Shape Recognition in 3D Point-Clouds

Ruwen Schnabel Raoul Wessel Roland Wahl Reinhard Klein
Computer Graphics Group, University of Bonn
53117 Bonn, Germany
Email: {schnabel, wesselr, wahl, rk}@cs.uni-bonn.de

ABSTRACT

While the recent improvements in geometry acquisition techniques allow for the easy generation of large and detailed point cloud representations of real-world objects, tasks as basic as for example the selection of all windows in 3D laser range data of a house still require a disproportional amount of user interaction. In this paper we address this issue and present a flexible framework for the rapid detection of such features in large point clouds. Features are represented as constrained graphs that describe configurations of basic shapes, e.g. planes, cylinders, etc. Experimental results in various scenarios related to the architectural domain demonstrate the feasibility of our approach.

Keywords: shape detection, graph matching, retrieval, recognition, point-cloud

1 INTRODUCTION

The acquisition and processing of digital 3D point-clouds has received increasing attention over the last few years. While visualization of very detailed and complex point-clouds has become possible, interaction capabilities on a semantic level are still very limited. Even tasks as basic as selecting all windows in a scan of a house currently require a disproportional amount of user interaction. This is due to the fact that the acquired raw data does not provide any structural, let alone semantic, information. Therefore the extraction of semantic elements from 3D point clouds is an important topic for a wide field of applications, including architecture, cultural heritage and city model reconstruction. Further applications can be envisioned in the engineering context, e.g. supporting the automatic inventory of industrial plants or traditional reverse engineering processes.

The reasons for working on point-clouds are twofold: The first, and maybe most obvious, is that point-clouds are the native output format of laser range scanners. Moreover, point-clouds are an extremely general geometry representation as they contain no interpretation of the data. Therefore, by choosing point-clouds as the starting point of our method, we are able to process any type of 3D surface representation including polygon soups and meshes, since these representations are easily converted into point clouds by sampling.

Concerning the task of shape recognition in point-clouds, two problems arise:

- The raw point-cloud contains an overwhelming amount of redundant information making it virtually impossible to operate directly on the points themselves in an efficient manner.
- For each element, we need a model that the computer can retrieve in the data. However, defining a special parameterized model for each sought feature is challenging and very inflexible in case new entity types shall be detected.

In this work we focus on an architectural application domain. Based on the observation that most man-made objects in this domain can be decomposed into parts corresponding to geometric primitives like planes, cylinders, spheres, etc. we propose novel solutions for the above mentioned problems:

Decomposition By decomposing the point data into primitive shapes, we obtain an abstraction of the point data that eliminates much of the redundancy. A topology graph captures the neighborhood relations between the primitives in a concise manner.

Constrained subgraph matching Primitive shapes are assembled into configurations characteristic for higher semantic elements, e.g. windows, roofs or columns. Such configurations can be quickly retrieved from the topology graph via constrained subgraph matching.

Our system allows even unexperienced users to quickly formulate complex configurations, since primitive shapes are easily graspable and combinable. The search results are visualized in an interactive framework, allowing for refinement of the query.

Our method was tested on a large amount of data including point-clouds from different kinds of sensors like *LIDAR (light detection and ranging)* and stereo reconstruction. We also applied our method to 3D CAD

modeled buildings that include interior structures. The approach is robust against noise, clutter, registration errors or miscalibrations which are frequently encountered in 3D laser scans.

2 RELATED WORK

Our shape matching technique is related to many works in the larger context of (partial) matching, classification and retrieval of 3D shapes. Various approaches to these challenges have been developed in the past. In this brief overview we concentrate on methods for partial matching and retrieval of 3D objects in larger 3D scenes. For a more detailed introduction to 3D matching and shape retrieval we refer to [TV04].

