Published in final edited form as:

Knowl Inf Syst. 2015 April 1; 43(1): 157–180. doi:10.1007/s10115-014-0729-z.

Indexing Volumetric Shapes with Matching and Packing

David Ryan Koes and Carlos J. Camacho

Department of Computational and Systems Biology, University of Pittsburgh

Abstract

We describe a novel algorithm for bulk-loading an index with high-dimensional data and apply it to the problem of volumetric shape matching. Our *matching and packing* algorithm is a general approach for packing data according to a similarity metric. First an approximate k -nearest neighbor graph is constructed using *vantage-point initialization*, an improvement to previous work that decreases construction time while improving the quality of approximation. Then graph matching is iteratively performed to pack related items closely together. The end result is a dense index with good performance.

We define a new query specification for shape matching that uses minimum and maximum shape constraints to explicitly specify the spatial requirements of the desired shape. This specification provides a natural language for performing volumetric shape matching and is readily supported by the geometry-based similarity search (GSS) tree, an indexing structure that maintains explicit representations of volumetric shape. We describe our implementation of a GSS tree for volumetric shape matching and provide a comprehensive evaluation of parameter sensitivity, performance, and scalability. Compared to previous bulk-loading algorithms, we find that matching and packing can construct a GSS-tree index in the same amount of time that is denser, flatter, and better performing, with an observed average performance improvement of 2X.

Keywords

Shape retrieval; Shape indexing; Bulk-loading; Matching and packing

1. Introduction

Our goal is to develop algorithms and data structures for the rapid search of large databases of three-dimensional shapes. We are primarily concerned with applications where the volumetric properties of a shape are of more importance than the surface contours of the shape. This is the case when there are rigid requirements on what space an object may occupy that are not well reflected by general evaluations of shape similarity.

Here we describe a precise and efficient method for indexing and searching databases of millions of volumetric shapes. The main contributions of the paper are:

- a description of a novel volumetric search criterion that, unlike conventional metric-based approaches, supports the sculpting of precise shape constraints with specific and variable tolerances,

- modifications to the geometry-based similarity search (GSS) tree (Keim 1999) that improve its performance,
- an improvement of the NN-Descent algorithm (Dong et al. 2011) for approximate k -nearest neighbor graph construction that decreases the construction time of the graph while improving the quality of the approximation,
- a novel *matching and packing* algorithm for the bulk-loading of the GSS tree that results in a dense, high-quality index structure, and
- a thorough evaluation of our approach, including a sensitivity analysis of the various parameters.

2. Background

The subject of 3D shape matching has been extensively researched and has been comprehensively reviewed (Iyer et al. 2005, Tangelder & Veltkamp 2008). The 3D Shape Retrieval Contest (SHREC) is a yearly contest that evaluates shape retrieval algorithms (Li et al. 2012). The key aspect of most approaches is how the similarity of different shapes is quantified. Most commonly, 3D shapes are reduced to a vector of numerical features. The similarity between shapes is then expressed in terms of a metric, such as the Euclidean or χ^2 distances, between feature vectors. These vectors may be indexed for efficient retrieval using data structures such as metric trees (Hjaltason & Samet 2003, Samet 2006). Metric trees can be dynamically constructed from a stream of input data, but if the data set is bulk loaded, then a denser, more efficient index can be created (Aronovich & Spiegler 2010).

A wide variety of descriptors, including histograms of geometric properties (Osada et al. 2002), “bags of words” of local features (López-Sastre et al. 2013), and the Fourier coefficients of depth-buffer images of the objects (Ohbuchi et al. 2003), have been explored (Bustos et al. 2005), and each provides a different mapping from shape to a lower dimension feature vector. If the features used are not translation or rotation invariant, then shapes are typically preprocessed with a pose normalization step, such as principal component analysis (Paquet et al. 2000). However, this step is neither necessary nor desired if the shapes are pre-registered in a specific coordinate system.

Reducing shapes to feature vectors simplifies the indexing and similarity problems, but necessarily results in a loss of information about the details of the shape and makes it difficult, if not impossible, to match against a precise volumetric specification of shape. One alternative that is applicable to voxelized shapes, as opposed to ‘polygon soups,’ is to maintain an explicit representation of the shape. Octrees provide an efficient representation for storing and comparing volumetric shapes (Zhang & Smith 2009). The normalized volume difference, $\delta_{VD}(vol_1, vol_2)$, defines a similarity metric on volumetric shapes (Keim 1999, 1997) and is defined as

$$\delta_{VD}(vol_1, vol_2) = 1 - \frac{\|vol_1 \cap vol_2\|}{\|vol_1 \cup vol_2\|}$$

where $\|vol\|$ is the volume of the shape vol .

Voxelized shapes can be indexed using a geometry-based similarity search (GSS) tree (Keim 1999, 1997) which has been used to index a dataset of 800 shapes from medical imaging. As illustrated by Figure 1, in a GSS tree each node maintains a maximum included volume (MIV) and minimum surrounding volume (MSV) that bound all the shapes stored in descendants of the node. That is, the MIV of a collection of shapes is the intersection of these shapes while the MSV is the union. Previously, these MIV and MSV representations were used to perform nearest neighbor similarity search by providing bounds on δ_{VD} .

Here we describe and evaluate the straightforward use of the GSS tree to support the precise querying of shapes through the specification of minimum and maximum shape constraints as well as detail improvements to the data structure that improve its efficiency. Additionally, contrary to previous work (Keim 1999, 1997) that created GSS trees dynamically, we describe a novel method for bulk-loading a GSS tree that results in a dense and efficient search tree that supports the querying of millions of volumetric shapes.

3. Problem Description

Our goal is to support the precise and exact search of millions of volumetric shapes. In addition to supporting classical nearest neighbor searching based on volume overlap similarity, we also allow the user to sculpt precise minimum and maximum shape constraints. A minimum/maximum shape query specification allows an experienced user to precisely specify exacting shape requirements (where the minimum and maximum shapes are close together) while also supporting more relaxed shape requirements. This is in stark contrast to existing similarity metric approaches where a highly similar shape, as shown in Figure 2(d), may be less interesting than a dissimilar shape, such as shown in Figure 2(c), that matches the desired shape constraints.

A key enabling assumption we make is that the provided volumetric shapes are registered to a common coordinate system. This can be achieved using pose normalization (Paquet et al. 2000). Pose normalization has been shown to be an effective and necessary preprocessing step when using similarity metrics that are not pose invariant for the general object recognition problem (Sfikas et al. 2011, Chaouch & Verroust-Blondet 2008).

The main problem addressed by this paper is the design of data structures and algorithms to support the efficient and scalable search of a database of volumetric shapes pre-registered to a coordinate system where search queries take the form of either a nearest neighbor query or a minimum/maximum shape constraint. Ideally, searches of hundreds of millions of volumetric shapes will be able to be performed in a few seconds to support the interactive modification and refinement of the query.

4. Data Structures

Our two main data structures are octrees for shape representation and GSS trees for shape indexing.

4.1. Octree

A region octree is a classical data structure for shape representation that recursively divides space into eight uniform octants (Samet 2006). An octant that is fully contained in or excluded from a shape is marked as a leaf, while octants that partially overlap the shape are further subdivided up to a specified resolution. An octree representation scales with the surface area of an object, not its volume (Meagher 1982).

As originally described by Keim (1997), the GSS tree used a linear octree representation of shape. In a linear octree, only the leaf nodes of the octree are explicitly stored. Each node can be represented by a single byte containing the level of the leaf and a bit indicating if the leaf is full or empty. The internal structure of the tree is implicitly reconstructed during a linear scan of the leaf array. Although linear octrees are space efficient, since there is no explicit internal structure, there is no possibility of short-circuit evaluation when performing operations such as intersection or union.

