is to investigate how best to recognize 3D objects from their parts. Of course, this is an active topic in computer vision, but the issues for 3D shapes are different than they are for 2D images. Our study seems to suggest that just a few shape features are sufficient to recognize most 3D objects. It will be interesting to see whether other object types follow this pattern, and whether effective algorithms can be developed using even fewer features.

References

[AFP00] AUDETTE M. A., FERRIE F. P., PETERS T. M.: An algorithmic overview of surface registration techniques for medical imaging. *Medical Image Analysis 4*, 3 (2000), 201–217. 2

[AKKS99] ANKERST M., KASTENMÜLLER G., KRIEGEL H.-P., SEIDL T.: 3D shape histograms for similarity search and classification in spatial databases. In *Proc. SSD* (1999). 2, 5

[BB76] BARROW H., BURSTALL R.: Subgraph isomorphism, matching relational structures and maximal cliques. *Inf. Process. Lett. 4* (1976), 83–84. 2

[BBM05] BERG A. C., BERG T. L., MALIK J.: Shape matching and object recognition using low distortion correspondence. In *IEEE Computer Vision and Pattern Recognition (CVPR)* (2005). 2, 3

[BKS*05] BUSTOS B., KEIM D., SAUPE D., SCHRECK T., VRANIĆ D.: Feature-based similarity search in 3D object databases. *ACM Computing Surveys 37*, 4 (2005), 345–387. 2, 4, 7

[BKS*06] BUSTOS B., KEIM D., SAUPE D., T.SCHRECK, VRANIĆ D.: An experimental effectiveness comparison of methods for 3D similarity search. *International Journal on Digital Libraries, Special issue on Multimedia Contents and Management in Digital Libraries 6*, 1 (2006), 39–54. 8

[BM92] BESL P., MCKAY N.: A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2 (1992), 239–256. 3

[BMP01] BELONGIE S., MALIK J., PUZICHA J.: Matching shapes. *ICCV* (2001). 1, 2, 4

[CJ96] CHUA C., JARVIS R.: Point signatures: A new representation for 3D object recognition. *International Journal of Computer Vision 25*, 1 (1996), 63–85. 1, 2, 5

[COTS03] CHEN D.-Y., OUHYOUNG M., TIAN X.-P., SHEN Y.-T.: On visual similarity based 3D model retrieval. *Computer Graphics Forum* (2003), 223–232. 2, 8

[CR00] CHUI H., RANGARAJAN A.: A new algorithm for non-rigid point matching. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2000), 44–51. 2

[FB81] FISCHLER M., BOLLES R.: Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. *Commun. Assoc. Comp. Mach. 24* (1981), 381–395. 3

[FHK*04] FROME A., HUBER D., KOLLURI R., BULOW T., MALIK J.: Recognizing objects in range data using regional point descriptors. In *European Conference on Computer Vision (ECCV)* (May 2004), pp. 224–237. 4, 5

[FMK*03] FUNKHOUSER T., MIN P., KAZHDAN M., CHEN J., HALDERMAN A., DOBKIN D., JACOBS D.: A search engine for 3D models. *Transactions on Graphics 22*, 1 (2003), 83–105. 5, 8

[GCO06] GAL R., COHEN-OR D.: Salient geometric features for partial shape matching and similarity. *ACM Transaction on Graphics* (January 2006). 2, 3, 5

[GD05] GRAUMAN K., DARRELL T.: The pyramid match kernel: Discriminative classification with sets of image features. In *IEEE International Conference on Computer Vision (ICCV)* (2005). 3

[GMGP05] GELFAND N., MITRA N. J., GUIBAS L. J., POTTMANN H.: Robust global registration. In *Symposium on Geometry Processing* (2005). 2, 3, 5

[HKSV02] HECZKO M., KEIM D., SAUPE D., VRANIĆ D.: Methods for similarity search on 3D databases. In *Datenbank-Spektrum* (2002), vol. 2, pp. 54–63. (In German). 2, 8

[IJL*05] IYER N., JAYANTI S., LOU K., KALYANARAMAN Y., RAMANI K.: Three dimensional shape searching: State-of-the-art review and future trends. *Computer-Aided Design 37*, 5 (2005), 509–530. 2, 4

[JH99] JOHNSON A., HEBERT M.: Using spin-images for efficient multiple model recognition in cluttered 3-D scenes. *IEEE PAMI 21*, 5 (1999), 433–449. 1, 2, 3, 5

[KFR03] KAZHDAN M., FUNKHOUSER T., RUSINKIEWICZ S.: Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Symposium on Geometry Processing* (June 2003), pp. 167–175. 5

[KPNK03] KORTGEN M., PARK G.-J., NOVOTNI M., KLEIN R.: 3D shape matching with 3D shape contexts. In *7th Central European Seminar on Computer Graphics* (April 2003). 4

[LG05] LI X., GUSKOV I.: Multi-scale features for approximate alignment of point-based surfaces. In *Symposium on Geometry Processing* (2005). 2

[LW88] LAMDAM Y., WOLFSON H.: Geometric hashing: a general and efficient model-based recognition scheme. In *Proc. ICCV* (December 1988). 3

[MBM01] MORI G., BELONGIE S., MALIK J.: Shape contexts enable efficient retrieval of similar shapes. In *IEEE Computer Vision and Pattern Recognition (CVPR)* (December 2001). 3, 4, 5

[MHKF03] MIN P., HALDERMAN J., KAZHDAN M., FUNKHOUSER T.: Early experiences with a 3D model search engine. In *Proceeding of the eighth international conference on 3D web technology* (2003), pp. 7–18. 9

[NDK05] NOVOTNI M., DEGENER P., KLEIN R.: *Correspondence Generation and Matching of 3D Shape Subparts*. Tech. Rep. CG-2005-2,ISSN 1610-8892, Friedrich-Wilhelms-Universität Bonn, June 2005. 2, 4, 5

[SF06] SHILANE P., FUNKHOUSER T.: Selecting distinctive 3D shape descriptors for similarity retrieval. In *Shape Modeling International* (June (to appear) 2006). 5

[SMKF04] SHILANE P., MIN P., KAZHDAN M., FUNKHOUSER T.: The Princeton Shape Benchmark. In *Shape Modeling International* (June 2004), pp. 167–178. 2, 5, 7, 8, 9

[SMS*04] SHAN Y., MATEI B., SAWHNEY H. S., KUMAR R., HUBER D., HEBERT M.: Linear model hashing and batch ransac for rapid and accurate object recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition* (2004). 2, 3, 5

[TV04] TANGELDER J., VELTKAMP R.: A survey of content based 3D shape retrieval methods. In *Shape Modeling International* (June 2004), pp. 145–156. 2, 4

[Vra06] VRANIĆ D.: Tools for 3d model retrieval, 2006. http://merkur01.inf.uni-konstanz.de/3Dtools/. 8

[YF02] YAMANY S., FARAG A.: Surfacing signatures: An orientation independent free-form surface representation scheme for the purpose of objects registration and matching. *IEEE Trans. Pattern Anal. Mach. Intell. 24* (2002), 1105–1120. 5