2.1 3D city reconstruction

Since in this work we mainly deal with data from the architectural domain, automatic reconstruction of 3D buildings and city models from aerial LIDAR data, a special case of the above formulated tasks, is especially relevant to our approach. In this setting detection is restricted to simple shapes of which most can be described by configurations of planes. Automatic building reconstruction has been studied extensively in the photogrammetry community. Most of the developed semi-automatical or manual approaches rely on user interaction or on semantic knowledge that is not contained in the 3D point-cloud. For Example, Vosselman et al. integrate LIDAR data with a 2D Geographic Information System (GIS) database and aerial photographs [Vos02]. The GIS delivers ground plan information about buildings in the point cloud. With this information at hand, the points belonging to a single edifice can be extracted and parametric building models can be fitted.

A fully automatic approach that is only relying on the information contained in the point-cloud is presented in [VKH06]. There, a region growing approach is used to detect planes in the point data. Roof-topology graphs are defined to describe configurations of planes for some simple building forms shaped like I, L and U. These configurations are sought in the set of the detected planes. In a second step, the detected simple buildings are extended to more complicated forms according to the plane configurations in the point-cloud. Compared to our approach, there are two differences: First, as only planes are detected in the LIDAR data, the approach is restricted to those shapes that can be decomposed into planar patches. Second, the method does not use any node or graph constraints during the subgraph search and is therefore susceptible to misclassifications in more general settings.

2.2 Partial matching and retrieval of arbitrary objects

The retrieval and matching of 3D objects is of particular interest to the computer graphics community. The developed methods often rely on triangle meshes or parametric representations of the objects. Among the abundance of proposed approaches, several graph-based shape retrieval methods rely on the extraction of certain geometric components and use a graph to capture the relations between these components.

Model graph-based approaches are mainly used for geometry such as is generated in CAD applications. Model graphs describe solid objects in terms of connectivity of freeform surfaces (Boundary Representation) or as a set of geometric primitives that are connected by Boolean operations (Constructive Solid Geometry). Local clique matching [EM03] [EMA03] or comparison of graph spectra [MPSR01] are used to globally determine the similarity of the graphs, i.e. no partial matching is supported. In [ZTS02], VRML objects are segmented according to different decomposition techniques. The resulting patches are assigned basic shapes like planes and spheres. An attributed decomposition graph is built containing the determined shapes as nodes. Neighboring shapes are connected by edges. The similarity between two objects is computed by matching the associated decomposition graphs using error-correcting subgraph isomorphism.

Reeb graph-based methods rely on a function that is computed on the model surface. The surface is divided into segments corresponding to intervals of this function. A skeleton graph, in which the resulting segments are represented as nodes, is built. Reeb graph-based methods are mainly used for matching of articulated objects [HSKK01][TL07]. In [PSBM07], a robust method for fast Reeb graph computation is proposed that even allows for the use of non-manifold meshes.

Skeleton graphs of 3D shapes can be computed using topological skinning of voxel representations [BNdB99], medial axis transform, ridge point tracking [CSM05] or deformable model-based reconstruction [SLSK07]. Matching of two shapes is done by comparing the associated skeleton graphs using greedy bipartite graph matching [SSGD03] or by detecting subgraph isomorphisms using decision trees [LJI$^+$03].

3 OVERVIEW

The algorithm consists of two main steps: Firstly a shape based representation of the data is derived by detecting primitive shapes in the unstructured point data and constructing a *topology graph* that captures the neighborhood relations between the different shapes. Then, in the second step, this topology graph is searched for characteristic subgraphs corresponding to sought elements, as they have been defined by the user.

The user is able to quickly define and retrieve new entities with geometric constraints in an interactive framework. Moreover the detected subgraphs may contain optional or repetitive components, which further simplifies the definition of new entities for the user. This

way our method is very flexible and easily extensible, which renders it suitable in a broad range of applications.

4 SHAPE REPRESENTATION

The construction of a shape representation for the data constitutes the first step in our method. At this stage the fundamental building blocks are established that lay the ground for all further operations. A set of primitive shapes is detected, which is then subsumed in a topology graph.