Since union and intersection calculations are common both in the construction and search of the GSS tree, we design our octrees to explicitly store the internal nodes of the tree. Additionally, to accelerate the computation of δ_{VD} , the volume of each subtree is stored in each node along with the eight references to the children of the node. A reference consists of a bit specifying if the child is a leaf node or internal node and the location index of the child node. If the child is a leaf, the leaf data (full or empty) is hoisted into the location index field. We perform *double hoisting* of leaf data where, if all the children of a child node are leaves, the 8 grandchildren are condensed into an 8-bit mask stored in the grandparent. This octree representation has a storage overhead comparable to a linear octree, but has substantially better performance due to the cache of subtree volumes and short-circuiting of intersection and union calculations.

4.2. GSS Tree

The defining aspect of the GSS tree, illustrated in Figure 1, is the use of a maximum included volume (MIV) and minimum surrounding volume (MSV) as aggregate descriptors of the descendants of a tree node. Additionally, like a B-tree, a GSS tree is height balanced and each node may contain several children. Unlike most B-tree structures, the data (MIV and MSV) contained in each node is not of fixed width, so we make no attempt to page-align nodes. The ideal maximum branching factor of a tree node is explored in Section 7.2. In our implementation, the internal nodes and leaf nodes are stored in separate files. After tree construction, these files are sorted in depth-first order. Each internal node contains start and end references to the range of leaf nodes of the subtree it specifies. This supports the immediate sequential scanning of all the leaf nodes of any given subtree. In our implementation, leaf nodes contain a single shape octree.

In the original GSS tree, the MIV and MSV representations stored in a node were specified to be only as accurate as needed to distinguish among them. For example, the MIV octrees stored in a node would be rounded down to the coarsest resolution where all the MIV octrees were still not identical. This results in smaller octrees with strictly fewer levels that are faster to load and query, but comparisons become less accurate. For our queries, we find

that the inaccuracies introduced by these lower resolution approximations prevented the pruning of unproductive subtree searches resulting in a net decrease in performance.

As an alternative, we perform *local adaptive rounding* of octrees stored in the same GSS tree node. Rather than reducing the resolution of the entire tree, we consider each leaf octant of the trees separately. A leaf octant of an MIV/MSV octree is only rounded down/up to an empty/full octant if the same octant in all the other MIV/MSV octrees in the node either has an identical pattern (in which case it is rounded as well) or is a full/empty octant (see Figure 3). That is, the resolution of octants that actually discriminate between different shapes within the octant are left untouched. Essentially, the distribution of shapes stored within the GSS tree is used to indicate which segments of the octree representation can be stored in a reduced resolution. This technique results in an average reduction in octree size of approximately 20–30% while resulting in almost no increase in unproductive searches.

5. Algorithms

The usefulness of the GSS tree depends on how well similar shapes are grouped together at lower levels of the tree. The grouping of dissimilar shapes results in large MSV shapes and small MIV shapes that are uninformative in guiding the search process. The GSS tree was originally described as a dynamic data structure where shapes were added and removed one at a time (Keim 1997). This approach produces a tree layout that is dependent upon the ordering of the inputs and, while balanced, may have a highly variable distribution of node occupancies throughout the tree resulting in poor search performance. For shape databases that are updated infrequently, constructing the GSS tree search index through the bulk-loading of the input data set is a natural alternative. We describe the first GSS tree bulk-loading framework, which includes a novel improvement to approximate k -nearest neighbor (k -nn) graph creation and a novel *matching and packing* algorithm for producing dense, efficient groups of shapes. Additionally, we describe our GSS tree search algorithms. For convenience, the parameters of the algorithms are collated in Table 1.

Algorithm 1

CreateIndex

Input : unordered array of shapes to index
Result: root of GSS index

nodes $\leftarrow$ shapes // convert shapes to leaf nodes

while size(nodes) > 1 **do**

 nextlevel $\leftarrow$ []

 CreateNextLevel (nodes,nextlevel)

 nodes $\leftarrow$ nextlevel

return nodes.front()

Algorithm 2**CreateNextLevel**

Input : Tree nodes to combine to form the nextlevel

if $size(nodes) > p$ **then**

nextlevel.add(Pack(nodes))

else

// *BulkLoad* partitioning (Aronovich & Spiegler 2010)

sample $\leftarrow$ SelectSampleSet(nodes, c)

nodePartitions $\leftarrow$ PartitionBL(nodes, sample)

foreach part *in* nodePartitions **do**

CreateNextLevel (part,nextlevel)

Algorithm 3**Pack**

Input : nodes to densely pack m per a parent node

Result: parent nodes

for $i \leftarrow 1$ **to** $\log_2(m)$ **do**

knn $\leftarrow$ CreateKNNGraph(nodes)

mergedNodes $\leftarrow$ []

foreach edge *in* MaximalMatching(knn) **do**

parent $\leftarrow$ { }

(n1, n2) $\leftarrow$ edge

parent.MSV $\leftarrow$ n1.MSV $\cup$ n2.MSV

parent.MIV $\leftarrow$ n1.MIV $\cap$ n2.MIV

if $i == 1$ **then**

parent.children $\leftarrow$ [n1, n2]

else

parent.children $\leftarrow$ [n1.children, n2.children]

mergedNodes.add(parent)

nodes $\leftarrow$ mergedNodes

return nodes

5.1. Bulk Loading Framework

The goal of bulk-loading a data structure like the GSS tree is to create a dense tree with maximum fanout that provides good search performance. Additionally, a bulk-loading algorithm must be able to handle a large input data set that may not fit in memory.

As a starting point for our framework, we use the *BulkLoad* algorithm (Aronovich & Spiegler 2010). This algorithm, originally designed to initialize a CM-tree metric tree structure (Aronovich & Spiegler 2007), uses a top-down partitioning scheme to build each layer of the tree from the bottom-up. The input data is partitioned into c partitions by (1)

randomly sampling cL data items (where L is a sampling parameter with a recommended value of five), (2) agglomeratively clustering this sample into c clusters, (3) identifying the most central element of these clusters, and then (4) assigning each data item in the input to the cluster center it is closest to. A single partitioning step performs a full scan of input data only once while performing $O(cn)$ distance computations in the cluster assignment step, and, as long as c and L are small, the $O(c^2L^2)$ agglomerative clustering can be performed in-memory with little practical overhead. This partitioning step is recursively applied until the resulting partition contains fewer than the maximum allowed number of items in a node, m , at which point the partition is converted into a tree node with the corresponding data items as its children. Once the input has been fully decomposed into tree nodes, these nodes are then treated as individual data items that are then partitioned to create the next highest level in the tree structure. This process is repeated until a single root node is obtained.

The *BulkLoad* algorithm requires that a distance function be defined both between individual data items, in our case the δ_{VD} metric between shapes, and between nodes, which represent a subtree of items. In the GSS tree every node contains a representative MIV and MSV shape. We define the distance between nodes to be the average of the δ_{VD} of the MIV and MSV shapes. Since this distance is the average of two metrics, it is also a metric. In fact, it is a generalization of shape distance since a single shape can be represented by a pair of identical MIV and MSV shapes. For simplicity, we also refer to this similarity metric as δ_{VD} .

Unfortunately, the *BulkLoad* algorithm does not result in a particularly dense tree since the partitioning scheme produces variable sized clusters. More subtly, the grouping of items around a “central” item, although appropriate for the CM-tree data structure, is not particularly well-suited for GSS tree construction, as illustrated in Figure 4. Instead, the desired grouping of shapes (or nodes representing shapes) is the grouping that results in the tightest, most specific MIV and MSV representations.