4.1 Primitive Shapes

The aim of the shape detection is to find simple elements in the point-cloud which shall be the building blocks of more complex structures later on. We employ the algorithm presented in [SWK07] which recognizes planes, spheres, cylinders, cones and tori in the unstructured point-cloud. In this section, we will only give a very brief outline of the shape detection technique and the interested reader is referred to the original paper. The data is decomposed into disjoint sets of points, each corresponding to a detected shape proxy respectively, and a set of remaining points that consists of outliers as well as areas of more complex geometry for which primitive shapes would give an inappropriate representation. For further processing all remaining points are ignored. Points that are represented by a shape primitive are also called the *support* of a shape. Thus, given the point-cloud $P = p_1, \ldots, p_N$, the output of the shape detection is the following:

$$P = S_{\phi_1} \cup \ldots \cup S_{\phi_A} \cup R, \tag{1}$$

where each subset (the support) S_{ϕ_i} is associated with a shape primitive ϕ_i. All points in S_{ϕ_i} constitute a connected component and fulfill the condition

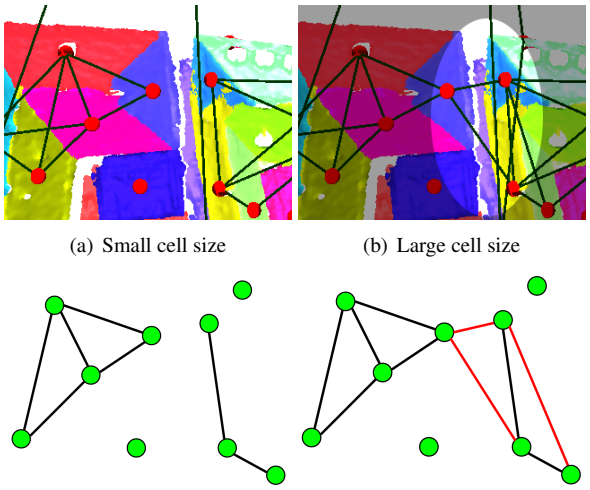
$$s \in S_{\phi_i} \Rightarrow \|s, \phi_i\| < \varepsilon \wedge \angle(n_s, n(\phi_i, s)) < \alpha, \tag{2}$$

where n_s is the normal of point s and $n(\phi_i, s)$ denotes the normal of the primitive ϕ_i at the point closest to s. The parameters ε and α are chosen by the user according to the precision of the acquisition device. The set R contains all the remaining, unassigned points.

4.2 Topology Graph

The topology graph $G(\Phi, E)$ describes the neighborhood relations between the primitive shapes detected in the point-cloud data. For each primitive ϕ_i a vertex is inserted into the graph, i.e. $\Phi = \phi_1 \ldots \phi_A$. Two shapes are connected with an edge if their supports are neighboring in space, i.e. the two vertices ϕ_i and ϕ_j are joined by an edge $e = (\phi_i, \phi_j)$ if

$$\exists p \in S_{\phi_i}, q \in S_{\phi_j} : \|p - q\| < t \tag{3}$$



(a) Small cell size (b) Large cell size

(c) Topology graph for small cell size (d) Topology graph for large cell size

Figure 1: Two houses viewed from above that are separated by a narrow alley. Primitive shapes have been detected and are depicted in random colors. a) The topology graph was built with a cell width of 50cm b) The cell width for the construction of the topology graph was set to 2m. Note that the roofs have been connected across the narrow alleyway. In c) and d) we show the resulting topology graphs. In d), the additional edges resulting from large cells are shown in red.

holds. Please note that computing the distance between the shapes directly and ignoring the support would result in many edges that have no counterpart in the data, since the shape primitives have indefinite extent.

Thus, to find the graph edges, the spatial proximity between the support of all detected shapes has to be determined. To this end we employ an axis aligned 3D grid. In a first step all points are sorted into the grid. Then for all grid cells that contain points belonging to different shapes, edges connecting the corresponding graph vertices are added to the graph, i.e. for each pair of shapes in the cell an edge is created. In order to avoid discretization dependencies due to the location of the grid cells, we use eight shifted and overlapping grids. Cells are stored in a hash table, so that memory is only allocated for occupied cells.