To address these issues we have developed a hybrid partitioning/packing framework that uses the *BulkLoad* algorithm to efficiently partition items (either individual shapes or internal GSS tree nodes) into clusters that are substantially larger than the desired node fanout, but remain small enough that the cluster data fits in memory. We then *pack* the cluster data into groups of size m , each of which is converted to a node in the GSS tree layer that is under construction. Once a tree layer is completed, the next highest layer is constructed by repeating the process. The pseudocode for this framework is provided in Algorithms 1 and 2.

5.2. Approximate K-NN Graph Construction

In order to generate a quality packing of n items, it is necessary to compute the pairwise distances between items. Computing all $O(n^2)$ distances is prohibitively expensive and unnecessary since only the distances between nearby items, which are candidates for packing, are relevant. Instead, a k -nearest neighbor graph, where each node is only connected to its k closest neighbors, can be used. Efficient, $o(n^2)$, construction of an optimal k -nn graph over a general metric space with an intrinsic high-dimensionality remains an open problem (Chávez et al. 2001). However, efficient algorithms for constructing approximate k -nn graphs have been developed (Paredes et al. 2006, Dong et al. 2011). We

use the *NN-Descent* algorithm (Dong et al. 2011) to construct an approximate k -nn graph of items.

The *NN-Descent* algorithm first constructs an initial graph where each node, representing an item, is connected to a random sampling of k other nodes. This candidate k -nn graph is then improved through a local search around each node that compares the node with its neighbor's neighbors. Each iteration of *NN-Descent* is $O(k^2n)$. Typically only a few iterations are needed to converge to a local minimum, and, at least for lower dimensional inputs, this local minimum is observed to be close to the global minimum (the true k -nn graph) (Dong et al. 2011).

We improve upon the *NN-Descent* algorithm by replacing the random initial graph with a *vantage-point initialized* graph. We project each item onto a lower dimensional Euclidean space by computing a vector of distances to a set of v vantage-point objects. A k -nn graph between these low-dimensional distance vectors can be computed efficiently using a spatial data structure. The nearest neighbor relation in this vantage space has been used directly to perform image retrieval (Vleugels & Veltkamp 2002). However, the k -nn graph in vantage space does not correspond directly to the k -nn graph in the original, higher-order metric space. Instead, we use the k -nn graph in vantage space to initialize the *NN-Descent* algorithm, which then further refines the k -nn graph.

We choose the v vantage-point objects with a greedy farthest-first selection process that requires $O(vn)$ distance computations. The k -nn graph in vantage space is constructed using the kd-tree module of the ANN library (Mount 2010). We run the *NN-Descent* algorithm to completion without any early termination in order to generate the most accurate graph. We demonstrate in Section 7.2 that our vantage-point initialization scheme both reduces the time to construct the k -nn graph and results in a higher-quality approximation.

5.3. Matching and Packing

Optimal packing into groups larger than two is NP-complete (Pagel et al. 1995), but the well-studied graph matching algorithm (Edmonds 1965) can be used to find optimal *pairings* of objects. A maximum matching is the set of pairwise non-adjacent edges in a graph with the maximum total edge cost. We greedily pack nodes into groups of size m (the maximum fanout of a GSS tree node) by iteratively finding a maximum graph matching. We compute the maximum matching of the approximate k -nn graph where the edge costs are set to $1 - \delta_{VD}$. The paired nodes are precisely those whose combined MIV and MSV representations will change the least. The MIV and MSV of all the paired nodes is computed to form a new set of nodes from which a new k -nn graph is generated, and the matching process is repeated $\log_2 m$ times until all the nodes have been joined into a set of parent nodes with a maximum fanout of m . We do not perform a perfect matching, which requires that every vertex in the graph be incident upon an edge in the matching. This allows outliers to remain unmatched and keeps the most closely related nodes packed together. The pseudocode for this process is shown in Algorithm 3.

Graph matching has been extensively studied and, for graphs with v vertices and e edges, the best worst-case results are $O(\sqrt{ve})$ (Micali & Vazirani 1980). However, simpler algorithms

with worse asymptotic bounds perform well in practice, particularly on sparse graphs (Kolmogorov 2009). In fact, matching of a random sparse graph with a constant average degree can be performed in expected $O(v)$ time (Chebolu & Frieze 2010). Consequently, even though we use a $O(v \log v)$ matching algorithm from the LEMON graph library (lem n.d.), we expect and observe excellent real-world performance, as quantified in Section 7.5. The use of a k -nn graph with $O(kn)$ edges instead of a complete graph with $O(n^2)$ edges is essential to achieving this performance, and the maximal matching of the k -nn graph will be identical to the maximal matching of the complete graph as long as the k -nn graph samples the appropriate edges. This makes the quality of the matching sensitive to both the quality of the k -nn graph approximation and the value of k , a dependency that is explored in Section 7.2.

5.4. Search

We support two forms of search: nearest neighbor search and search by shape constraints.

5.4.1. Nearest Neighbor—The GSS tree was originally designed to support nearest neighbor search (Keim 1999). The MIV and MSV representations of each node can be used to compute minimum and maximum bounds on the volume overlap, δ_{VD} , between the query object and all subtree leaves, and this bound is used to prune the search of the tree. The minimum, δ_{min} , and maximum, δ_{max} , bounds for a given query volume, vol , are computed as follows:

$$\delta_{min}(vol, MIV, MSV) = 1 - \frac{\|vol \cap MSV\|}{\|vol \cup MIV\|}$$

$$\delta_{max}(vol, MIV, MSV) = 1 - \frac{\|vol \cap MIV\|}{\|vol \cup MSV\|}$$

Subtrees are explored by processing those nodes with the largest $\delta_{min} + \delta_{max}$ value first. As leaf nodes are explored, the current best volume overlap value is maintained. When searching for the k nearest neighbors, the k th best value is retained as the best solution. Subtrees are not explored if the bounds for the subtree indicate that its descendants will not improve upon the current best solution.

5.4.2. Shape Constraints—The search algorithm for identifying volumetric shapes that match a provided minimum/maximum shape constraint follows naturally from the structure of the GSS tree. The subtree beneath a node in the GSS tree is explored if and only if the MIV of the node is contained within the maximum shape constraint and the MSV of the node fully contains the minimum shape constraint. If the minimum shape is fully contained within the MIV and the maximum shape fully contains the MSV, then the descent of the tree can be short-circuited and the leaves of the subtree read directly.

6. Implementation

Our implementation is written in C/C++. Database creation takes as input either a directory of voxelized shapes in the MIRA format, or a file of shapes defined by molecular structures. In either case the shapes are assumed to be pre-aligned to the desired coordinate system. Voxelized shapes are represented by an octree with a maximum dimension and resolution determined by the user. These octrees are written out to disk along with the original shape data as they are created. The levels of the GSS tree are then created as previously described. When creating a new level, the previous level is opened as a memory mapped file while the new level is written to a new file to minimize the memory requirements of index construction. After the levels of the GSS tree are constructed, they are put in depth first order into a file containing the internal nodes and a separate file containing only leaves. During this optimization pass, local adaptive rounding is applied to the MIV and MSV octrees.

The OpenBabel toolkit (O'Boyle et al. 2011) is used to parse and manipulate molecular data. The molecular surface of each structure is computed analytically (Connolly 1983) using a probe radius of 1.4Å. The resulting shape is then voxelized to a .5Å resolution and has a maximum dimension of 32Å.

Nearest neighbor search takes as input either a MIRA voxel file or a molecular structure file. Shape constraint search may take MIRA voxel files of the desired minimum and maximum shape constraints, or may take two molecular structure files that define an included shape and an excluded shape. The included shape corresponds to the minimum shape constraint. Any matching shape must fully contain the included shape. The excluded shape represents the area where a matching shape should not overlap. Its inverse corresponds to the maximum shape constraint. In both cases, a gap distance may be specified by which to reduce the voxelized shape of the included shape and the uninverted excluded shape. Shapes are reduced by iteratively removing the appropriate number of surface voxels to achieve the appropriately sized gap between the original and reduced shape. The larger the gap between the two shape constraints, the more likely that matching shapes will be found. An illustration of different possible query shapes is provided in Figure 5. The GSS tree is accessed through memory-mapped files.

The complete source code is available under a GNU General Public License at <https://github.com/dkoes/shapedb>.

7. Results

Here, we investigate the sensitivity of our approach to various parameters and assess its performance and scalability. We first describe the benchmarks and benchmarking methodology used. Then we investigate the sensitivity of our approach to various parameters and select a default set of algorithmic parameters. Additionally, we compare our novel matching and packing algorithm to prior work. Finally, we evaluate the shape recognition performance and scalability of our approach.

7.1. Benchmarks

We consider two sets of benchmarks: the SHREC 2012 Generic Shape Retrieval Benchmark and a custom molecular shape benchmark we define here. The SHREC benchmark allows us to directly compare against published results, but is limited by its small size of 1200 shapes. In contrast, our custom molecular shape database contains millions of shapes. This large size allows us to meaningfully evaluate parameter sensitivity under both IO-bound and CPU-bound conditions.

All performance results are for single-threaded execution. All timing results are the average of at least three measurements. Queries are evaluated using a 3.33GHz Intel Core i7 975 with 12GB of RAM running Ubuntu Linux. Data is stored on two 7200RPM hard drives in a striped software RAID. *IO-bound* performance is measured by clearing the file system cache before each query so that all accesses to the GSS tree result in disk I/O. *CPU-bound* performance is measured by executing the same query multiple times in the same session and ignoring the performance of the first query since it loads accessed data into the file system cache. Real-world performance on large databases will be in between these two extremes since portions of the GSS tree, particularly the highest levels, are expected to be resident in the file system cache. Furthermore, in situations where queries are interactively refined, each successive query is similar to the previous and so query performance will be mostly CPU-bound rather than IO-bound.

7.1.1. SHREC 2012 Benchmark—The SHREC Generic Shape Retrieval Benchmark (Li et al. 2012) consists of 1200 generic shapes uniformly distributed among 60 different categories. We utilize the provided evaluation methodology (shr n.d.) where a shape retrieval method is evaluated using a complete 1200×1200 distance matrix between the objects. This assesses the overall performance of the distance metric, which in our case is the normalized volume difference, δ_{VD} , applied to pose normalized voxelizations of the benchmark shapes. Additionally, we use the SHREC benchmark to evaluate the run-time performance of nearest neighbor and shape constraint search and to compare our approach against the published results of the 2012 SHREC exercise.

We preprocess the shapes of the SHREC benchmark with MeshLab (mes n.d.) and the Modular toolkit for Data Processing (MDP) (Zito et al. 2009). Meshlab is used to remove isolated vertices (noise) from each shape and to center the shapes. The PCA module of MDP is then used to align each shape. The resulting normalized shapes are then voxelized using binvox (Min n.d., Nooruddin & Turk 2003). We evaluate three different resolutions of voxelization: 256³, 128³, and 64³.

In order to better evaluate the effect of database size on the search algorithm, we sample smaller benchmark sets of 300, 600, and 900 shapes from the SHREC dataset. Additionally, we create larger benchmark sets of size 2400, 3600, and 4800 by generating alternative poses of each shape with respect to the three coordinate axes. The set of 2400 shapes contains all the original shapes plus these shapes rotated around the x-axis, the next largest set adds shapes rotated around the y-axis, and the largest set adds shapes rotated around the z-axis. In addition to allowing us to further investigate the scaling properties of our search algorithm, these larger datasets also allow us to evaluate the volume difference metric when

applied to multiple pose alignments of a shape. In this case, only the best volume difference of any alignment of a shape is considered.

To compute average nearest neighbor query time we search for the 1, 2, 4 and 8 nearest neighbors of each of the 1200 benchmark shapes and take the average time across all 1200 queries. Only the search time is measured; time spent voxelizing the inputs or writing out the results is omitted. Additionally, we consider the unindexed performance of performing a simple linear scan of the octrees to compute the volume difference of a query with respect to the entire set of shapes. Since even the largest benchmark is entirely memory resident we only consider CPU-bound performance when evaluating SHREC.

In order to evaluate shape constraint search on the SHREC benchmark we create optimal shape constraints for each shape category by computing the intersection and union of all the shapes in the category to obtain the respective minimum and maximum shape constraints. These constraints are optimal in that they are the tightest possible constraints that include all members of each category. The distance matrix for shape constraint search is computed by assigning a maximum distance to any shape that does not match the constraints of the query category and δ_{VD} to those that do. This results in a potentially asymmetric distance matrix and represents an upper-bound of the shape retrieval performance obtainable through the use of shape constraints.

7.1.2. Molecular Benchmark—In order to better assess the performance of our algorithms at scale, we created large libraries of molecular shapes containing up to 22 million shapes. We use these shape libraries to evaluate parameter sensitivity, our packing algorithm, and the scalability of shape constraint based search. Importantly, although we make every effort to design a synthetic benchmark with realistic properties, the focus here is on evaluating our algorithms and data structures on a volumetric shape database that is large enough to not be memory resident. Future work will more thoroughly investigate the efficacy of volumetric molecular shape matching as applied to the virtual screening of compounds.

In order to construct our benchmark, we make use of the AnchorQuery library of synthetically accessible compounds (Koes et al. 2012). These compounds have all been defined to include an *indole*, a specific molecular fragment with a planar double 5–6 ring, as shown in Figure 7. AnchorQuery includes more than 400 million shapes representing more than 5 million compounds, all of which contain this fragment. We generate benchmark data sets by sampling uniformly from the 22 AnchorQuery reaction classes with more than a million available shapes and then aligning the selected molecular shapes to a uniform coordinate system centered on the indole fragment. We create four libraries with 22 thousand, 220 thousand, 2.2 million and 22 million molecular shapes respectively.

In our evaluation of our molecular benchmark, we primarily consider shape constraint search in order to focus on the most novel aspect of our approach. In order to define realistic query shape constraints, we extract structures of bioactive small-molecules bound to receptor proteins from the DUD (Huang et al. 2006) and CSAR (Dunbar et al. 2011) datasets where the small-molecule possesses the desired molecular fragment. This results in a total of 57 complexes. These small-molecules are then aligned and included in all the benchmarking

sets to ensure there is always at least one match. A minimum shape constraint was derived from the small-molecule and a maximum shape was derived from the receptor as described in Section 6. Queries with different precisions were constructed by applying gap distances of 1.0, 1.5, and 2.0 Ångströms to both the minimum and maximum shapes, as illustrated in Figure 5. A larger gap distance corresponds to a less precise query that is expected to return more results. For instance, with the 2.2 million sized library, the 1.5Å gap query returns an average of 66 shapes while the 2Å gap query returns an average of 10195 shapes.

Average shape query times are the average of the 57 queries. Only the search time is measured; time spent voxelizing the inputs or writing out the results is omitted. Due to practical considerations, index construction was performed in parallel on a 3.33GHz Intel Xeon X5680 storage server. Each process shared 96GB of RAM, but was allocated a dedicated core and hard drive. The creation time of each index was measured four times and the average of the best three times is reported. Unless otherwise specified, error bars show the standard error of the measurement.