The width of the grid cells defines how far apart shapes are allowed to be in order to still get connected in the topology graph. Given the distance threshold t, the width of the cells is set to t and the shifted versions of the grid are created with an offset of $\frac{1}{2}t$ along the respective axes. Of course this means that shapes can get connected in some cases even though the distance between their support is in fact only less than $\sqrt{3}t$. It would be possible to eliminate these cases by checking the distance between the points in each cell, but we found this additional overhead unnecessary in practice,

as the few errors in the less restricted topology graph did not influence the performance of our algorithm. In Figure 1 resulting graphs are depicted for different cell widths.

Once the graph is complete the first step of our method is finished and a shape based representation of the point-cloud data has been derived. It can now be used to efficiently detect more complex configurations of primitive shapes that correspond to semantic entities in the data.

5 SHAPE MATCHING

In order to achieve an automatic matching between semantic entities and point-clouds, we have to find a common language for them. As we abstracted the point data to obtain a higher level description, we have to concretize the representation of feature elements in terms of primitive shape configurations.

5.1 Query graph

To this end we define a *query graph* for an element as a graph that captures its characteristic shape configuration. Basically a query graph is a topology graph with the difference that it does not stem from a point-cloud, but from knowledge about the shape of an element which is introduced by the user. The recognition of an element in the data then corresponds to a matching of the query graph to a subgraph of the topology graph. Even though subgraph matching is a NP-complete problem [Coo71], in our applications the query graphs will be small, i.e. usually less than twenty vertices, and a simple brute-force implementation of subgraph matching performs well in such a setting.

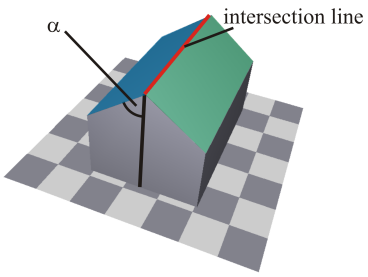


Figure 2: Illustration of the constraints that can be used to detect saddleback roofs: The angle α is constraint to be less than 90 degrees and similar for both planes. The intersection line is required to run parallel to the ground.

However, a representation solely based on topology is not sufficient to discriminate between many different feature elements. For example in the simple case of a saddleback roof (see Figure 2), the model graph consists of two vertices corresponding to planes connected by an edge. If such a graph is searched in a topology graph numerous false matches can be expected, as such a simple configuration occurs frequently. To make the detection more reliable, the user can add constraints to the query graph. For instance in the case of the saddleback roof we require the two planes to exceed a certain size, to be of similar size, and to intersect in a line that is parallel to the ground.

If we take a closer look at these geometric constraints, we find that they can be divided into classes that access different kinds of information. There are *node constraints* which only restrict the primitive shape associated with a node (e.g. type, size or orientation). There are *edge constraints* which restrict the relation between two incident shapes (e.g. angle between two planes). Any constraint not fitting into one of the first two classes belongs to the class of *graph constraints*, because it relies on the topology to be checked (e.g. sums of sizes, parallelism of disconnected planes).

Thus, when modeling a query graph the user specifies the sought shape configuration on a topological level by insertion of shape nodes and connecting edges. Geometric relations, however, are attached to these graph elements in the form of constraints which are formulated in a simple scripting language. The scripts have access to all parameters of the supported primitives as well as to the set of assigned points for each shape. Moreover predefined functions for computation of intersections, test for parallelism or orthogonality etc. exist.

5.2 Constrained subgraph matching

The outline of the recursive constrained query graph matching is illustrated in algorithm 1. To simplify the discussion of the procedure, no explicit statements are given for neither the maintenance of a data structure storing the matching, nor keeping track of visited nodes. However these actions are assumed to take place implicitly and it should be noted that they are mandatory for any correct implementation of the method. In the following the different parts of the algorithm will be described in detail:

In lines 2-8 the outer matching function is given, which searches for a suitable node in the topology graph, where the matching can be started. This outer matching function has to be started repeatedly to retrieve all possible matches.

Matching a node

In lines 10-18 the function for matching a node is sketched. First a check is made to see if all edges of the node have already been matched and if this is the case, the matching of the node has been successful (lines 11-14). Otherwise a yet unmatched edge of the node is chosen and matched to an edge of the topology graph by calling the MatchEdge function. If the edge

```
1:  Input A topology graph T = (V_T, E_T) and a query
    graph Q = (V_Q, E_Q)
2:  Function MatchSubgraph(Q, T)
3:  v_Q ← StartNode(V_Q)
4:  for all v_i ∈ V_T do
5:      if CheckNodeConstraint(v_Q, v_i) then
6:          if MatchNode(v_Q, v_i) then
7:              return true
8:  return false
9:
10: Function MatchNode(v_Q, v_T)
11: if v_Q has no unmatched edge then
12:     if all nodes in V_Q are matched then
13:         return CheckGraphConstraint()
14:     return true
15: e_Q ← first unmatched edge of v_Q
16: if MatchEdge(e_Q, v_T) then
17:     return MatchNode(v_Q, v_T)
18: return false
19:
20: Function MatchEdge(e_Q, v_T)
21: for all unmatched outgoing edges e_i of v_T do
22:     if CheckNodeConstraint(dest(e_Q), dest(e_i))
        then
23:         if CheckEdgeConstraint(e_Q, e_i) then
24:             if MatchNode(dest(e_Q), dest(e_i)) then
25:                 return true
26: return false
```

Algorithm 1: Recursive Constrained Subgraph Matching

was matched successfully, MatchNode is called recursively on the same node, in order to match any remaining edges of the node (lines 15-18).

Matching an edge

In lines 20-26 the MatchEdge function is outlined. It checks if any of the unmatched edges of the given topology graph's node can be matched to the given query graph edge (line 21-25). This is the case if the end-nodes of the edges can be matched successfully - which is tested via a call to MatchNode (line 24).

Checking constraints

Constraints are always verified just before a match is established (lines 5, 13, 22, 23). In line 13 the graph constraints are checked as soon as all nodes of the query graph have been matched. If this test fails, the matching will backtrack and continue the search. As can be seen, in contrast to graph constraints, both, node and edge constraints, have the advantage that they can be checked early on during the subgraph matching procedure, as they do not rely on other parts of the graph. This is an important performance factor since this way many of the topologically correct matches can be quickly discarded, without causing extensive backtracking.

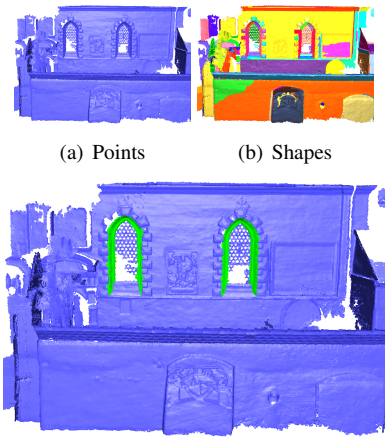
In order to avoid the need for graph constraints when using asymmetric edge constraints (e.g. sphere A has to be larger than sphere B) we also support directed edges as carriers of a constraint.

5.3 First results

At this point the methods presented so far are powerful enough to recognize large classes of semantic entities and we will first give a couple of examples illustrating the possibilities before presenting further extensions to the basic matching framework:

In Figure 3 a query graph was designed to detect Gothic windows in a scan of a medieval chapel. The windows were modeled as two spheres for the arches and two planes for the sides of the window. The spheres were constrained to have roughly equal radius and the planes to be tangential to the spheres.

To find the columns of the choir screen in Figure 4 they were modeled as a cylinder connected to two tori at both ends. The cylinder and the tori were constrained to posses the same axis of rotation (with a small tolerance of 5 degrees).



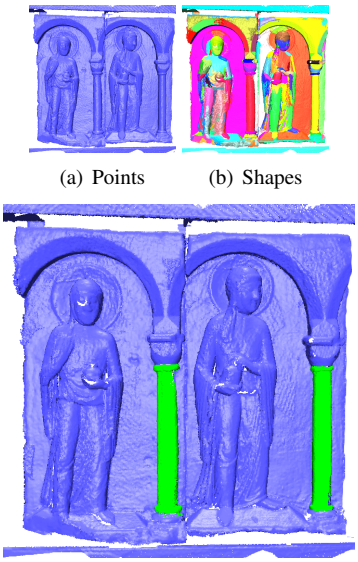
(a) Points (b) Shapes

(c) Windows

Figure 3: A scan of a medieval chapel with Gothic windows containing 4.2M points. The windows were detected by matching the query graph with subgraphs of the topology graph. In a) the original point-cloud is depicted. b) shows the support of the detected shape primitives in random colors. In c) the detected columns are highlighted in green.