7.2. Parameter Sensitivity

We consider the effect of different choices of parameters on index creation time and index quality, as measured by the average IO-bound and CPU-bound shape constraint query time. Due to its larger, more meaningful size, we utilize the molecular benchmark for this evaluation, but limit ourselves to the 2.2 million shape benchmark for practical purposes. Additionally, for purposes of evaluating parameter sensitivity, we consider only the average query performance of a shape constraint search with a gap distance of 1.5.

We evaluate the relationship between the number of k nearest neighbors computed in the k -nn graph construction and the number v vantage-points used to initialize the k -nn approximation. We also consider the relationship between the number of clusters, c , generated by the *BulkLoad* algorithm and the maximum cluster size, p , at which we perform packing. That is, the algorithm will recursively partition the input set into c clusters until the cluster size is less than or equal to p , at which point the cluster is packed into dense GSS tree nodes. Finally, we consider the effect of the node fanout, m , which is the maximum number of children of a GSS tree node. Unless otherwise specified, we use default values of $c=8$, $p=32768$, $k=8$, $v=32$, and $m=16$. These parameters are defined in Table 1.

The ability of the k -nn graph approximation to correctly sample the best edges for matching in the complete graph is a limiting factor of the performance of the matching and packing algorithm. Figure 8 shows the effect of different values of k and of v . For $k = 4$, where the quality of the matching is particularly sensitive to errors in the k -nn approximation, there is a noticeable improvement in query performance for larger values of v . This is particularly clear for CPU-bound performance. For $k = 8$, the influence of v on query time is less clear. However, vantage-point initialization results in a substantial reduction (more than 2X) in index construction time relative to random initialization, particularly for larger values of k .

The effect of the number of clusters, c , and the maximum size of the clusters, p , is shown in Figure 9. The case of $p = 0$ is equivalent to the standard *BulkLoad* algorithm and provides a baseline for comparison. Partitioning the data into a larger number of clusters reduces the

number of recursive splits, but increases the cost of each split resulting in longer construction times. In contrast with previous findings (Aronovich & Spiegler 2010), we find that increasing the number of clusters does result in improved query performance, possibly because a larger number of clusters can result in a more discriminatory partition of the input that more cleanly separates out outliers. Increasing p results in a clear and substantial improvement in query performance, and as p increases the effect of the choice of c is diminished. Small values of p actually improve index construction times relative to a clustering-only approach while simultaneously exhibiting better IO-bound query performance. For CPU-bound performance, small amounts of packing can result in a slow down relative to just clustering. This is because the clustering algorithm produces relatively sparse search trees with a significantly lower average fan-out per a node. Since the CPU-bound case incurs no page-fault penalty for accessing additional nodes, the smaller number of evaluations per a node results in faster overall performance. This effect is also evident in Figure 11.

At the expense of longer construction times and larger memory requirements, larger values of p result in better query performance, although with diminishing returns for $p > 2^{15}$. Packing has the effect of evenly and densely distributing items in the bottom of the tree, resulting in a shorter, more space efficient tree. For example, the shape databases created without packing are 18GB in size, while those with $p = 128$ are 15GB. In general, matching and packing halves the time it takes to perform a query relative to a clustering-only approach for the same cost in index construction time.

The effect of the maximum node fanout, m , is shown in Figure 10. Only average query performance is shown. There was no significant relationship between m and index creation time since, while m determines the number of items packed into a tree node, it does not significantly affect the overall number of matchings, which is logarithmic with the input size. Smaller values of m result in better CPU-bound performance since less unnecessary work (the evaluation of uninteresting children) is performed at each node. In contrast, larger values of m result in better IO-bound performance, up to a point, since evaluating fewer, larger nodes results in fewer page-faults. The optimal choice of m is heavily dependent on the size of the items in the node, the size of the file system readahead cache, and the computational cost of evaluating an item. We find that $m = 16$ provides a good compromise between IO-bound and CPU-bound performance.

7.3. Packing Performance Comparison

We consider two alternative means of packing in Figure 11. We evaluate the packing algorithms using shape constraint queries with a gap size of $1.5\text{\AA}$ performed on the 2.2 million shape molecular benchmark. A greedy algorithm matches pairs of unmatched items in the order they are found in a sorted list of distances between items. This greedy algorithm is a 2X approximation of the optimal matching algorithm (Ausiello 1999). Another approach to packing is to assign a linear order to the dataset which can be done in a mathematically precise way using spectral graph theory (Mokbel et al. 2003). We evaluate these algorithms using the complete graph of distances between items and so set $p = 2048$ to reduce the number of distance computations. For comparison purposes, we also evaluate our matching

and packing algorithm using the complete graph rather than the approximate k -nn graph. Since the matching and spectral packers have running times that scale differently with a complete graph compared to the greedy packer, we do not compare index construction times.

The three packing algorithms are compared to clustering without packing, which is equivalent to the *BulkLoad* CM-tree initialization algorithm (Aronovich & Spiegler 2010). Spectral packing does not provide any significant improvement in performance compared to clustering, indicating that it is not an effective means of grouping related items. Greedy packing improves upon clustering, but does not perform as well as matching, illustrating the advantage of using an optimal, rather than approximate, algorithm.

Although the performance difference between matching and greedy packing for $p = 2048$ is slight, this difference increases for larger values of p as shown in Figure 12. Unlike in Figure 11, here the k -nn graph construction is used. For small values of p , greedy packing has surprisingly erratic performance, possibly due to differences in how the two algorithms treat outliers. For larger values of p , the difference is pronounced for both IO-bound and CPU-bound performance with greedy matching producing a 2X slowdown and only improving index construction time by a small percentage.

7.4. Shape Retrieval Performance

The performance of the volume difference metric, δ_{VD} , at identifying related shapes in the SHREC database is shown in Table 2 and the discounted cumulative gain is shown in Figure 13. The performance of this simple metric is roughly equivalent with the lowest performing SHREC 2012 submission, and there is very little decrease in performance as lower resolution voxelizations are used. This is unsurprising since the volume difference metric, unlike the SHREC submissions, is not pose invariant, and we use a basic pose alignment method. Indeed, if multiple pose alignments are considered, the discounted cumulative gain improves from 0.50 to 0.57. This improvement is at least partially due to the fact that our generic PCA alignment does not necessarily orient shapes in a consistent and natural direction, as shown by the overlay of rocket shapes in Figure 6(b). Previous work has demonstrated that more sophisticated alignment and normalization methods improve the performance of similarity metrics that are not pose invariant (Sfikas et al. 2011, Chaouch & Verroust-Blondet 2008).

More relevant to our volumetric shape representation is the shape retrieval performance using shape constraints. Optimal shape constraints, as shown in Table 2 and Figure 13, outperform even the best SHREC 2012 submissions, even using the lowest voxel resolution. Since these shape constraints are the strictest possible constraints for each category, their shape retrieval performance is an upper bound of what is possible for our set of aligned shapes. However, a more sophisticated shape alignment process might serve to further improve the shape retrieval performance while resulting in more natural shape constraints (such as a rocket constraint where all the fins are oriented the same way, unlike in Figure 6).

7.5. Query Time

The speed of a query of the GSS tree indexing structure depends on the type and breadth of the query and the contents of the database. As a shape constraint query is relaxed to match more shapes, the speed of the query will approach that of a linear scan. On the other hand, a shape constraint query that is narrowly defined or substantially different from most of the database shapes may only have to explore a single branch of the tree (or terminate even earlier if the query matches no shapes), resulting in logarithmic search. In contrast, k nearest neighbor search always returns a limited number of shapes (k), but the speed of the query in the GSS tree is limited by how well the current k th best match prunes the remaining search tree. If the current k th best match is not significantly closer to the query than the remaining shapes in the tree, the speed of the query will approach that of a linear scan. Alternatively, if the k shapes identified in the first descent of the tree are distinctly different from the remaining shapes, then the speed will scale logarithmically with the database size. Finally, query speed is dependent on the voxel resolution of the stored shapes. Higher resolution shapes result in larger database sizes and octree operations, such as computing the volume difference, take longer to perform.

Average CPU-bound query times for the SHREC 2012 benchmark are shown in Figure 14 and Table 3. This benchmark is small enough that the entire database can reside in memory. The total size of the database, including both the index and the objects, is 20MB, 89MB, and 401Mb for the 64^3 , 128^3 , and 256^3 resolution shapes respectively. As shown in Table 2 and Figure 13, the decrease in resolution has minimum effect on shape retrieval performance. However, as shown in Table 3, there is the expected substantial increase in search time as the shape resolution is increased. The more shape comparison operations a query invokes, the more prominent this increase. For example, linear scan is more than 30 times slower at 256^3 resolution than 64^3 resolution, but only 8 times slower for shape constraint search. To put these numbers in perspective and compare against a more traditional, non-volumetric, feature vector approach, we evaluated our SHREC 2012 benchmark using the 3DSP method (López-Sastre et al. 2013, Redondo-Cabrera et al. 2012), which currently is the only submission to SHREC 2012 with publicly available source code. This method achieves an average query time of 0.03 seconds using a χ^2 distance metric and 0.001 seconds with a histogram intersection distance metric with the default settings. Other such approaches will likely have similarly large ranges of query times depending on the distance metric and size of the feature vector. Our approach lies within this range, but, more importantly, because shapes are indexed, can scale sub-linearly with the size of the database.

The scaling of query time performance with respect to SHREC benchmark size is shown in Figure 14(a). Despite the fact that, by design, the shape constraint queries all return at least 20 shapes, compared to the smaller result sets of the k nearest neighbor queries, shape constraint queries are significantly faster. This is because shape constraint checking is typically a faster octree operation than volume difference due to short-circuit evaluations and because these shape constraints are narrowly enough defined that only a limited number of comparisons are necessary to identify the matching shapes in the GSS tree.

The performance of the k nearest neighbor queries is limited by the size and content of the shape database. The single nearest neighbor search does reasonably well because all queries in our benchmark have an exact match in the database. Once this exact match, which has a volume difference of zero, is found, the rest of the tree can be meaningfully pruned. Larger values of k are indicative of performance when there is no exact match. In this case, the search tree is only pruned at the lowest levels, if at all, resulting in minimal improvements in search times. However, these improvements become more noticeable as the size of the database is increased. As expected, query performance is highly dependent on the composition of the query, as shown by the box plots in Figure 14(b). Even linear scan exhibits query time variability since different query shapes will induce different amounts of short-circuit behavior when computing the volume difference.

The molecular shape benchmark provides a means to evaluate queries on a large shape database and query performance values for this benchmark are shown in Figures 15 and 16. When searching larger databases, the benefit of the GSS tree index for performing nearest neighbor search is more apparent. As shown in Figure 15(a), linear scan of the 2.2 million molecular shape database is more than 8 times longer than the average query time of the 8th nearest neighbor search and more than 2800 times longer than shape constraint search. Still, as with the SHREC benchmark, a distinct difference is seen between single nearest neighbor search, where an exact match is found, and other nearest neighbor searches. Additionally, as illustrated by Figure 15(b), different query shapes result in a wide range of k nearest neighbor search times, particular for larger values of k .

The IO-bound and CPU-bound average performance of shape constraint queries constructed using three different gap sizes across the molecular shape benchmark databases is shown in Figure 16. Note that search times are shown on a log scale and that error bars indicate the range between the fastest and slowest queries. Also shown is the performance of performing a shape constraint linear scan over the full set of shapes. This is faster than a volume difference linear scan since shape constraint checking is more likely to induce short-circuited octree operations.

The performance of our GSS tree index is clearly better than linear scan. Particularly for narrow, precise queries (gap size 1.0), the method scales sub-linearly with respect to database size. Our approach also benefits from requiring a smaller memory footprint. For example, the 22 million shape database is too large to fit in main memory, and so linear scan is always IO-bound when searching this database. In contrast, the CPU-bound performance when searching for queries with gap size 1.0 or 1.5 is orders of magnitude faster than IO-bound when using the GSS tree index since the whole working set fits in memory. Of course, for sufficiently broad queries, indexing does not provide any benefit. This is the case for the slowest query with gap size 2. This query returns more than a tenth of the database.

8. Conclusions

We have described novel data structures and algorithms for performing rapid and scalable searches of large numbers of volumetric shapes. Additionally, we have shown that vantage-point initialization accelerates and improves approximate k -nn graph construction, which is

a key component in many web related applications (Dong et al. 2011). Furthermore, our matching and packing algorithm should be readily adaptable to any tree-like indexing structure for storing metric data.

We have evaluated our approach using the SHREC 2012 benchmark dataset and demonstrated sub-linear scaling with database size and object recognition performance in the same range as the current state of the art. A fundamental advantage of our approach is it deals directly with volumetric constraints and so supports an explicit shape specification in the form of minimum and maximum shape constraints. Shape constraints have the advantage that they can eliminate globally similar shapes that lack necessary volumetric features (such as clashing with a receptor in the case of molecular shape matching or overlapping with a connecting component in the case of computer aided design). Although constructing well-aligned volumetric shape libraries and defining appropriate shape constraints present their own set of challenges, properly constructed shape constraints have the potential to perform exceedingly well at object retrieval, and we have described the first efficient data structures and algorithms for performing shape constraint search of volumetric shapes.

Acknowledgments

This work was supported by the National Institute of Health [R01GM097082]. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institute of General Medical Sciences or the National Institutes of Health.

References

- Aronovich L, Spiegler I. CM-tree: A dynamic clustered index for similarity search in metric databases. *Data & Knowledge Engineering*. 2007; 63(3):919–946.
- Aronovich L, Spiegler I. Bulk construction of dynamic clustered metric trees. *Knowledge and Information Systems*. 2010; 22(2):211–244.
- Ausiello, G. Complexity and approximation: Combinatorial optimization problems and their approximability properties. Springer Verlag; 1999.
- Bustos B, Keim DA, Saupe D, Schreck T, Vrani DV. Feature-based similarity search in 3D object databases. *ACM Computing Surveys*. 2005; 37(4):345–387.
- Chaouch, M.; Verroust-Blondet, A. A novel method for alignment of 3d models; *Shape Modeling and Applications*, 2008. SMI 2008. IEEE International Conference on; 2008. p. 187-195.
- Chávez E, Navarro G, Baeza-Yates R, Marroquín JL. Searching in metric spaces. *ACM Comput. Surv.* 2001; 33(3):273–321. **URL:** <http://doi.acm.org/10.1145/502807.502808>.
- Chebolu P, Frieze A. Finding a maximum matching in a sparse random graph in $O(n)$ expected time. *Journal of the ACM (JACM)*. 2010; 57(4):24.
- Connolly ML. Analytical molecular surface calculation. *Journal of Applied Crystallography*. 1983; 16(5):548–558. **URL:** <http://dx.doi.org/10.1107/S0021889883010985>.
- Dong, W.; Moses, C.; Li, K. Proc. of the Intl. WWW Conf. ACM; 2011. Efficient k-nearest neighbor graph construction for generic similarity measures; p. 577-586.
- Dunbar JB, Smith RD, Yang C-Y, Ung PM-U, Lexa KW, Khazanov NA, Stuckey JA, Wang S, Carlson HA. CSAR Benchmark Exercise of 2010: Selection of the Protein-Ligand Complexes. *J Chem Inf Model*. 2011; 51(9):2036–2046. [PubMed:21728306] [PubMed Central:PMC3180202] [doi:10.1021/ci200082t] **URL:** <http://pubs.acs.org/doi/abs/10.1021/ci200082t>. [PubMed: 21728306]
- Edmonds J. Paths, trees, and flowers. *Canadian Journal of mathematics*. 1965; 17(3):449–467.
- Hjaltason GR, Samet H. Index-driven similarity search in metric spaces (Survey Article). *ACM T. Database Syst*. 2003; 28(4):517–580.