5.4 Query Graph Extensions

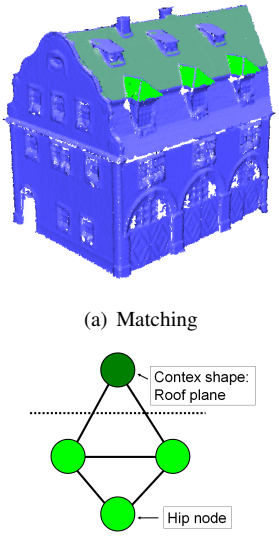
Although the given definition of a query already covers many shape configurations, there remain cases which it is still insufficient for. In the following we discuss some of these cases and demonstrate how they are overcome by extensions to the query graph model:



(a) Points (b) Shapes

(c) Columns

Figure 4: A scan of a choir screen consisting of 2M points. The query graph for the columns consisted of a cylinder connected to tori at both ends. In a) the original point-cloud is depicted. b) shows the support of the detected shape primitives in random colors. In c) the detected columns are highlighted in green.



(a) Matching

(b) Query graph

Figure 5: a) Detection of dormers on a roof. The roof plane shown in darker green is a context shape of the dormers. b) The query graph containing a context node.

Context nodes

Certain features benefit from a *context object* to distinguish them from other structures. For instance, a balcony needs a wall as context. Therefore we need to model the context in the query graph, but without declaring it an integral part of the balcony. This is achieved by tagging these query graph nodes as context nodes, so that after searching they can be removed from the match. In Figure 5 a) we give an example for this concept. There the roof planes have been modeled as the context shapes of the dormers, see Figure 5 b).

Optional nodes

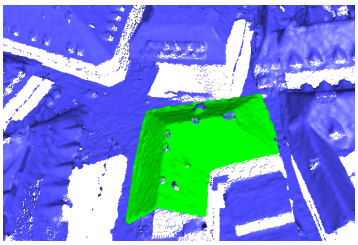
A limitation of the way we model queries so far is that we are not able to specify variants of an entity without duplicating the original query graph. For instance L-shaped roofs like the one shown in Figure 6 may occur in four variants: not hipped, hipped on either end or hipped on both ends. Thus a total of four query graphs, that only marginally differ, would have to be defined separately by the user. Since in practice this additional work may become quite burdensome, we augment the query graphs with what we call *optional nodes*. These nodes may be ignored by the matching procedure if it is unable to find any suitable counterparts in the given topology graph. To incorporate optional nodes, the matching procedure of Sec. 5.2 is extended in the following manner: First the matching is performed in the same way as described above, but ignoring any optional nodes. Then, for each matched instance of the query graph, as many optional nodes as possible are matched. To this end the graph traversal examines all possible matchings of the optional nodes but returns only those with the largest number of matches.

A problem arises if there are entire optional query graph components, instead of only single optional nodes. In such a case simply declaring all the nodes in question as optional could lead to incomplete matchings of the component. Therefore, we use a graph constraint which asserts the completeness of the matching.

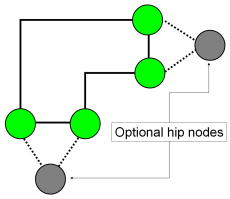
Although optional nodes increase the complexity of the search, they greatly reduce the number of required query graphs if different variants of a basic concept have to be detected. In Figure 6 the single hip of an L-shaped roof was matched by an optional node.