- Huang N, Shoichet BK, Irwin JJ. Benchmarking sets for molecular docking. *J Med Chem.* 2006; 49(23):6789–6801. [PubMed: 17154509] [PubMed Central:PMC3383317] [doi:10.1021/jm0608356]. [PubMed: 17154509]
- 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.* 2005; 37(5):509–530.
- Keim, DA. Habilitation thesis. University of Munich; 1997. Efficient Support of Similarity Search in Spatial Data Bases.
- Keim, DA. Proc. of the Intl. Conf. on Management of Data. New York, NY, USA: ACM; 1999. Efficient geometry-based similarity search of 3D spatial databases; p. 419-430. URL: <http://doi.acm.org/10.1145/304182.304219>
- Koes D, Khoury K, Huang Y, Wang W, Bista M, Popowicz GM, Wolf S, Holak TA, Dömling A, Camacho CJ. Enabling Large-Scale Design, Synthesis and Validation of Small Molecule Protein-Protein Antagonists. *PLoS ONE.* 2012; 7(3):e32839. EP-. [PubMed:22427896] [PubMed Central:PMC3299697] [doi:10.1371/journal.pone0032839]. [PubMed: 22427896]
- Kolmogorov V. Blossom V: A new implementation of a minimum cost perfect matching algorithm. *Math. Prog. Comp.* 2009; 1(1):43–67.
- lem (n.d.). LEMON Graph Library, version 1.2.2. <http://lemon.cs.elte.hu>.
- Li, B.; Godil, A.; Aono, M.; Bai, X.; Furuya, T.; Li, L.; López-Sastre, R.; Johan, H.; Ohbuchi, R.; Redondo-Cabrera, C., et al. Proceedings of the 5th Eurographics conference on 3D Object Retrieval. Eurographics Association; 2012. Shrec'12 track: Generic 3d shape retrieval; p. 119-126.
- López-Sastre R, García-Fuertes A, Redondo-Cabrera C, Acevedo-Rodríguez F, Maldonado-Bascón S. Evaluating 3d spatial pyramids for classifying 3d shapes. *Computers & Graphics.* 2013; 37(5): 473–483. URL: <http://www.sciencedirect.com/science/article/pii/S0097849313000551>.
- Meagher D. Geometric modeling using octree encoding. *Computer Graphics and Image Processing.* 1982; 19(2):129–147.
- mes (n.d.). [Accessed July 8, 2013] Meshlab. <http://meshlab.sourceforge.net>.
- Micali, S.; Vazirani, VV. Proc. of Symp. on Foundations of Computer Science. IEEE; 1980. An $O(\sqrt{|V|} |E|)$ algorithm for finding maximum matching in general graphs; p. 17-27.
- Min P. binvox, version 1.17, build #586. (n.d.) <http://www.cs.princeton.edu/~min/binvox/>.
- Mokbel, MF.; Aref, WG.; Grama, A. Proc. of the Intl. Conf. on Data Engineering. IEEE; 2003. Spectral LPM: an optimal locality-preserving mapping using the spectral (not fractal) order; p. 699-701.
- Mount, DM. ANN programming manual, version 1.1.2. 2010. <http://www.cs.umd.edu/mount/ANN/>
- Nooruddin FS, Turk G. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics.* 2003; 9(2):191–205. URL: <http://dx.doi.org/10.1109/TVCG.2003.1196006>.
- O'Boyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR. Open Babel: An open chemical toolbox. *Journal of Cheminformatics.* 2011; 3:33. [PubMed:21982300] [PubMed Central:PMC3198950] [doi:10.1186/1758-2946-3-33]. [PubMed: 21982300]
- Ohbuchi R, Nakazawa M, Takei T. Retrieving 3D shapes based on their appearance. Proc. of the Intl. Wksp. on Multimedia Info. Retrieval. 2003:39–45.
- Osada R, Funkhouser T, Chazelle B, Dobkin D. Shape distributions. *ACM T. on Graphics.* 2002; 21(4):807–832.
- Pagel, BU.; Six, HW.; Winter, M. Proc. of the Symp. on Principles of Database Systems. ACM; 1995. Window query-optimal clustering of spatial objects; p. 86-94.
- Paquet E, Rioux M, Murching A, Naveen T, Tabatabai A. Description of shape information for 2-D and 3-D objects. *Signal Processing: Image Communication.* 2000; 16(1):103–122.
- Paredes R, Chávez E, Figueroa K, Navarro G. Practical construction of k-nearest neighbor graphs in metric spaces. *Exp. Algo.* 2006:85–97.
- Redondo-Cabrera, C.; López-Sastre, R.; Acevedo-Rodríguez, J.; Maldonado-Bascón, S. Surfing the point clouds: Selective 3d spatial pyramids for category-level object recognition; *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*; 2012. p. 3458-3465.
- Samet, H. Foundations of multidimensional and metric data structures. Morgan Kaufmann; 2006.

- Sfikas K, Theoharis T, Pratikakis I. Rosy+: 3d object pose normalization based on pca and reflective object symmetry with application in 3d object retrieval. *Int. J. Comput. Vision.* 2011; 91(3):262–279. **URL:** <http://dx.doi.org/10.1007/s11263-010-0395-x>.
- shr (n.d.). [Accessed July 8, 2013] Shrec 2012 - shape retrieval contest based on generic 3d dataset. <http://www.itl.nist.gov/iad/vug/sharp/contest/2012/Generic3D/>.
- spr (n.d.). sproxel, r173. <http://code.google.com/p/sproxel/>
- Tangelder JWH, Veltkamp RC. A survey of content based 3D shape retrieval methods. *Multimedia Tools and Applications.* 2008; 39(3):441–471.
- Vleugels J, Veltkamp RC. Efficient image retrieval through vantage objects. *Pattern Recognition.* 2002; 35(1):69–80.
- Zhang J, Smith S. Shape Similarity Matching With Octree Representations. *Journal of Computing and Information Science in Engineering.* 2009; 9:034503.
- Zito T, Wilbert N, Wiskott L, Berkes P. Modular toolkit for data processing (mdp): a python data processing framework. *Frontiers in Neuroinformatics.* 2009; 2(8) **URL:** <http://www.frontiersin.org/neuroinformatics/10.3389/neuro.11.008.2008/abstract>.

Biographies

David Koes is an Assistant Professor in the Department of Computational and Systems Biology at the University of Pittsburgh and has a Ph.D. in computer science from Carnegie Mellon University. The current focus of his research is the development of novel algorithms for rapid and inexpensive computational drug discovery. He has developed and deployed a number of software packages and online systems that enable *interactive* drug discovery. See <http://www.pitt.edu/~dkoes> for more information.