Multinodes

An even more complex case arises if we want to be able to model repetitive patterns like the steps of a stairway. In order to be able to model stairways with an arbitrary number of steps, a generic way of model extension is necessary. A simple approach is to define *multinodes* in the query graph that may match *several* different topology graph nodes. A multinode is defined as a query graph node that has a self-loop, i.e. an edge connecting the node with itself (this edge is implicitly considered optional by the matching algorithm). Via multinodes we are able to match arbitrarily large chains in the topology graph. This allows us to define a query graph



(a) L-shaped roof



(b) Query graph

Figure 6: An L-shaped roof may be hipped on either end. This is best modeled by optional nodes in the query graph. a) A matched L-shaped roof in a stereo reconstruction of a city containing 4M points. b) The query graph used for detection. Optional nodes are shown in grey.

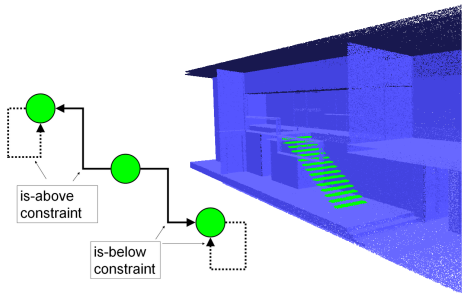


Figure 7: Detection of a stairway in a sampled CAD model of a house.

for stairways using only three nodes, as depicted in Figure 7. Note that we make use of directed edges in this example so that the multinodes do not need to match additional neighbor nodes each time the self-loop has been traversed (since the multinode does not have an outgoing edge other than the optional self-loop).

Query refinement

After the search for a query graph, the system presents the user with the results in an interactive framework that allows query refinement by changing constraints as well as query graph topology at runtime. As soon as an element of the query graph is modified, the new results are computed and displayed. This was achieved in real-time for all our tested examples.

6 CONCLUSION

Our shape detection system works on point-clouds so that we are able to work on data stemming from virtually arbitrary sources, such as terrestrial or airborne LIDAR data, polygon soups as well as ordinary meshes. As we mainly target applications in the architectural or cultural heritage domain, we safely assumed that most objects under consideration can be well represented by a set of primitive shapes. Thus we employ a fast primitive shape extraction method to effectively reduce the redundancy in the point-cloud and to derive a concise shape representation consisting of a topology graph on the shape primitives. In this graph, our system allows the detection of features that can be described as compositions of simple primitive shapes. Due to the simple structure of our representation it is not necessary for the primitive shape detection to output an optimal segmentation with a minimal number of primitives, nor to find the correct edges and transitions between different shapes. Only the detection of the relevant structures and their rough outlines has to be ensured.

Our system is unable to deal with cases for which features cannot be defined as configurations of primitive shapes, e.g. if trying to detect ornate frescos. However, we have demonstrated that for a wide range of frequently encountered structures our approach is very well suited and is able to deliver results as expected by the user. The user is able to specify the sought structures in a general way, even permitting fuzzy search within the limits of the graph constraints and the inclusion of optional components.

A potential drawback of our method could arise if large topology graphs with a lot of nodes and edges are used and at the same time the node and edge constraints of the query graph are chosen in a way that a wrong match will not be encountered early on during the matching procedure. Since the search for subgraph isomorphisms is a NP-hard problem, the retrieval performance might degenerate. However, such pathological cases are unlikely and in practice we observe very fast response times of the system, i.e. in the order of a few milliseconds (see timings in Table 1).

6.1 Future work

Future work concerning our method should address the improvement of usability. Up to now, the query graphs and the attached constraints are defined by hand. To make this process more comfortable for less experienced users we propose the development of a graphical user interface in which query graphs and constraints can easily be defined. A further step of research would be the automatic extraction of query graphs from modeled or scanned objects by means of statistical learning. Techniques like relevance feedback could be used to further enhance the retrieval performance.

dataset	#nodes	#edges	top. graph	matching
chapel (Figure 3)	232	406	1.8s	$<$ 10ms
choir screen (Figure 4)	537	2731	1.8s	$<$ 10ms
dormer roof (Figure 5)	106	138	0.27s	$<$ 10ms
city model (Figure 6)	431	351	2.5s	$<$ 10ms
CAD-house (Figure 7)	160	513	3.9s	$<$ 10ms

Table 1: Some statistics on test models. *#nodes* describes the number of primitive shapes that were found in the point cloud and by that the number of nodes in the resulting topology graph. *#edges* states how many edges describing neighborhood relations between the primitive shapes were found. *top.graph* shows how long it took to build the topology graph. The last column describes how much time was needed for matching the query graph.