Carlos J. Camacho, PhD, Associate Professor, Department of Computational and Systems Biology, University of Pittsburgh. Dr. Camacho's area of research is protein interactions. Highlights of his research include revealing the role of solvation in protein specificity, describing the role of anchor residues as key determinants of molecular recognition, understanding the role of intrinsic disordered proteins in regulation, and developing the first automated method to predict docked proteins *ClusPro*. More recently, the lab of Dr. Camacho has revolutionized rational drug discovery by developing the first interactive technologies to sample billion size compound libraries.

Fig. 1.

An illustration of a GSS-tree. Leaves of the tree contain voxelized shapes. Each internal node stores a maximum included volume (MIV) and a minimum surrounding volume (MSV) that bound the geometry of the shapes descended from the node.

Fig. 2.

A 2D illustration of minimum and maximum shape constraints. (a) A reference solid (solid) and receptor shape that it must be complementary to (gradient). (b) A minimum (white) and maximum (overlay) shape constraint derived from the reference. Only the contacting portion of the shape is used to define the minimum shape while the maximum shape is extended to include additional volume complementary to the receptor shape. (c) A shape matching these constraints. This shape is quite different from the reference shape when compared using a traditional similarity metric. (d) In contrast, a shape that is highly similar to the reference shape, but that violates the constraints of the receptor shape is less likely to be of interest.

Fig. 3. An illustrative GSS tree node (a) before and (b) after local adaptive rounding. The MIV/MSV shapes within a GSS-tree node are only rounded if all the shapes are the same or are full/empty. This promotes the hoisting of rounded leaves and reduces the overall height and memory requirements of the tree.

Fig. 4.

An illustration of the problems with partitioning around a central object. The light dotted line indicates potential clusters formed using such an approach, where all the objects are close to the center block, but not necessarily close to each other. This partitioning is imbalanced, with five objects in one cluster and only one in the other. More importantly, the corresponding MIV and MSV representations, top right, are less discriminating than those of an alternative partitioning, indicated by the dashed line. Unlike existing clustering methods, matching and packing explicitly optimizes for the discriminative ability of the MIV and MSV of each cluster.

Fig. 5.

The voxelized shape of a molecule (dark voxels) and its receptor (light voxels). This shape forms the basis for a minimum shape constraint while the inverse, or negative image, of the receptor defines a maximum shape constraint. Molecules whose surface falls in the gap between the two shapes match the constraints. Both shapes are shown after being reduced by the same gap distance: (a) 0Å, (b) 1Å, (c) 1.5Å, and (d) 2Å. Images generated with sproxel (spr n.d.).

Fig. 6. An example of optimal shape constraints. The (a) MIV and (b) MSV constraints obtained by taking the intersection and union, respectively, of the aligned shapes in the “Rocket” category of the SHREC 2012 benchmark.

Fig. 7.

An example of two very different molecular shapes that have been registered to the same coordinate system defined by a common molecular fragment, the planar 5–6 ring.

Fig. 8.
The effect of the approximate k -nn graph construction parameters k and v on index creation time and query performance (a) when IO-bound and (b) when CPU-bound.

Fig. 9. The effect of the number of clusters, c , and the size limit, p , at which a cluster is packed using the matching and packing algorithm on index creation time and query performance (a) when IO-bound and (b) when CPU-bound.

Fig. 10.
The effect of the maximum node fanout, m , on query performance when IO-bound and when CPU-bound.

Fig. 11.

The query performance of greedy packing, spectral packing, and clustering (no packing) compared to matching (a) when IO-bound and (b) when CPU-bound. All three packing algorithms were run using the complete graph of similarity distances instead of the k -nn approximation with $p = 2048$.

Fig. 12.

The query performance and index construction time of greedy packing and matching as the size limit, p , at which a cluster is packed is increased (a) when IO-bound and (b) when CPU-bound.

Fig. 13.

The discounted cumulative gain of the volume difference metric on the SHREC 2012 dataset. Performance for three different resolutions of voxelization are provided. Single pose performance considers only the volume difference with each aligned pose. Multiple poses performance considers the best volume difference of each aligned pose and its three axial rotations. Optimal shape constraints considers only the volume difference of those poses that satisfy minimum and maximum shape constraints that are derived directly from the category shapes.

Fig. 14.

The CPU-bound average query time of shape constraint (SC), k nearest neighbor (NN k), and linear scan (Scan) searches of the SHREC benchmark at a 64^3 voxelization resolution. (a) Query performance as the number of shapes is increased. (b) The box plot distributions of query times when querying the 1200 shape database.

Fig. 15.

The CPU-bound average query time of shape constraint (SC), k nearest neighbor (NN k) and linear scan (Scan) searches of the molecular shape benchmark. (a) Query time as the number of shapes is increased. (b) The box plot distributions of query times when querying the 2.2 million shape database. For clarity the linear scan box plot is omitted as it is of the scale with an average value of 45.7 seconds.

Fig. 16.

The average shape constraint query time for three different precisions of queries as the size of the database is increased by a factor of ten. Both (a) IO-bound and (b) CPU-bound average performance are shown. Query times are shown on a log scale. Error bars indicate the range spanned from the fastest to the slowest queries. Also indicated is the performance of a search performed using a full linear scan of the database instead of an indexing approach.

Table 1

Algorithm parameters.

Symbol	Description
k	nearest neighbors in k -nn graph
v	vantage points used to initialize k -nn graph
c	clusters created in top-down partitioning
p	threshold for packing a cluster
m	node fanout of GSS tree

Table 2

The shape recognition performance of the volume difference metric on the SHREC 2012 benchmark. Performance metrics are defined and computed as specified by the SHREC 2012 benchmark guidelines (shr n.d.). In all cases, values closer to one indicate better object recognition performance. The performance for three different resolutions of voxelization is provided. Single pose performance considers only a single normalized pose for each object, while multiple pose performance takes the best volume difference among the four poses available through axial rotations. Optimal shape constraints set the the distance of non-matching shapes to the maximum value. The published minimum, median, and maximum scores of the SHREC 2012 exercise are also provided as a reference.

		Nearest Neighbor	1-Tier	2-Tier	e-Measure	Discounted Cumulative Gain
Single Pose	256 ³	0.4367	0.1686	0.2351	0.1616	0.4980
	128 ³	0.4308	0.1687	0.2357	0.1618	0.4976
	64 ³	0.4333	0.1671	0.2314	0.1594	0.4952
Multiple Poses	256 ³	0.5358	0.2423	0.3275	0.2274	0.5686
	128 ³	0.5267	0.2412	0.3272	0.2268	0.5672
	64 ³	0.5175	0.2376	0.3214	0.2240	0.5641
Optimal Shape Constraints	256 ³	0.9475	0.8691	0.9471	0.6995	0.9538
	128 ³	0.9275	0.8166	0.9313	0.6849	0.9348
	64 ³	0.8800	0.7308	0.8764	0.6370	0.8923
SHREC 2012	Minimum	0.431	0.179	0.257	0.175	0.510
	Median	0.6965	0.3715	0.499	0.3485	0.682
	Maximum	0.879	0.661	0.799	0.576	0.871

Table 3

The average query time of shape constraint (SC), k nearest neighbor (NN k), and linear scan (Scan) searches of the 1200 shape SHREC benchmark for all three voxel resolutions.

Resolution	SC	NN1	NN2	NN4	NN8	Scan
256 ³	0.00503	0.11237	0.44692	0.51444	0.54276	0.53709
128 ³	0.00117	0.01889	0.05094	0.05361	0.05507	0.04974
64 ³	0.00062	0.00692	0.01745	0.01853	0.01914	0.01787