Moreover, we plan to apply our system to basic point-cloud editing operations such as copy and paste of semantic units. Another interesting avenue of future research is exploiting the ability to detect self-similarities in the data for compression. Replacing instances of a query graph by generic representations might lead to very high compression ratios.

ACKNOWLEDGEMENTS

This work was partially funded by the German Science Foundation (DFG) under grant GZ 554975(1) Oldenburg BIB 48 OLof 01-02 *Probado* and as part of the bundle project "Abstraction of Geographic Information within the Multi-Scale Acquisition, Administration, Analysis and Visualization". The city model of Graz (Figure 6) is courtesy of German Aerospace Center (DLR) – Institute of Robotics and Mechatronics and was derived by semi-global matching from Vexcel Imaging Graz^{TM} imagery. The model of the rochus chapel (Figure 3) is courtesy of Zoller+Fröhlich GmbH.

REFERENCES

[BNdB99] Gunilla Borgefors, Ingela Nyström, and Gabriella Sanniti di Baja. Computing skeletons in three dimensions. *Pattern Recognition*, 32(7):1225–1236, 1999.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[CSM05] Nicu D. Cornea, Deborah Silver, and Patrick Min. Curve-skeleton applications. In *IEEE Visualization*, page 13. IEEE Computer Society, 2005.

[EM03] M. El-Mehalawi. A database system of mechanical components based on geometric and topological similarity. part ii: indexing, retrieval, matching, and similarity assessment. *Computer-Aided Design*, 35(1):95–105, January 2003.

[EMA03] Mohamed El-Mehalawi and Allen. A database system of mechanical components based on geometric and topological similarity. part i: representation. *Computer-Aided Design*, 35(1):83–94, January 2003.

[HSKK01] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proceedings of ACM SIGGRAPH*, 2001.

[LJI+03] K. Lou, S. Janyanti, N. Iyer, Y. Kalyanaraman, S. Prabhakar, and K. Ramani. A reconfigurable 3d engineering shape search system part ii: database indexing, retrieval and clusturing. In *DETC*, 2003.

[MPSR01] David McWherter, Mitchell Peabody, Ali C. Shokoufandeh, and William Regli. Database techniques for archival of solid models. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 78–87, New York, NY, USA, 2001. ACM Press.

[PSBM07] Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. Robust on-line computation of reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26(3):58, 2007.

[SLSK07] Andrei Sharf, Thomas Lewiner, Ariel Shamir, and Leif Kobbelt. On–the–fly curve-skeleton computation for 3d shapes. In *Eurographics*, pages 323–328, Prague, september 2007.

[SSGD03] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval, 2003.

[SWK07] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

[TL07] Gary K. L. Tam and Rynson W. H. Lau. Deformable model retrieval based on topological and geometric signatures. *IEEE TVCG*, 13(3):470–482, 2007.

[TV04] J. W. Tangelder and R. C. Veltkamp. A survey of content based 3d shape retrieval methods. In *SMI '04: Proceedings of the Shape Modeling International 2004 (SMI'04)*, pages 145–156, Washington, DC, USA, 2004. IEEE Computer Society.

[VKH06] Vivek Verma, Rakesh Kumar, and Stephen Hsu. 3d building detection and modeling from aerial lidar data. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2213–2220, Washington, DC, USA, 2006. IEEE Computer Society.

[Vos02] George Vosselman. Fusion of laser scanning data, maps, and aerial photographs for building reconstruction. In *Geoscience and Remote Sensing Symposium*. IEEE International, 2002.

[ZTS02] Emanuel Zuckerberger, Ayellet Tal, and Shymon Shlafman. Polyhedral surface decomposition with applications. *Computers and Graphics*, 26(5):733–743, October 2002.